Capítulo 8
Desarrollo de la aplicación cliente para los centros ambulatorios

8.1. Introducción

Esta aplicación sigue todos los pasos del estándar DICOM en la implementación de las funciones. Se ha querido realizar un programa multiplataforma y que esté de acuerdo con las especificaciones requeridas.

Como dijimos antes, el programa ha sido elaborado en Java en el entorno JBuilder con soporte de librerías SDK 1.4.2 y JDT de DICOM.

La aplicación tiene varias partes. Lo primero, y a través de los componentes de *javax.swing* es construir una GUI o Interfaz Gráfica de Usuario (*Graphic User Interface*) que cumpla los requisitos que se establecieron en el capítulo 2, que no eran otros que conseguir una interfaz amigable (fácil de rellenar, lógica, inteligible…) que además recogiese los mismos datos del paciente que el formulario que actualmente se

utiliza en los centros ambulatorios junto con la imagen de la lesión que motivó la consulta. Esto será lo que explicaremos en primer lugar.

En segundo lugar, dichos datos del paciente deben adaptarse a las pautas que marca el estándar DICOM, por lo que habrá que hacer que tomen el formato correspondiente a un archivo DICOM. Asimismo, habrá que implementar dos tipos de servicios DIMSE (vistos en el capítulo 5): el C-ECHO o de verificación de la conexión con el servidor y el C-STORE o de petición de almacenamiento del archivo en la base de datos.

El código completo de la aplicación se puede encontrar en el anexo B.

8.2. Interfaz gráfica de usuario

Hemos dividido la interfaz en dos partes para facilitar la inserción de los datos del paciente y seguir un orden lógico. En la mitad superior y mediante una instancia de la clase *JPanel* se encuentran los campos correspondientes a los datos personales del paciente, y en la mitad inferior hemos dispuesto varias pestañas que albergan paneles mediante las clases *JTabbedPane* y *JPanel* para el resto de los datos que se necesitan. De esta forma se puede poner en cada panel lo deseado. Podemos ver en la figura 8.1 el diseño de la interfaz definitiva, donde se aprecian los paneles que hemos mencionado.

Vamos a ver en primer lugar para qué sirven cada uno de estos paneles y una pequeña muestra de cómo han sido implementados.

8.2.1. Panel para datos personales

Se han incluido en él los mismos campos que contiene el formulario de consulta. Lo vemos en la figura 8.2.

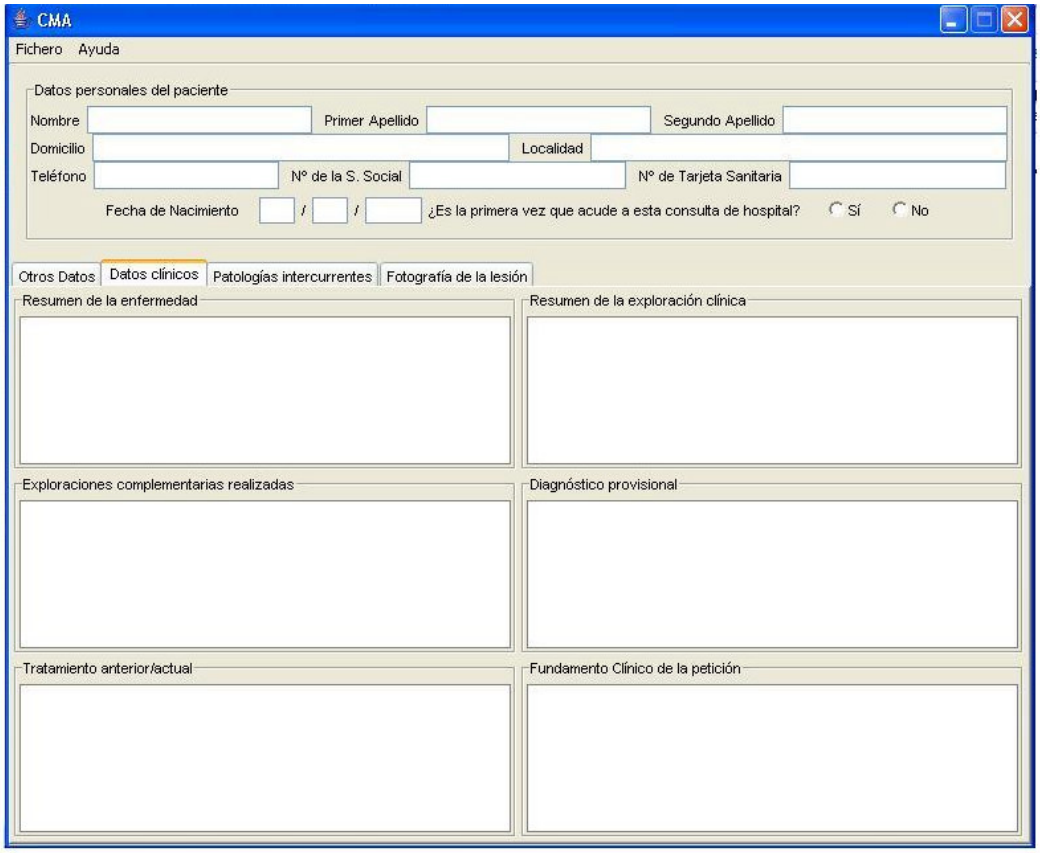

Figura 8.1. Diseño completo de la interfaz de usuario

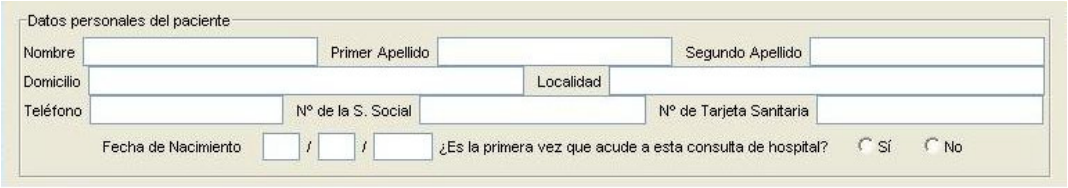

Figura 8.2. Panel de datos personales

A continuación, se indican los campos que se incluyen junto con una pequeña explicación de cómo han sido implementados.

- *Nombre*: Para el nombre de pila del paciente. Se emplea un campo de texto que está vacío inicialmente y que además estará habilitado para la escritura en todo momento. La forma de añadir cada campo y su correspondiente etiqueta en JAVA será la siguiente:

```
/**
* En primer lugar, habrá que declarar e inicializar los objetos
*  que se usan
*/

/** Campo para el nombre del paciente */
public static javax.swing.JTextField nombreCampo;
/** Etiqueta para el campo de nombre */
public javax.swing.JLabel nombre;

. . .

nombreCampo = new javax.swing.JTextField("");
nombreCampo.setEnabled(true);
nombre = new javax.swing.JLabel("Nombre");
nombre.setLabelFor(nombreCampo);

. . .


/** Ya sólo nos queda añadirlos en el panel correspondiente */

subpanel_1.add(nombre);
subpanel_1.add(javax.swing.Box.createRigidArea(new
   java.awt.Dimension(5,0)));
/** Con esto se fija la separación física entre la etiqueta y el
*   campo
*/
subpanel_1.add(nombreCampo);
```

El mecanismo de inserción para cada campo y su etiqueta correspondiente es el mismo, por lo que no lo repetiremos para todos y cada uno de los que nos quedan porque resultaría muy pesado. Conviene señalar también que para ayudar al facultativo en la inserción de los datos, se ha añadido la herramienta *Tool Tip Text* (texto de ayuda) de modo que cuando se sitúa el cursor del ratón sobre cualquier campo o etiqueta de la interfaz durante algo más de 2 segundos, aparece una leyenda indicando el dato que se debe insertar en ese campo y la forma de hacerlo si es necesario. Esto lo podemos ver en la figura 8.3.

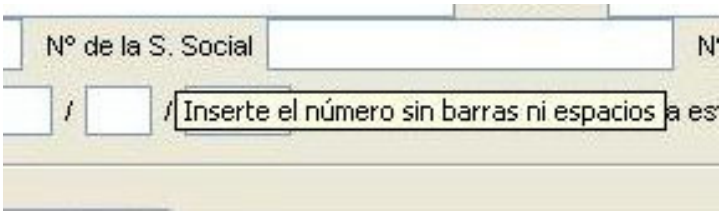

Figura 8.3. Herramienta de ayuda para el campo del Número de Afiliación a la Seguridad Social

La manera de implementar esta herramienta es muy sencilla, tan sólo hay que emplear el método *setToolTipText(String ayuda)* que incluyen casi todas las clases del paquete *javax.swing*, como son *JLabel* y *JTextField* que son las que aquí estamos utilizando. Se hace así:

```
nombreCampo.setToolTipText("Inserte el nombre del paciente");
nombre.setToolTipText("Inserte el nombre del paciente");
```

Esto también se repite para todos los campos y etiquetas que siguen a continuación. Los demás campos del panel son:

- *Primer apellido*: Campo de texto vacío inicialmente y habilitado permanentemente para el primer apellido del paciente.
- *Segundo apellido*: Campo de texto vacío inicialmente y habilitado permanentemente para el segundo apellido del paciente.
- *Domicilio*: Campo de las mismas características que los anteriores para el domicilio habitual del paciente.
- *Localidad*: Campo semejante a los anteriores para la localidad en la que reside el paciente.
- *Teléfono*: Campo de texto habilitado de modo permanente y de 9 caracteres para el número de teléfono de contacto del paciente.
- *Nº de la S. Social*: Campo de texto de 12 caracteres para que el facultativo inserte el número de afiliación a la Seguridad Social del paciente sin barras ni espacios, tal y como se indica en el CMBDA (Conjunto Mínimo Básico de Datos de Andalucía). [20]
- *Nº de Tarjeta Sanitaria*: Campo de texto de 12 caracteres para insertar el número de tarjeta sanitaria del paciente que disponga ya de ella. Se escribirá dicho número sin barras ni espacios, tal y como se indica en el CMBDA.
- *Fecha de nacimiento*: En realidad se compone de tres campos de texto para la fecha de nacimiento del paciente con el siguiente formato: día / mes / año. Todo en formato numérico de dos caracteres para el día y el mes y cuatro para el año.
- Lo único que nos queda por señalar es que hemos añadido dos casillas de verificación en las que se indicará si es o no la primera vez que el paciente que se está tratando acude a una consulta de este tipo. Esto se implementa mediante las clases *Checkbox* y *CheckboxGroup*, ambas del paquete *java.awt.* así:

```
/** Declaración de las clases */

public java.awt.CheckboxGroup vecesCons;
public java.awt.Checkbox si;
public java.awt.Checkbox no;
public javax.swing.JLabel veces;

. . .

/** Inicialización */

vecesCons = new java.awt.CheckboxGroup();
veces = new javax.swing.JLabel("¿Es la primera vez que acude a
      esta consulta de hospital?");
si = new java.awt.Checkbox("Sí",false,vecesCons);
no = new java.awt.Checkbox("No",false,vecesCons);

. . .

/** Se añaden ahora al panel */

subpanel_5.add(veces);
subpanel_5.add(javax.swing.Box.createRigidArea(new
      java.awt.Dimension(10,0)));
subpanel_5.add(si);
subpanel_5.add(javax.swing.Box.createRigidArea(new
      java.awt.Dimension(10,0)));
subpanel_5.add(no);

. . .
```

Los campos de este panel tienen que estar rellenos para que el archivo DICOM se pueda generar, con excepción del número de afiliación a la Seguridad Social y el de Tarjeta Sanitaria, ya que para estos campos sólo es necesario rellenar uno de los dos. Si cuando se solicite crear el archivo DICOM hay algún campo vacío, saldrá un mensaje de error. Esto lo veremos con detenimiento en el apartado correspondiente.

8.2.2. Panel para datos clínicos

Este panel es uno de los correspondientes al JTabbedPane de la mitad inferior de la interfaz de usuario. Lo podemos ver en la figura 8.4 y contiene 6 áreas de texto para insertar datos correspondientes a la exploración realizada por el facultativo, si el paciente sigue o ha seguido algún tratamiento, un diagnóstico de la lesión… A continuación, los estudiaremos con algo más de detenimiento.

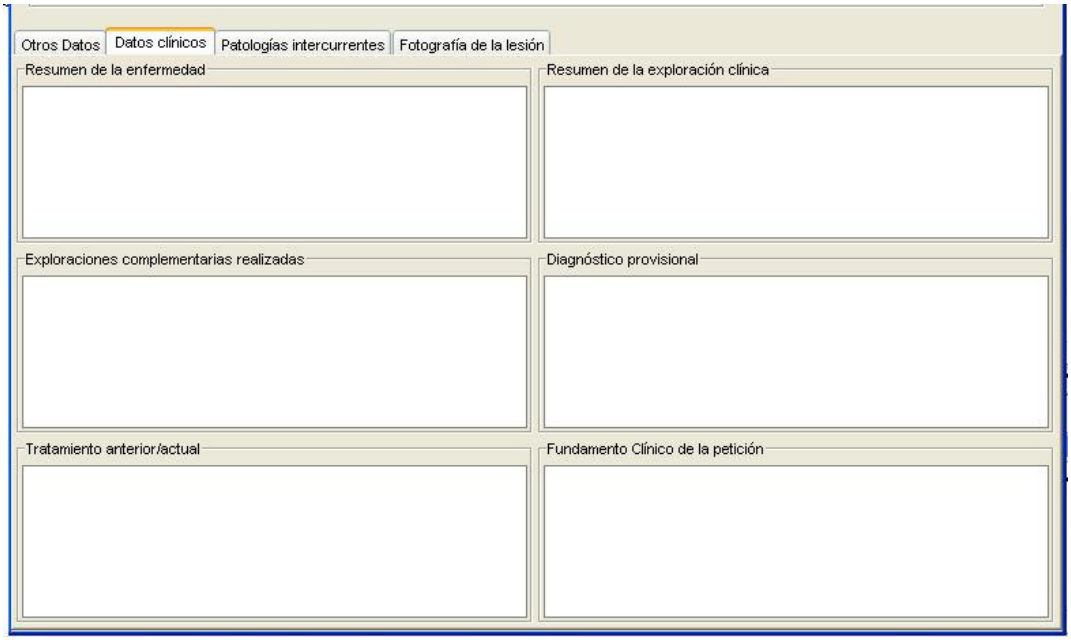

Figura 8.4. Panel para datos clínicos del paciente

Los campos son:

- Resumen de la enfermedad.
- Resumen de la exploración clínica.
- Exploraciones complementarias realizadas.
- Diagnóstico provisional.
- Tratamiento anterior / actual.
- Fundamento clínico de la petición.

En el caso de estos campos, es opcional el rellenarlos todos o no, ya que el facultativo podría considerarlo como innecesario. Por este motivo, cuando se solicite la creación del archivo DICOM, la aplicación comprobará si hay datos en alguno de estos campos y si los hay, los incluye en el objeto, mientras que si no los hay, continuará su ejecución sin añadir modificaciones al mismo.

A continuación, vamos a ofrecer una breve explicación de la forma en que se ha implementado esta parte de la aplicación. Para insertar datos en forma de texto, lo más conveniente era utilizar la clase *JTextArea* (área de texto) para cada uno de los campos y posteriormente, incluir cada *JTextArea* dentro de un *JScrollPane* por si los datos

ocupan más que el área de texto que aparece en pantalla. Con esto, aparecerá una barra de desplazamiento vertical en el momento en que sea necesaria. Además, se habilitará el salto de línea cuando sea necesario (así no será necesaria la barra de desplazamiento horizontal) de manera que tampoco se corten las palabras escritas. Además, estas áreas de texto se encuentran permanentemente habilitadas y también incluyen la herramienta de ayuda. Veamos un ejemplo de cómo se implementará una de estas áreas de texto:

```
/** Declaraciones de las clases */
/** Área de texto para resumen de la enfermedad */
public static javax.swing.JTextArea resumenEnfermedad;
/** para escrollar el área de texto */
public javax.swing.JScrollPane resEnfScrollPane;

. . .

/** Inicializaciones */
resumenEnfermedad = new javax.swing.JTextArea();
resumenEnfermedad.setToolTipText("Escriba aquí un resumen de la
      enfermedad");
resumenEnfermedad.setEnabled(true);
resumenEnfermedad.setEditable(true); // dejamos editar
// configuramos que salte de linea si no cabe, sin cortar las
// palabras
resumenEnfermedad.setLineWrap(true);
resumenEnfermedad.setWrapStyleWord(true);
// le añadimos un borde al área de texto
resumenEnfermedad.setBorder(javax.swing.BorderFactory.
      createLineBorder(java.awt.Color.gray));
// incrustamos en un panel de scroll
resEnfScrollPane = new
      javax.swing.JScrollPane(resumenEnfermedad);
// configuramos barra de desplazamiento vertical
resEnfScrollPane.setVerticalScrollBarPolicy(
 javax.swing.JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);

. . .

/**
*   Ya sólo nos quedaría añadirlo al panel que corresponda
*   mediante el método add(), tal y como se vio en el apartado
*   anterior
*/
```

8.2.3. Panel para otros datos

Es otro de los paneles contenidos en las pestañas del JTabbedPane de la mitad inferior de la interfaz y sirve para insertar en él otros datos de interés referentes al paciente, tales como los referentes al médico que le atendió, si padece alguna patología que inmediatamente lo hiciese no apto para CmA, el centro sanitario al que se envía la

petición y la forma en que el paciente desea ser avisado del resultado de la solicitud de consulta. Podemos ver este panel en la figura 8.5.

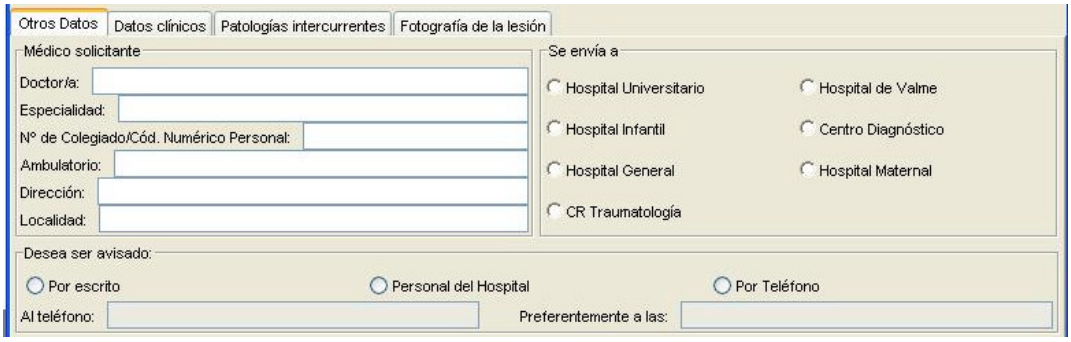

Figura 8.5. Panel para otros datos del paciente

Como se puede observar, este panel a su vez se encuentra subdividido en cuatro subpaneles:

Subpanel Médico Solicitante

Para insertar en él los datos correspondientes al facultativo que solicita la interconsulta. Se implementa al igual que en el caso del panel para datos personales con un conjunto de campos de texto vacíos y permanentemente habilitados con sus correspondientes etiquetas y su herramienta de ayuda. Los campos que se incluyen son:

- *Doctor / a*: Para el nombre y apellidos del facultativo solicitante.
- *Especialidad*
- *Nº de Colegiado / Código Numérico Personal*: Es un campo de texto de 8 dígitos que se podrá consignar de dos maneras, según el CMBDA [20]:

    - Mediante el Número Oficial de Colegiado, que es como se ha venido haciendo hasta ahora.
    - Mediante el Código Numérico Personal (CNP) de identificación individual del Sistema Sanitario Público de Andalucía (SSPA). En este caso se asignarán los siete dígitos centrales del CNP precedidos del 9.

- *Ambulatorio*: centro desde el que se realiza la interconsulta.

- *Dirección*.
- *Localidad*.

Todos estos campos y etiquetas se implementan de forma similar a como vimos anteriormente.

Subpanel Centro de Destino de la petición

Se trata de un conjunto de casillas de verificación (*CheckboxGroup*) en el que sólo hay que seleccionar el centro al que el facultativo quiere enviar su petición. Los posibles centros a los que se puede enviar son:

- Hospital Universitario.
- Hospital Infantil.
- Hospital General.
- CR Traumatología.
- Hospital de Valme.
- Centro de Diagnóstico.
- Hospital Maternal.

Ya hemos analizado la forma en que hay que implementar este tipo de objetos.

Subpanel Modo de Aviso

En este caso también hay un conjunto de casillas de verificación en el que se seleccionará la forma en que el paciente desearía ser avisado del resultado de la interconsulta, así como de la fecha y hora de su intervención en el caso correspondiente. El paciente puede elegir ser avisado por:

- Escrito.
- Personal del hospital.
- Teléfono.

A continuación, se incluyen dos campos de texto que en principio están deshabilitados. Sólo se habilitan si el paciente pide ser avisado por teléfono, en cuyo caso se rellenaría el primer campo con un número de teléfono de contacto y el segundo con la hora del día a la que el paciente prefiere ser llamado.

La forma de deshabilitar un campo de texto no es otra que utilizar el método *setEnabled (boolean f)* que contiene la clase *JTextField* con el valor "*false*" en la variable booleana que se le pasa como parámetro a dicho método, es decir:

```
porTfnoCampo.setEnabled(false);
```

8.2.4. Panel para patologías intercurrentes

Este es el tercer panel de los que se encuentran en el JTabbedPane de la mitad inferior de nuestra interfaz. Sirve fundamentalmente para que el facultativo complete los datos referentes a ciertas patologías que pudiera sufrir el paciente y que directamente podrían clasificarlo como no apto para CMA. Lo podemos observar en la figura 8.6.

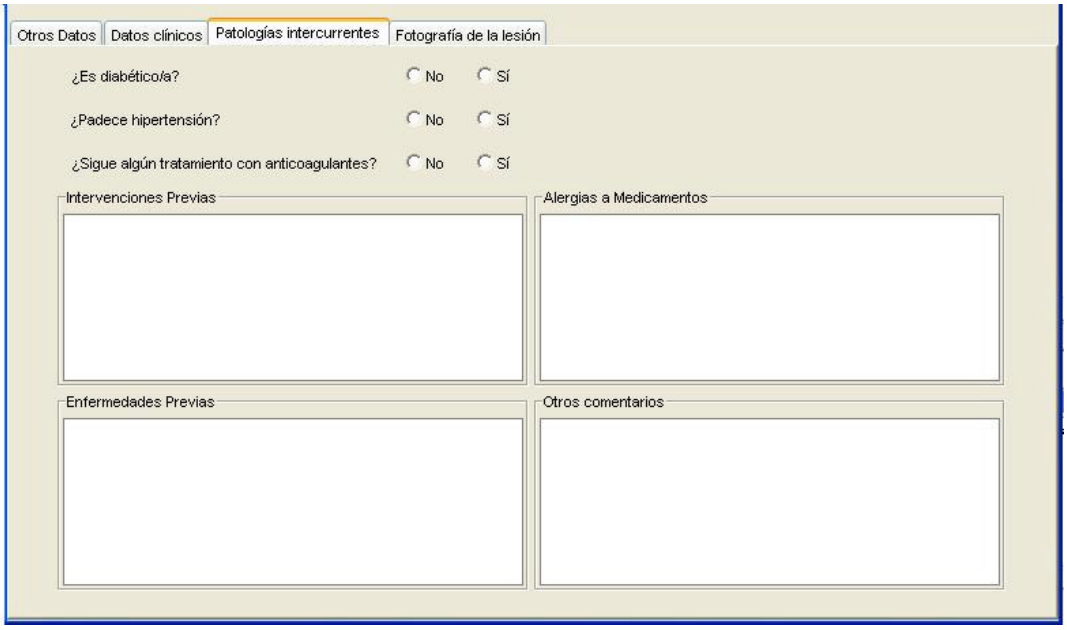

Figura 8.6. Pantalla para patologías intercurrentes

Los campos que comprende este panel son los siguientes:

- ♣ Conjunto de casillas de verificación (clase CheckboxGroup de java.awt) para marcar si el paciente padece diabetes o no.
- ♣ Conjunto de casillas de verificación para marcar si el paciente sufre hipertensión o no.
- ♣ Conjunto de casillas de verificación para marcar si el paciente está siguiendo algún tipo de tratamiento con anticoagulantes (como por ejemplo, aspirinas).
- ♣ Área de texto para insertar intervenciones quirúrgicas previas (ya fuesen de CmA o no) a las que fue sometido el paciente.
- ♣ Área de texto para insertar alergias a medicamentos que pudiese tener el paciente.
- ♣ Área de texto para insertar enfermedades previas de importancia que pudiese haber padecido el paciente.
- ♣ Área de texto adicional por si el facultativo quisiese añadir algún comentario más.

La implementación de todas estas clases ya se ha visto, por lo que no haremos más hincapié en ellas. No obstante, cabe señalar que todos estos campos están permanentemente habilitados y con sus correspondientes herramientas de ayuda y estilo implementadas. Hay que decir también que igual que en el caso de los campos del panel anterior, tampoco es obligatorio que estén rellenos todos, por este motivo en caso de que no se inserte ningún dato en este panel, en la central del HUVR se supondría que el paciente no padece ninguna de estas patologías que podrían hacerlo no apto para CmA.

8.2.5. Panel para la imagen de la lesión

Este es el último panel que se incluye en el *JTabbedPane* de la mitad inferior de la interfaz. Desde este panel se seleccionará la imagen de la lesión del paciente que quiere enviar el facultativo, se comprueba el estado de la conexión con el servidor (servicio DIMSE C-ECHO), se crea el archivo DICOM y se envía a través de la red para su almacenamiento, se puede salir de la aplicación y también limpiar los datos escritos en los campos de la interfaz. Podemos observar cómo quedará cuando todavía no se ha seleccionado ninguna imagen en la figura 8.7.

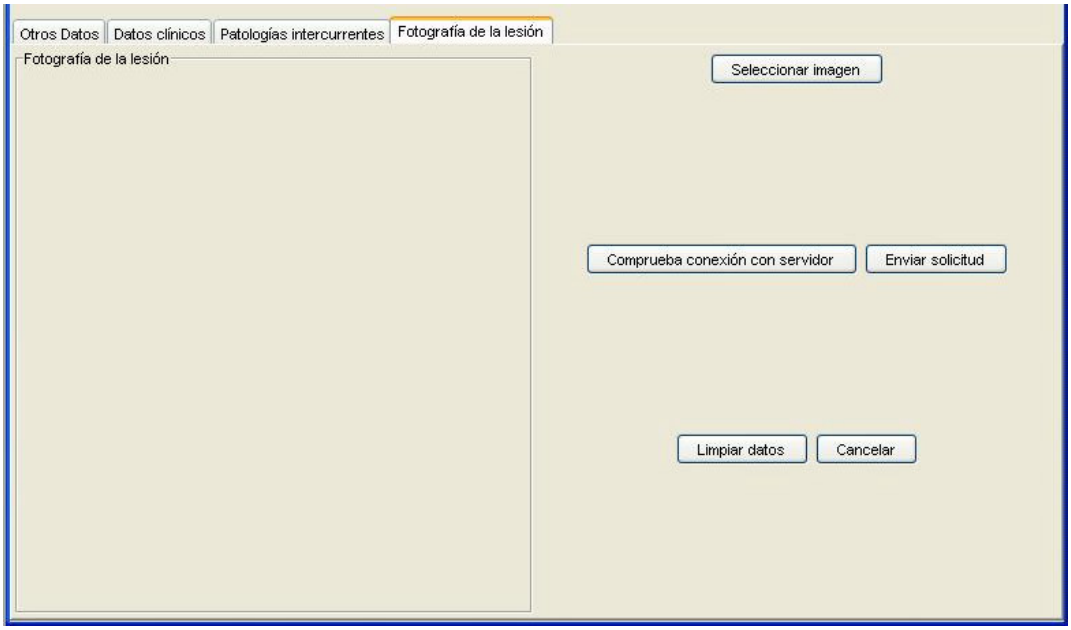

Figura 8.7. Panel para añadir la imagen de la lesión

Este panel se subdivide a su vez en dos: en la mitad izquierda tenemos el visor de la imagen que en la figura 8.7 aparece en blanco porque todavía no hemos seleccionado ninguna imagen. En la mitad derecha tenemos un conjunto de botones (implementados con la clase *JButton* de *javax.swing*) que sirven para realizar las acciones que hemos mencionado antes. La manera de implementar cada botón es muy sencilla, sólo hay que declarar e inicializar la instancia de forma similar a como venimos haciendo con otras clases y después añadirle un oyente de eventos *ActionListener* (de la clase java.awt.event) para que cuando se pulse el botón con el ratón, se invoque al método que ejecuta la acción correspondiente a dicho botón:

```
/** Declaración de la clase y el oyente del evento */
public javax.swing.JButton cancelar;
OyenteEventoCancelar oyente5 = new OyenteEventoCancelar();

. . .

/** Inicializamos */
cancelar = new javax.swing.JButton("Cancelar");
/** Asignamos al botón su oyente de eventos */
cancelar.addActionListener(oyente5);

. . .

/** En la clase que sigue es donde se implementa el oyente */
class OyenteEventoCancelar implements
```

```
            java.awt.event.ActionListener{
            public void actionPerformed(java.awt.event.ActionEvent e) {
                 System.exit(0); // Significa salir de la aplicación
    }
  }
```

Los botones implementados cumplen las siguientes funciones:

- *Botón" Seleccionar imagen"*: si vamos por partes, lo primero que habrá que hacer es seleccionar la imagen una vez que se han rellenado todos los campos anteriores. En este momento, el colegiado pulsará este botón, tras lo cuál se abrirá un cuadro de diálogo de selección de archivos, como se puede observar en la figura 8.8.

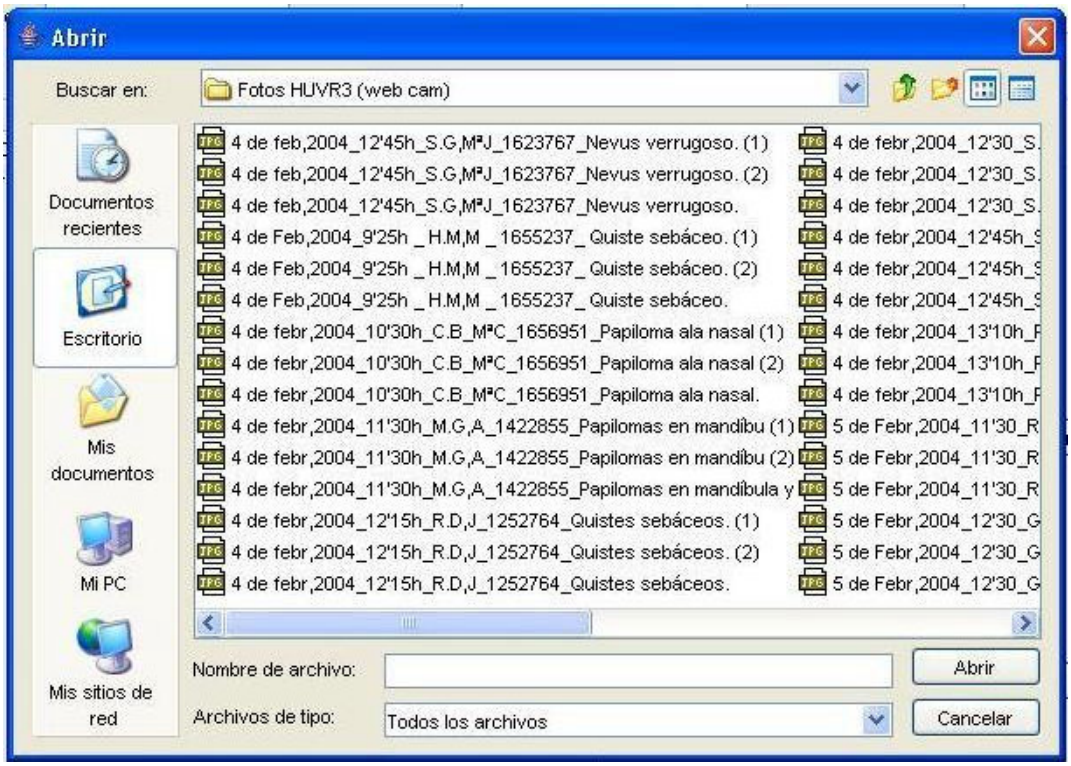

Figura 8.8. Cuadro de diálogo para selección de una imagen

Una vez que hayamos seleccionado la imagen de la lesión, dicha imagen aparecerá en el visor de la mitad izquierda del panel como se puede comprobar en la figura 8.9; además, tal y como se ve en la figura 8.10, en la línea inferior derecha del panel aparece una etiqueta con el nombre de la imagen que se ha escogido. Si en primer término nos confundimos en la selección de la imagen, no pasa nada, ya que se puede

seleccionar otra imagen sin que la aplicación dé problemas. El visor y la etiqueta con el nombre de la imagen cambiarán y podremos continuar con la ejecución normal del programa.

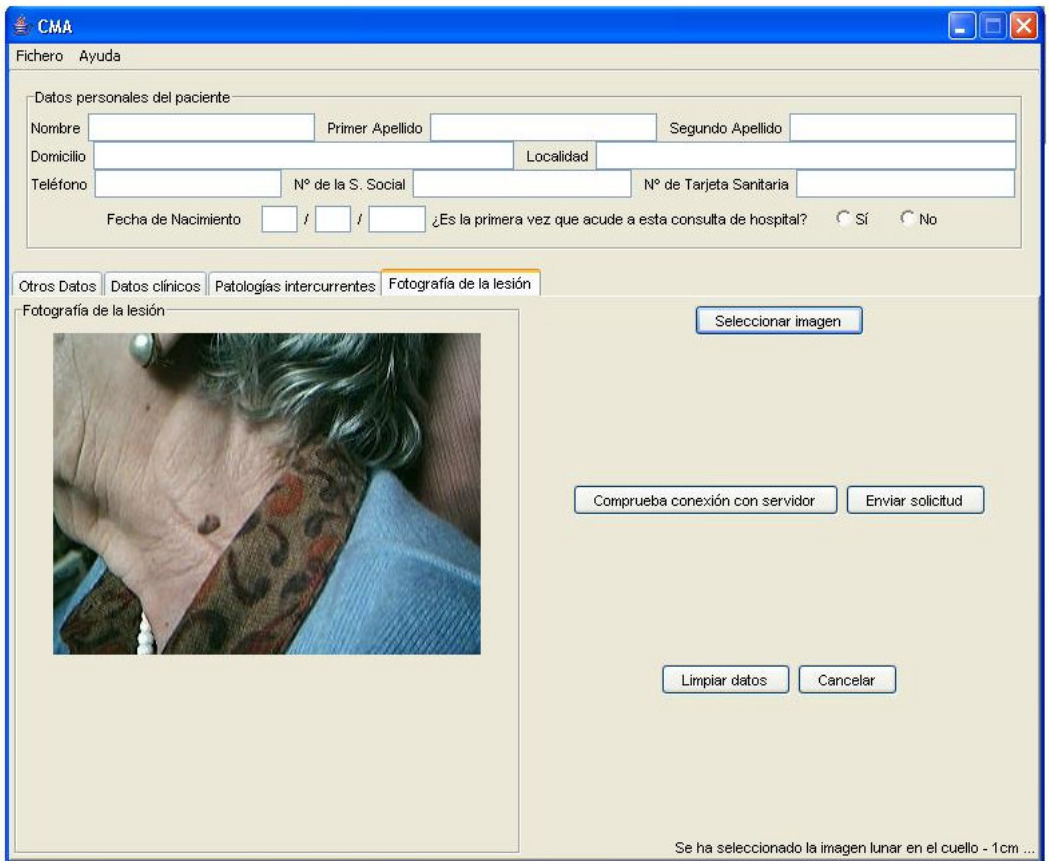

Figura 8.9. Interfaz con la imagen seleccionada

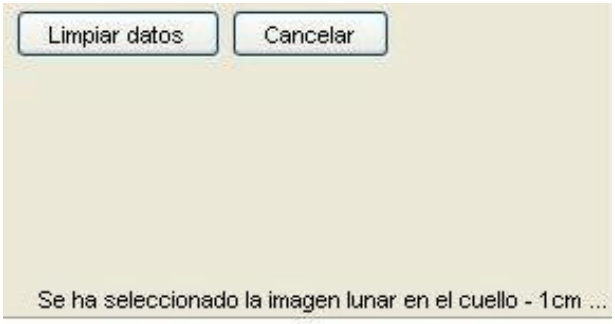

Figura 8.10. Etiqueta con el nombre de la imagen seleccionada

El oyente del evento que hemos creado para este botón lo que hace es utilizar una instancia de la clase JFileChooser de javax.swing para crear el selector de archivos y extraer el path de la imagen seleccionada. Con esto crea un objeto de la clase ImageIcon (también en javax.swing) que será el responsable de que la imagen aparezca en el visor. Se implementó de la siguiente forma:

```
/** Oyente del evento */
class OyenteEventoBoton implements
      java.awt.event.ActionListener{public void actionPerformed
       java.awt.event.ActionEvent evento){

       javax.swing.JFileChooser chooser = new
            javax.swing.JFileChooser();
       int returnVal = chooser.showOpenDialog(parent);
       if(returnVal == javax.swing.JFileChooser.CANCEL_OPTION)
            parent.getParent();
       if(returnVal == javax.swing.JFileChooser.APPROVE_OPTION){
            /** Toma el path de la imagen */
           nuevaFoto = chooser.getSelectedFile().getPath();
           marcoFoto.setIcon(dameFoto(nuevaFoto));
           fotodic = new
           javax.swing.ImageIcon(dameFoto(nuevaFoto).getImage());
            /** Para que aparezca la etiqueta del nombre */
           nombreFoto.setText("Se ha seleccionado la imagen " +
            chooser.getSelectedFile().getName());
       }
}}

/**
*  El siguiente método devuelve un objeto de la clase ImageIcon
*  a partir del path de la imagen seleccionada
*/

javax.swing.ImageIcon dameFoto(String cadena){
      javax.swing.ImageIcon f = new
            javax.swing.ImageIcon(cadena);
return f;
}
```

- *Botón "Comprobar conexión con el servidor"*: el oyente de este botón se encargará de implementar el servicio de verificación de conexión C-ECHO de DICOM en el papel de SCU o usuario de clase de servicio (*Service Class User*). Lo estudiaremos en detalle en el apartado 8.4.1.
- *Botón "Enviar Solicitud"*: cuando se pulsa este botón, el oyente hace lo siguiente: en primer lugar, crea una instancia SOP DICOM (par objeto-servicio) con el UID (identificación) correspondiente a una instancia de almacenamiento segundario, lo veremos en el apartado 8.3. A continuación, envía dicha instancia a través de la red para su almacenamiento en la base de datos, es decir, realiza el

papel de SCU para el servicio de almacenamiento C-STORE de DICOM. Esto último será analizado en el apartado 8.4.2.

- *Botón "Limpiar Datos"* : Este botón sirve fundamentalmente para limpiar los datos de un paciente de una manera rápida cuando ya se ha enviado su archivo al servidor. De este modo, podremos dar cabida a los datos de un paciente nuevo sin tener que salir de la aplicación y volver a entrar o borrar uno por uno todos los campos.
- *Botón "Cancelar"* : Es el correspondiente al oyente que pusimos de ejemplo antes. Este botón lo único que hace es salir de la aplicación sin guardar cambios en el fichero del paciente ni enviar nada mediante el método *System.exit(0)* que se encuentra en *java.lang*.

Cabe destacar también que hemos añadido accesos directos a todos los botones mediante la siguiente línea del programa:

```
cancelar.setMnemonic(java.awt.event.KeyEvent.VK_C);
```

De esta manera, con tan sólo pulsar en el teclado del ordenador Alt + C ya accedemos al oyente correspondiente, que en este caso de ejemplo supondría salir de la aplicación. Los accesos directos para el resto de los botones son:

- Botón "Seleccionar imagen": Alt + S.
- Botón "Comprobar conexión con el servidor": Alt + O.
- Botón "Enviar solicitud": Alt +E.
- Botón "Limpiar datos": Alt + L.
- Botón "Cancelar": Alt + C.

8.3. Creación de un Objeto DICOM

Una vez que se completa la inserción de los datos del paciente por parte del facultativo, habrá que enviar el formulario al servidor que se encuentra en el HUVR para su almacenamiento, pero dicho formulario no puede viajar de cualquier manera por la red, tendrá que cumplir las especificaciones del estándar DICOM para que el servidor pueda hacerse cargo del fichero. Como ya hemos dicho, cuando se pulsa el botón

"Enviar solicitud" del panel para la image n de la lesión del paciente, se desencadenan dos acciones: en primer lugar, se comprueba si los campos necesarios de la interfaz están rellenos (en caso contrario, aparece un mensaje de error como el que vemos en la figura 8.11) y en caso de que lo estén se genera un fichero compatible a los requerimientos del estándar (todo esto es lo que analizaremos en este apartado) y finalmente, se envía dicho fichero en una instancia de petición de almacenamiento al servidor (esto lo veremos en el punto 8.4.2).

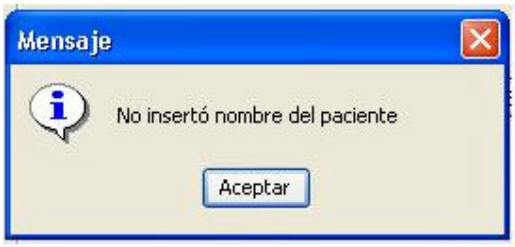

Figura 8.11. Mensaje de error que aparece si falta alguno de los campos por rellenar

Los datos que se insertarán en el archivo DICOM son de dos tipos: datos de texto (todos los referentes al paciente) por una parte, y por otra, datos de una imagen, la de la lesión para la que queremos hacer la clasificación. Además, dentro de los datos de texto tenemos que distinguir entre aquellos campos que son recogidos por el estándar (como el nombre del paciente, la dirección, el teléfono…) que analizaremos en el apartado 8.3.2 y aquellos otros que no lo son y que recogemos en el apartado 8.3.3.

8.3.1. Inserción de datos de imagen

Para crear el archivo DICOM, utilizamos la clase *ImageDicom*, que se encuentra dentro del paquete *com.archimed.dicom.image* (proporcionado en las librerías JDT a cuyo estudio se dedicó el capítulo 7 de la memoria). Esta clase es heredera de la clase *DicomObject*, lo que nos facilitará la creación del archivo DICOM que estamos tratando de conseguir. La forma de insertar los datos de imagen en el archivo DICOM será por tanto la siguiente:

```
/**
*  Constructor de la clase
*/
```

```
public ObjetoDicom(){

try{
      image.PixelGrabber grabber;
      Image imagenJPG = PanelFotografia.fotodic.getImage();
      grabber = new image.PixelGrabber (imagenJPG,0,0,
       imagenJPG.getHeight(null),imagenJPG.getWidth(null),true);
      try {
        grabber.grabPixels();
      }catch{. . .}

      int[] pix = (int[])grabber.getPixels();
      dcmNuevo.set(com.archimed.dicom.DDict.dSOPClassUID,
        "1.2.840.10008.5.1.4.1.1.7");
      dcmNuevo.set(com.archimed.dicom.DDict.dSOPInstanceUID,
        "1.2.840.10008.5.1.4.1.1.7");

      dcmNuevo.imagePixelData(imagenJPG.getWidth(null),
        imagenJPG.getHeight(null),0,pix);

      . . .
}
```

Como podemos ver, la imagen que se encuentra seleccionada en el panel para la fotografía de la lesión se almacena en una instancia de la clase *Image* de *java.awt* (imagenJPG). Después y gracias a la clase *PixelGrabber* de *java.awt.image*, podemos coger los datos de la imagen píxel a píxel, de modo que el resultado sea un array de *int* (enteros) donde se encuentran estos datos.

Las dos líneas que hay a continuación de la creación del array de enteros se colocan porque son datos que se deben encontrar obligatoriamente en un archivo DICOM, pues sin ellos no es posible crear dicho archivo. De este modo, se asigna el identificador (UID) correspondiente a "*Secondary Capture Image Storage*" a la clase y a la instancia SOP respectivamente. Más tarde, en nuestra instancia de la clase *DicomImage* (dcmNuevo) se insertan estos datos de imagen mediante el método de esta clase *imagePixelData (int filas, int columnas, int planarConf, int[] pixelData)*.

Con esto ya hemos conseguido insertar los datos de imagen que necesitábamos.

8.3.2. Inserción de datos de texto recogidos en DICOM

Insertar datos de texto en el archivo DICOM que ya recoge el estándar es bien sencillo, puesto que existe una clase en *com.archimed.dicom.DDict* (Diccionario de Datos) para cada atributo del objeto DICOM, como pueden ser por ejemplo

DDict.dPatientName, *DDict.dPatientAddress* o *DDict.dRegionOfResidence*. Lo que hay que hacer es coger los datos del campo de texto en el panel correspondiente de nuestra interfaz mediante el método *getText()* e incluirlos en el objeto DicomImage mediante el método *set(int datoDeDDict, Object datoAsociado)*. Lo vemos en detalle:

```
String telefono = PanelDatosPersonales.tfnoCampo.getText();
dcmNuevo.set(com.archimed.dicom.DDict.dPatientTelephoneNumber,
      telefono);
```

Donde *telefono* es el dato del número de teléfono del paciente que se inserta en el campo del archivo DICOM *DDict.dPatientTelephoneNumber*.

Hecho esto, el dato estaría en el objeto DicomImage pero no todavía en el archivo .dcm, para lo que se implementan estas otras líneas de código una vez que ya hemos insertado todos los datos necesarios en nuestro objeto:

```
FileOutputStream save = new FileOutputStream(openFileName);
dcmNuevo.write(save,true);
```

Siendo *write* un método de la clase *DicomObject* que lo que hace es escribir a través de un *FileOutputStream* todos los datos de la instancia DicomImage en un archivo *.dcm* que, si no existe, crea uno.

8.3.3. Inserción de datos de texto que no están recogidos en DICOM

Cuando queremos añadir nuevos campos al objeto que no se recogen en el estándar, lo único que tenemos que hacer es crear nuevos pares atributo – VR (valor) y añadirlos al diccionario de datos. Las librerías JDT bis dan la posibilidad de crear campos donde se albergan datos de tipo *Integer* o datos de tipo *String*, es decir, campos para datos numéricos y campos para datos de texto.

El código crear campo de texto básicamente es:

```
// En primer lugar, creamos una instancia DDict (diccionario
//  de datos DICOM)
com.archimed.dicom.DDict diccionario = new
      com.archimed.dicom.DDict();
```

```
// A continuación, una entrada de datos de paciente, por eso
//  grupo = 0010, tipo cadena de caracteres
com.archimed.dicom.DDictEntry entrada1 = new
      com.archimed.dicom.
      DDictEntry(0010, 4001, com.archimed.dicom.
      DDict.tST, "Resumen enfermedad", "1");
diccionario.addEntry(entrada1); // la añadimos al diccionario
String resumen = PanelDatosClinicos.resumenEnfermedad.getText();
//   Y finalmente, la añadimos a nuestro objeto de imagen DICOM
// tener en cuenta q tenemos que convertir a String para que los
// tipos de datos sean compatibles
dcmNuevo.set_ge(0010, 4001, new String(resumen.toString()));
```

Lo primero que se hace es crear una instancia de la clase *DDictEntry*. Al pasar los parámetros fijamos el número de grupo, el número de elemento, la descripción del campo y el tipo de dato DICOM que se puede meter en este campo. Como estamos introduciendo datos referentes al paciente, todas las nuevas entradas que añadimos pertenecen al grupo 0010, que es el correspondiente en el diccionario de datos DICOM (parte 3 del estándar) [12]. Para el número de elemento se toma uno que no esté ya definido en dicho diccionario. Los nuevos campos que se crearon para nuestra aplicación los podemos encontrar en la tabla 8.1.

Una vez hecho esto, creamos un objeto de la clase *DDict*, en el cual vamos a insertar nuestro nuevo campo por medio del método *addEntry(DDictEntry a)*. En este momento tenemos la posibilidad de poder meter en el objeto de la clase DicomObject (dcmNuevo) el nuevo dato mediante el método *set_ge(grupo,elemento,dato)*, que en este caso debe ser una cadena de texto ya que el tipo ST DICOM es tipo String en Java. Ver tablas de conversión 7.1 y 7.2.

Para el caso de insertar un campo que albergue un dato de tipo Integer se debe proceder de la misma forma, lo único que cambia es el tipo de dato que vamos a insertar, por lo que el tercer parámetro que se pasa al constructor de la clase *DDictEntry* es *DDict.tUS*.

Tabla 8.1. Nuevos campos de datos para el objeto DICOM

DESCRIPCIÓN	Nº GRUPO	Nº ELTO.	VM
Resumen de la enfermedad	0001	4001	1
Resumen de la exploración clínica	0001	4002	1
Exploraciones complementarias	0001	4003	1
Diagnóstico provisional / definitivo	0001	4004	1
Tratamiento anterior / actual	0001	4005	1
Fundamento clínico de la petición	0001	4006	1
CNP[1] del médico solicitante	0001	4007	1
Especialidad del médico solicitante	0001	4008	1
Localidad del centro ambulatorio	0001	4009	1
Nº de afiliación a la SS del paciente	0001	4010	1
Padece diabetes	0001	4011	1
Padece hipertensión	0001	4012	1
Tratamiento con anticoagulantes	0001	4013	1
Intervenciones previas	0001	4014	1
Enfermedades anteriores	0001	4015	1
Alergias a medicamentos	0001	4016	1
Otros comentarios	0001	4017	1

8.4. Implementación de servicios DIMSE

Lo único que nos queda por ver en este capítulo es la forma en que se implementan las comunicaciones de nuestra aplicación cliente con el servidor. Obviamente, también aquí vamos a seguir las pautas marcadas por el estándar DICOM. Las librerías JDT vuelven a ser una herramienta fundamental a la hora de construir la aplicación.

Como ya vimos en el capítulo 4, según sea lo que queramos hacer (comprobar la conexión con el servidor, almacenar una imagen DICOM en una base de datos o buscarla en dicha base de datos…), es decir, el servicio DIMSE que queremos utilizar, tendremos que recurrir a una clase SOP o a otra.

[1] Código Numérico Personal

La forma de proceder ya la hemos analizado en capítulos anteriores, pero ahora vamos a ver la forma en que nosotros hemos llevado a cabo la implementación de los servicios.

En nuestro proyecto haremos uso de dos de los servicios DIMSE: el de verificación de conexión o C-ECHO y el de petición de almacenamiento del objeto DICOM en la base de datos de la central del HUVR o C-STORE. Por este motivo, tendremos que implementar ambos servicios, lo vamos a ver en los apartados 8.4.2 y 8.4.3.

8.4.1. Comunicación entre dos entidades DICOM pares

Cuando dos entidades DICOM pares se quieren comunicar entre sí, hay algunos pasos que ambas entidades deben realizar con independencia del servicio DIMSE que quieran llevar a cabo. Así, la entidad que va a ser la usuaria del servicio (*SCU*, *Service Class User*), tratará de establecer una asociación con la entidad proveedora de dicho servicio (*SCP*, *Service Class Provider*). Una vez que dicha asociación está establecida, ambas entidades intercambiarán la información correspondiente al servicio DIMSE que implementan y finalmente, hay que liberar la asociación.

En nuestro proyecto, la entidad que lleva a cabo el papel de SCU será aquella en la que se esté ejecutando la aplicación cliente (ordenadores de los centros ambulatorios), puesto que es la que solicita el servicio de verificación y almacenamiento al servidor que se encuentra en el HUVR que por tanto será quien realice el papel de SCP atendiendo las peticiones que se le hacen desde la aplicación cliente.

Para establecer una asociación con una entidad del par DICOM, se hace uso las clases del paquete *com.archimed.dicom.network*. La aplicación cliente de los centros ambulatorios es quien inicia la asociación tanto si usa el servicio de almacenamiento como el de verificación de la conexión. Pero veámoslo paso a paso:

1. En primer lugar, tenemos que hacer una conexión TCP/IP con la entidad del par DICOM mediante las clases estándares de *java.net*. Como ya se explicó en su momento, cuando se utiliza la torre de protocolo TCP/IP, DICOM agrupa los

Servicios de Capa Superior OSI en una capa denominada "Capa Superior DICOM" (*DUL*, *DICOM Upper Layer)*, que realiza las mismas funciones. Pero a la hora de establecer las asociaciones entre entidades, habrá que adaptarse a los mecanismos definidos en TCP/IP. Por lo tanto, para poder establecer la comunicación a través de la red, tendremos que hacer uso de una instancia *Socket* de la siguiente forma:

```
Socket s = new Socket (host,port);
```

Donde los parámetros que pasamos al objeto socket son una string (*host*) con el nombre del servidor con el que queremos establecer la asociación y el puerto por el que se realiza la conexión, que para el caso del servidor DICOM será el 104.

2. Crear un objeto *Association* con los InputStreams y OutputStreams derivados de Socket. De este modo, asociaremos un canal por el que leer las entradas de datos desde el servidor y otro por el que enviarlos a la conexión que hemos creado.

```
InputStream in = s.getInputStream();
OutputStream out = s.getOutputStream();
Association as = new Association(in,out);
/**
* Otra posible forma de hacerlo será:
*/
as = new Association(s.getInputStream(),s.getOutputStream());
```

3. Preparar un objeto *Request* con los parámetros necesarios para establecer una asociación. Los parámetros son al menos un título de la entidad de la aplicación llamada (called), un título de la entidad de la aplicación llamante (calling) y un array de enteros con la sintaxis de transferencia para que la entidad a la que se está llamando sea consciente de la misma. Los títulos de las entidades no deben ser más largos de 16 caracteres. En el ejemplo que sigue a continuación, conectamos con el servidor para verificar la conexión con sintaxis de transferencia *Implicit Little Endian*.

```
/**
*  Preparamos el objeto request con entidad llamante (calling) y
*  llamada (called)
*/
```

```
Request request = new Request();
request.setCalledTitle(called);
request.setCallingTitle(calling);
/**
*  Ya sólo nos queda añadirle a request el contexto de
*  presentación con la clase SOP y la sintaxis de transferencia
*/
int[] tsids = {TransferSyntax.ImplicitVRLittleEndian};
request.addPresentationContext(SOPClass.Verification,tsids);
```

4. Se envía la petición a la entidad del par DICOM y recibe la respuesta.

```
as.sendAssociateRequest(request);
Response response = as.receiveAssociateResponse();
```

5. Analiza la respuesta. Se comprueba si se acepta, se rechaza o se aborta nuestra petición.

```
. . .

// Analiza la respuesta, que será siempre del tipo
// Acknowledge, Reject o Abort
if (response instanceof Acknowledge){
    ack = (Acknowledge)response;
    for (i = 0 ; i < request.getPresentationContexts() ; i++){
      if (request.getAbstractSyntax(i).getConstant() !=
          SOPClass.Verification)
             continue;
      for (j = 0 ; i < ack.getPresentationContexts() ; j++){
          if (request.getID(i) == ack.getID(j)){
          if (ack.getResult(j) == Acknowledge.ACCEPTANCE)
            usablePCIndices.addElement(new Integer(i));
        }
      }
    }
    return true;
}
else if (response instanceof Reject){
    if (verbose){
      System.out.println("Asociación rechazada: " + response);
    }
    javax.swing.JOptionPane.showMessageDialog(null,"Petición de
      asociación rechazada: " + response);
    return false;
  }
  else if (response instanceof Abort){
    if (verbose){
      System.out.println("Asociación abortada: " + response);
    }
    javax.swing.JOptionPane.showMessageDialog(null,"Asociación
      abortada: " + response);
    return false;
  }
  . . .
```

6. En este punto se tiene una asociación válida para el intercambio de información de verificación de conexión entre ambas entidades y se puede empezar a realizar dicho intercambio.

7. Ya sólo nos quedaría liberar la asociación de la siguiente manera:

```
as.sendReleaseRequest();
as.receiveReleaseResponse();
/**
*  En este punto, la aplicación se mantiene en espera hasta que
*  recibe el asentimiento para la liberación de la asociación
*/
if (verbose) {
    System.out.println("Recibido el asentimiento de
      liberación");
}
```

Ya hemos terminado de estudiar la forma en que se establece y se libera una conexión o asociación entre dos entidades DICOM cuando la SCU quiere solicitar la realización de un servicio DIMSE a la SCP. Ya sólo nos resta ver la implementación de dichos servicios, que analizaremos en los dos apartados siguientes. Para ello, supondremos en ambos casos que la asociación entre las entidades está establecida de modo correcto.

8.4.2. Implementación del servicio DIMSE C-ECHO

Los pasos que hay que seguir para la consecución de este servicio son los que presentamos a continuación:

1. La entidad que solicita el servicio (SCU) crea una petición C-ECHO mediante el método *createEchoRequest (int messageid,int sopclassuid)* (que pertenece a la clase *DimseUtil* del paquete *com.archimed.dicom.network*), la envía y recibe la respuesta:

```
DicomObject echoreq = DimseUtil.createEchoRequest
    (new Integer(0), new Integer(SOPClass.Verification));
// Añadimos los contextos de presentación
as.sendInPresentationContext(pcid,echoreq,null);
DicomObject echores = as.receiveCommand();
```

2. Hay que analizar a continuación si echores es una respuesta C-ECHO correcta o no. Podría no ser correcta por varios motivos, los analizaremos en primer lugar y si no es incorrecta, se deduce que será correcta.

```
/**
*  Primero, vemos si la respuesta es distinta del tipo C-ECHO
*/
if (DimseUtil.getCommandType(echores) !=
   DimseUtil.C_ECHO_RESPONSE){
    if (verbose) {
      System.out.println("No es una respuesta de tipo C-ECHO
         RESPONSE : " + DimseUtil.getCommandType(echores));
    }
    javax.swing.JOptionPane.showMessageDialog(null,
      "No es una respuesta de tipo C-ECHO RESPONSE:" +
      DimseUtil.getCommandType(echores));
    /**
    *  Como no es válida, abortamos la asociación
    */
    as.sendAbort(Abort.DICOM_UL_SERVICE_USER,
      Abort.REASON_NOT_SPECIFIED);
    return false;
}

/**
*  Ahora comprobaremos si el UID de la clase SOP contenida en la
*  respuesta C-ECHO es el correcto
*/
String sopclass = echores.getS(DDict.dAffectedSOPClassUID);
if (DimseUtil.getAffectedSOPClass(echores) !=
    SOPClass.Verification){
    if (verbose) {
      System.out.println("UID de la clase SOP contenida en la
        C-ECHO RESPONSE incorrecto: " +
        DimseUtil.getAffectedSOPClass(echores));
    }
    javax.swing.JOptionPane.showMessageDialog(null," UID de la
       clase SOP contenida en la C-ECHO RESPONSE incorrecto: " +
       DimseUtil.getAffectedSOPClass(echores));
   /**
   *  Abortamos la asociación por el mismo motivo que antes
   */
    as.sendAbort(Abort.DICOM_UL_SERVICE_USER,
      Abort.REASON_NOT_SPECIFIED);
    return false;
}

/**
*  Ahora comprobaremos si el ID del mensaje C-ECHO response es
*  correcto o no
*/
if (DimseUtil.getMessageIDBeingRespondedTo(echores) != 0){
  if (verbose) {
    System.out.println("ID del Mensaje contenido en C-ECHO
      RESPONSE incorrecto: " +
      DimseUtil.getMessageID(echores));
  }
  javax.swing.JOptionPane.showMessageDialog(null," ID del
      Mensaje contenido en C-ECHO RESPONSE incorrecto: " +
      DimseUtil.getMessageID(echores));
```

```
    /**
    *  Abortamos la asociación
    */
    as.sendAbort(Abort.DICOM_UL_SERVICE_USER,
      Abort.REASON_NOT_SPECIFIED);
    return false;
  }

  /**
  *  Finalmente, ya sólo nos queda comprobar el estado de la
  *  C-ECHO RESPONSE
  */
  int status = echores.getI(DDict.dStatus);
  if (status != 0){
    if (verbose) {
       System.out.println("No hay estado de éxito en la respuesta
         C-ECHO: " + status);
    }
    javax.swing.JOptionPane.showMessageDialog(null," No hay estado
       de éxito en la respuesta C-ECHO: " + status);
    /**
    *  Abortamos la asociación
    */
    as.sendAbort(Abort.DICOM_UL_SERVICE_USER,
      Abort.REASON_NOT_SPECIFIED);
    return false;
  }
  /**
  *  Si no hubo fallo, la conexión entre las entidades es correcta
  */
  return true;

  . . .
```

3. Si la respuesta C-ECHO es correcta, significa que la conexión se encuentra en perfecto estado, y si no lo es, significa que algo falla y obtendremos el motivo. Esto es lo que se persigue con la utilización del servicio DIMSE C-ECHO, por lo que una vez que hayamos descubierto el estado de la conexión, podremos liberar tranquilamente la asociación de la forma que vimos en el apartado anterior.

8.4.3. Implementación del servicio DIMSE C-STORE

Al igual que en el apartado anterior, partimos de la suposición de que la asociación entre entidades DICOM ya se encuentra establecida (teniendo en cuenta que se ha hecho para una clase SOP del tipo *Secondary Capture Image Storage*), y también que se ha creado correctamente el archivo DICOM que queremos enviar al servidor para su almacenamiento. Para tal fin se ha creado un método nuevo que se llama *exchangeInfo(DicomObject dcmNuevo)* dentro de la clase *EnviaFich* que como

podemos ver, recibe como parámetro un objeto DICOM que será el que mandemos al servidor a través de la red. Este método hace lo siguiente:

1. Tras inicializar los contextos de presentación pertinentes, se crea una petición de almacenamiento C-STORE mediante el método *createStoreRequest(int messageid, int sopclassuid, int priority, String sopinstanceuid, String moveae, String moveid)* (que está dentro de la clase *DimseUtil* del paquete *com.archimed. dicom.network),* la enviamos y esperamos a recibir la respuesta:

```
DicomObject storereq = DimseUtil.createStoreRequest(new
   Integer(0),new Integer(SOPClass.SecondaryCaptureImageStorage)
   ,new Integer("0002H"),"1.2.840.10008.5.1.4.1.1.7",calling,
   called);
/**
*  Enviamos el objeto Dicom que generamos antes en el contexto
*  de presentación seleccionado
*/
as.sendInPresentationContext(pcid,storereq,dcm);
DicomObject storeresp = as.receiveCommand();
```

2. De igual modo que en el apartado anterior, tenemos que comprobar si el almacenamiento del *dataset* DICOM en la base de datos del servidor se realizó con éxito o no mediante la respuesta C-STORE que obtengamos. Primero comprobaremos si la respuesta C-STORE es incorrecta, y después si el mensaje que incluye es de éxito en la operación de almacenamiento o fracaso. Se hizo de la siguiente forma:

```
boolean enviaFich (DicomObject dcm){

. . .


/**
*  En primer lugar, comprobamos si la respuesta obtenida es del
*  tipo C-STORE RESPONSE o no lo es
*/
if (DimseUtil.getCommandType(storeresp) !=
   DimseUtil.C_STORE_RESPONSE){
   if (verbose){
     System.out.println("No es una respuesta del tipo C-STORE
        RESPONSE: " + DimseUtil.getCommandType(storeresp));
   }
   // Abortamos la asociación
   as.sendAbort(Abort.DICOM_UL_SERVICE_USER,
     Abort.REASON_NOT_SPECIFIED);
   return false;
}
```

```
/**
*  Ahora comprobamos si el UID de la clase SOP contenida en la
*  respuesta C-STORE es el correcto, es decir, el que
*  corresponde a Secondary Capture Image Storage
*/

String sopClass = storeresp.getS(DDict.dAffectedSOPClassUID);
if (DimseUtil.getAffectedSOPClass(storeresp) !=
   SOPClass.SecondaryCaptureImageStorage){
   if (verbose){
     System.out.println("UID UID de la clase SOP contenida en la
        C-STORE RESPONSE incorrecto: " +
        DimseUtil.getAffectedSOPClass(storeresp));
   }
   // Abortamos la asociación
   as.sendAbort(Abort.DICOM_UL_SERVICE_USER,
     Abort.REASON_NOT_SPECIFIED);
   return false;
}

/**
*  Hay que comprobar si el ID del mensaje contenido en la
*  respuesta C-STORE es correcto o no
*/
if (DimseUtil.getMessageIDBeingRespondedTo(storeresp) != 0){
   if (verbose){
   System.out.println("ID del Mensaje contenido en C-STORE
      RESPONSE incorrecto: " +
      DimseUtil.getMessageID(storeresp));
   }
   // Abortamos la asociación
   as.sendAbort(Abort.DICOM_UL_SERVICE_USER,
     Abort.REASON_NOT_SPECIFIED);
   return false;
}

/**
*  Ya sólo nos queda comprobar el estado de la respuesta. Si
*  este campo vale 0000, es que el almacenamiento se realizó con
*  éxito, si no, será que hubo algún fallo
*/
int status = storeresp.getI(DDict.dStatus);
if (status != 0){
  if (verbose){
     System.out.println("El estado en la respuesta C-STORE no es
        de éxito: " + status);
  }
  // Abortamos la asociación
  as.sendAbort(Abort.DICOM_UL_SERVICE_USER,
    Abort.REASON_NOT_SPECIFIED);
  return false;
}
/**
*  Si no se produjo ninguna de las condiciones anteriores, el
*  envío del archivo DICOM y su correspondiente almacenamiento
*  se realizaron con éxito
*/
javax.swing.JOptionPane.showMessageDialog(null,"El formulario ha
      sido enviado y almacenado correctamente");
return true;
```

```
    . . .
}
```

3. Una vez que sabemos si el almacenamiento del formulario se realizó con éxito o por el contrario, no se pudo realizar, sólo nos quedaría liberar la asociación entre las entidades de la misma forma que vimos en el apartado 8.4.1.