Capítulo 7
Estudio de las librerías JDT

7.1. Introducción a las librerías JDT

Java DICOM Toolkit (conjunto de librerías JDT) es la ayuda para un programador en JAVA para construir una aplicación que siga lo marcado por el estándar DICOM 3.0. Combina las ventajas y la fuerza de DICOM y JAVA en una API muy fácil de usar. Proporciona numerosas clases y métodos que simplifican la programación de aplicaciones DICOM. Será una herramienta fundamental en el desarrollo de la aplicación cliente de los centros ambulatorios, ya que permitirá convertir las imágenes que se envían junto con los datos de los pacientes a un formato compatible con el estándar de comunicaciones médicas por excelencia.

JDT[1] es el primer equipo de desarrollo de software DICOM que está implementado totalmente en JAVA. Con esto, los desarrolladores DICOM pueden beneficiarse de las numerosas ventajas del lenguaje de programación JAVA y desplegarlo en DICOM applets, aplicaciones independientes JAVA y en software de

[1] Java Dicom Toolkit

servidor. JDT viene con una API bien documentada que ha sido diseñada para hacer que la estructura compleja del estándar DICOM sea más accesible para los desarrolladores. JDT funciona sobre JDK 1.1.X y JAVA 2.

7.1.1. Terminología

En esta sección se usan palabras como conjunto de datos (*Dataset*) y atributos. Conjunto de datos se usa para identificar una colección de atributos DICOM. Un atributo se identifica por su par (número de grupo, número de elemento), y tiene un cierto tipo DICOM (value representation - VR).

7.1.2. Datasets o Conjuntos de Datos

En JDT, todos los conjuntos de datos están representados por *com.archimed. dicom.DicomObject*. Esta clase es una subclase de *com.archimed.dicom.GroupList*. Básicamente es donde residen los atributos DICOM *com.archimed.dicom.TagValue*.

Manipulación de los atributos

Un conjunto de datos DICOM es representado por un objeto de la clase *DicomObject*. Los diferentes atributos de estos conjuntos de datos están internamente almacenados en el DicomObject como objetos *com.archimed.dicom.Tagvalue*. El objeto TagValue contiene un par (grupo, elemento) y los valores del atributo están almacenados en un Vector. El tamaño del Vector corresponde a la multiplicidad del atributo.

El tipo JAVA que representa el valor de un atributo depende del tipo DICOM (*Value Representation* - VR) definido para ese atributo específico. Las tablas 7.1 y 7.2 muestran la equivalencia entre los tipos DICOM y sus correspondientes tipos JAVA.

Tabla 7.1. Tabla de Conversión entre tipo Java y tipo DICOM

Tipo JAVA	Tipo DICOM
Short	SS
Integer	US, US-SS, IS, AT, SS, SL, UL
Long	SL, UL
Float	FL, DS
Double	FD
byte[]	OB, OW, OW-OB
int[]	AT (int[2])
String	AS, AE, CS, LO, LT, SH, ST, UI, TM, DM, PA, DS, IS, SS, US, SL, UL, FD, OB, OW, OW-OB
Person	PN
DDate	DA
DDateRange	DA

Manipulación de atributos simples

Hay un número de métodos *set/get* en DicomObject con los que se pueden meter, sacar o modificar los atributos guardados en un DicomObject. La forma de hacer esto es con la clase *com.archimed.dicom.DDict*. La clase DDict tiene una constante definida para cada atributo definido en el estándar DICOM. Por ejemplo:

```
DDict.dPatientName
DDict.dAccessionNumber
```

Esas constantes pueden ser usadas en los métodos set/get del DicomObject:

```
Person p = (Person)dcm.get(DDict.dPatientName);
Integer acnumber = (Integer)dcm.get(DDict.dAccessionNumber);
dcm.set(DDict.dPatientName, new Person("Fernando"));
dcm.set(DDict.dAccessionNumber, new Integer(12345));
```

El método set implícitamente convierte el valor del argumento dado al tipo DICOM correcto. También se puede acceder a los atributos de un DicomObject a través del par (grupo,elemento).

```
Person p = (Person)dcm.get ge(0x0010, 0x0010);
```

```
Integer acnumber = (Integer)dcm.get ge(0x0008,0x0050,
      DDict.dAccessionNumber);
dcm.set ge(0x0010, 0x0010, new Person("Fernando"));
dcm.set ge(0x0008, 0x0050, new Integer(12345));
```

Hay también dos métodos get que convierten directamente a String o int:

```
String s = dcm.getS(DDict.dPatientName);
int i = dcm.getI(DDict.dAccessionNumber);
```

Tabla 7.2. Tabla de Conversión de tipo DICOM a tipo Java

Tipo DICOM	Tipo JAVA
AE	String
AT	String
AT	Integer (0xggggeeee)
CS	String
DA	DDate o DDateRange
DS	Float
FL	Float
FD	Double
IS	Integer
LO	String
LT	String
OB	byte[]
OW	byte[]
OB-OW	byte[]
PN	Person
SH	String
SL	Long
SS	Short
ST	String
TM	String
UI	String
UL	Long
US	Integer
US-SS	Integer

Para manipular atributos con una multiplicidad mayor de uno, se usan los métodos get/set con un argumento adicional. La multiplicidad de un atributo se obtiene con *getSize()*. Por ejemplo, suponiendo que un DicomObject dcm contiene un atributo ImageType con multiplicidad 2 y valores DERIVED/SECONDARY, entonces se puede coger esos valores de la siguiente forma:

```
int multiplicity = dcm.getSize(DDict.dImageType);
//devuelve el valor 2
String str1 = (String)dcm.get(DDict.dImageType,0);
//devuelve el valor DERIVED
String str2 = (String)dcm.get(DDict.dImageType,1);
//devuelve el valor SECONDARY
```

Insertando el valor de un atributo ocurre lo mismo:

```
dcm.set(DDict.dImageType,0,"DERIVED");
dcm.set(DDict.dImageType,1,"SECONDARY");
```

Secuencias

Las secuencias se tratan de una forma parecida a como se hace con los valores de multiplicidad mayor de uno. Cuando un atributo en un DicomObject representa una secuencia (tipo SQ DICOM), entonces los "items" de la secuencia pueden ser insertados y cogidos con los mismos métodos get/set del DicomObject que ahora aceptan y devuelven los DicomObejcts completos. Por ejemplo, suponiendo que el DicomObject dcm contiene un atributo DirectoryRecordSequence con un cierto número de items:

```
int numberofitems = dcm.getSize(DDict.dDirectoryRecordSequence);
DicomObject item0 = (DicomObject) dcm.get
      (DDict.dDirectoryRecordSequence,0);
DicomObject item1 = (DicomObject) dcm.get
      (DDict.dDirectoryRecordSequence,1);
```

Para añadir un valor a una secuencia, simplemente se usa el método set que tiene el DicomObject que representa el item y el índice del item.

Son accesibles con las mismas funciones get/set de las secuencias como las usadas en los datasets de los DicomObject. Esto provee a las secuencias de la misma funcionalidad que los DicomObject.

Para un rápido acceso, hay una utilidad de la clase c*om.archimed.dicom.tools. Sequences* que contiene varios métodos para acceder a las secuencias. También cuida de todas las diferentes excepciones encontradas. Los nombres elegidos para los métodos son muy similares que los encontrados en el DicomObject.

Comentarios:

- ♣ Las longitudes de los grupos no se almacenan en un DicomObject, se calculan cuando se necesitan, así que no hay acceso a ellos.
- ♣ Cuando usamos set para insertar valores de atributos, los valores se almacenan por referencia y no se copian.
- ♣ Cuando usamos get para sacar valores de atributos, se devuelve una referencia al valor.

Manipulación de grupos de atributos

Para usuarios que quieran manipular los grupos enteros del conjunto de datos, está el acceso al grupo proporcionado por *com.archimed.dicom.GroupList*, el cual es la superclase de *com.archimed.DicomObject*. Un grupo es un conjunto de atributos que tienen el mismo número de grupo en su par (grupo,elemento).

```
DicomObject copyGroup(int groupnr);
```

Copia el grupo con el número *groupnr* y lo devuelve como un DicomObject nuevo. Notar que se hace una copia no profunda, pues sólo se copian referencias a los atributos.

```
void addGroups(DicomObject o);
```

Agrega a todos los grupos encontrados en "o" a este dataset. Solamente se agregan los grupos que no estaban ya presentes en este dataset. Una vez más lo único que se hace es una copia baja o no profunda.

```
DicomObject removeGroup(int groupnr);
```

Suprime un grupo entero con el número "groupnr" de este dataset. Y lo devuelve como un DicomObject.

7.1.3. Entrada/Salida de los datasets DICOM

Hay varias formas de leer o escribir datasets DICOM a y desde InputStreams / OutputStreams (canales de entrada / salida). En ambos casos hay un método básico y algunos métodos más que permiten especificar todas las clases de parámetros.

Leer datasets

```
void read(java.io.InputStream in);
```

Este método lee todos los atributos DICOM desde un inputStream de un DicomObject. El inputStream puede contener tanto un archivo DICOM como un conjunto de datos (datasets) DICOM. Cuando el inputStream contiene un archivo de la parte 10 DICOM, se lee y se almacena la Metainformación del fichero (*Meta File Information*) en un DicomObject separado que se puede coger mediante *getFileMetaInformation()*. Se asume que la sintaxis de transferencia es *Implicit VR Little Endian* cuando el inputStream contiene un conjunto de datos DICOM. Cuando el inputStream contiene un archivo de la parte 10 de DICOM, la sintaxis de transferencia se detecta desde la Metainformación del fichero.

```
void read(java.io.InputStream in, boolean process);
```

Este método hace lo mismo que el anterior pero además permite especificar si analizar o saltar datos de píxel. Esto ahorra el tiempo de transformación en el que no se necesitan los datos de píxel.

```
void read(java.io.InputStream in, int ts, boolean process);
```

En el parámetro "ts" se puede especificar qué sintaxis de transferencia se usa. Este parámetro puede venir dado cuando se conoce por adelantado con qué sintaxis de transferencia se codificó el dataset.

Escribir datasets

```
void write(java.io.OutputStream out, boolean f);
```

Éste es el método más básico de la exportación. Si el argumento f se fija como "true", el dataset se codifica en un archivo de la parte 10 de Dicom (archivo Dicom). En este caso, la Metainformación de fichero se usa si está presente, o se crea cuando es necesitada. Hay que señalar que la creación de la metainformación puede lanzar una excepción, cuando los atributos requeridos faltan (*SOP Class UID, SOP Instance UID*).

```
void write(java.io.OutputStream out, boolean f, int ts,
       boolean seq_undef);
```

Los parámetros adicionales "ts" y "seq_undef" se pueden utilizar para escribir conjuntos de datos casi en todos los posibles tipos de DICOM. Los parámetros ts y seqUndef pueden ser usados para insertar la sintaxis de transferencia. Toda posible combinación de parámetros ts y f se permiten, pero no todas las combinaciones siguen los dictados del estándar de DICOM. Si se escribe un conjunto de datos con f "false" y ts otra cosa que no sea Implicit VR Little Endian, será imposible para otras aplicaciones decodificarlo, ya que no pueden detectar la sintaxis de transferencia utilizada. El parámetro seqUndef indica al escribir secuencias si se usa longitud indefinida (true) o definida (false).

Visualización de datasets

DicomObject contiene dos métodos para mostrar todos los atributos contenidos en un OutputStream de forma ordenada para que el ser humano sea capaz de leerlo.

```
void dumpVRs(java.io.OutputStream os);
void dumpVRs(java.io.OutputStream os, boolean metainfo);
```

Habrá que poner "metainfo" como "true" si se quiere ver la Metainformación del fichero.

Ejemplos

Estos trozos de código demuestran el fácil uso de JDT. Con sólo unas líneas de código, es posible construir soluciones DICOM poderosas. El código siguiente lee un archivo DICOM desde un archivo y muestra los atributos (incluidos los contenidos en la metainformación si existe) por la pantalla.

```
DicomObject dcm = new DicomObject;
dcm.read(new FileInputStream("foo.dcm"));
dcm.dumpVRs(System.out, true);
```

El siguiente ejemplo lee los datos de un archivo y lo escribe en otro archivo usando una sintaxis de transferencia y una secuencia de codificación específicas.

```
DicomObject dcm = new DicomObject;
dcm.read(new FileInputStream("in.dcm"));
dcm.write(new FileOutputStream("out.dcm"), true,
       TransferSyntax.ExplicitVRBigEndian, true);
```

7.1.4. Métodos útiles

Enumeración de los atributos de un dataset

Con el método *enumerateVRs* es posible conseguir una enumeración (clase de java) de todos los atributos contenidos en un DicomObject.

```
Enumeration e = dcm.enumerateVRs(false);
```

enumerateVRs() devuelve una enumeración de todos los atributos de dcm como objetos TagValue. TagValue tiene métodos para coger el valor y el par (grupo,elemento) de un atributo.

Metainformación de fichero

```
DicomObject fmi = dcm.getFileMetaInformation();
```

Devuelve la Metainformación del DicomObject dcm si existe. Se puede usar ahora fmi para añadir o alterar atributos específicos de esta información como se quiera.

Información general

Unos pocos métodos de *com.archimed.dicom.GroupList* proporcionan información general sobre un conjunto de datos:

- *int numberOfElements()*: devuelve el número total de atributos.
- *int numberOfGroups()*: devuelve el número total de grupos.
- *boolean isEmpty()*: devuelve si el GroupList contiene algún atributo.
- *boolean containsGroup(int g)*: devuelve si el GroupList contiene el grupo especificado.

7.2. Depósitos

Hay unas clases en el paquete *com.archimed.dicom* que hacen más fácil a los programadores usar los elementos de datos y los registros UIDs.

7.2.1. Clase DDict: depósito de atributos

La clase *com.archimed.dicom.DDict* es un almacén de los VRs (representaciones de valores) y los elementos de datos. Contiene un gran número de constantes que representan los diferentes VRs y un gran conjunto de constantes para los elementos de datos:

- *DDict.tPN*: representa el tipo DICOM PERSON.
- *DDict.tUS*: representa el tipo DICOM UNSIGNED SHORT.
- *DDict.dAccesionNumber*: representa el atributo AccessionNumber.
- *DDict.dPatientName*: representa el atributo PatientName.

Los métodos de DDict

La clase DDict tiene métodos para obtener el par (grupo,elemento) de un atributo, un tipo de atributo, una descripción del atributo y para consultar una constante DDict del atributo basada en su par (grupo,elemento).

```
static int getGroup(int dct);
```

Devuelve el número de grupo para una constante DDict que representa un atributo.

```
static int getElement(int dct);
```

Devuelve el número de elemento para una constante DDict que representa un atributo.

```
static int getTypeCode(int dct);
```

Devuelve el tipo para una constante DDict que representa un atributo. Los elementos de datos que no están en la lista de arriba tendrán un *DDict.tUNKNOWN*.

```
static java.lang.String getDescription(int dct);
```

Da una descripción elaborada para una constante DDict que representa un atributo.

```
static int lookupDDict(int g, int e);
```

Devuelve la constante DDict que representa el atributo con número de grupo "g" y número de elemento "e". Si no se encuentran co nstantes, el método devolverá *DDict.dUNDEFINED*.

7.2.2. Clase UID: depósitos UID

La clase *com.archimed.dicom.UID* contiene un depósito de los UIDs DICOM registrados (Parte 6 DICOM).Las constantes que representan los UIDs y que tienen que ser usadas a través de JDT se definen en subclases de UID:

- *com.archimed.dicom.TransferSyntax*: constantes que representan las sintaxis de transferencia.
- *com.archimed.dicom.SOPClass*: constantes que representan las clases SOP.
- *com.archimed.dicom.MetaSOPClass*: constantes que representan las clases Meta SOP.

- *com.archimed.dicom.SOPInstance*: constantes que representan las instancias SOP.

El objeto *com.archimed.dicom.UIDEntry* representa un único identificador y se obtiene con la clase UID de dos formas. Usando una constante de una de las subclases UID:

```
UIDEntry entry = UID.getUIDEntry(TransferSyntax.JPEGBaseline);
```

O usando una cadena UID:

```
UIDEntry entry = UID.getUIDEntry("1.2.840.10008.1.2.4.50");
```

Cada uno de esos métodos de consulta devuelve un objeto UIDEntry, el cual es un contenedor de varias propiedades del identificador único específico. Esos atributos pueden ser obtenidos usando los métodos get.

7.3. Imágenes en JDT

Esta parte habla de las capacidades de imagen de JDT. Todas las clases relacionadas con las imágenes pueden ser encontradas en el paquete *com.archimed.dicom.image*.

7.3.1. La clase DicomImage

La clase base para las imágenes es *com.archimed.dicom.image.DicomImage*. Aparte de los métodos set / get básicos para manipular los datos, inherentes al DicomObject, contiene otras formas de coger e insertar los atributos requeridos (tipo 1 y tipo 2) que son comunes a todos los DICOM Image Modality IODs (ver parte 3 de DICOM). Cuando se escribe una DicomImage en un outputstream, no hay un reconocimiento de si todos los atributos requeridos están presentes.

7.4. Guía para usuarios de JDT

7.4.1. Insertar datos que no son de imagen

DicomImage contiene algunos métodos para insertar los atributos requeridos que no están relacionados con los datos de imagen. Ofrecemos a continuación una breve descripción de algunos de estos métodos:

- *void patientData(String patientName, String patientID, String birthDate, String sex)*: inserta los datos del paciente.
- *void generalStudyData(String instanceUID, String date, String time, String phys-Name, String studyID, String orderNumber)*: inserta los atributos del estudio general.
- *generalSeriesData(String modality, String instanceUID, String seriesNumber)*: añade los datos generales de serie.
- *generalEquipmentData(String manufacturer)*: añade datos sobre el equipo utilizado.
- *generalImageData(String imageNumber)*: añade datos de imagen.
- *sopCommonData(String sopClassUID, String sopInstanceUID)*: inserta los datos comunes SOP. SOP Class UID es el identificador de una de las diferentes clases SOP de almacenamiento, que pueden ser encontradas en *com.archimed. dicom.SOPClass*.

7.4.2. Insertar datos de imagen con los métodos de DicomImage

Hay también métodos en *com.archimed.dicom.image.DicomImage* que pueden ser usados para insertar los datos de los píxeles de la imagen. La clase contiene métodos para imágenes en escala de grises, en paleta de colores y en RGB. Una definición de todos los parámetros pueden ser encontrados en la API.

```
DicomImage dcmi = new DicomImage();

byte[] pixels = new byte[640*480]; // rellena este array con
                                   // píxeles.
byte[] red = new byte[256]; // rellena este array con un LUT
                            //rojo.
```

```
byte[] green = new byte[256]; // rellena este array con un LUT
                              //verde.
byte[] blue = new byte[256]; // rellena este array con un LUT
                              //azul.
dcmi.imagePixelData(480, 640, pixels, red, green, blue);
```

Esto da una DicomImage con una interpretación fotométrica PALETTE COLOR, tamaño 640x480 y con una profundidad de píxel de 8.

7.4.3. Insertar los datos de imagen con ImageProducer

Hay otra forma de insertar los datos de la imagen. Dada una "*ImageProducer ip*", construyendo un objeto ImageIO y usando el método *setImageProducer()* para copiar los datos de imagen desde la ImageProducer a la DicomImage:

```
DicomImage dcmi = new DicomImage();
ImageIO imgio = new ImageIO(dcmi);
imgio.setImageProducer(ip);
```

Cuando se llama al método setImageProducer(), todos los datos se copian dentro de la DicomImage. Obteniendo una ImageProducer desde una DicomImage es también posible. Todo marco simple de una imagen DICOM se puede coger como una ImageProducer usando uno de los m´etodos siguientes:

```
java.awt.image.ImageProducer getImageProducer();
java.awt.image.ImageProducer getImageProducer(int i);
java.util.Vector getImageProducers();
```

Para imágenes de escalas de grises con profundidad de píxel mayor que 8 bits, hay un color model (clase de java) especial *com.archimed.dicom.image. GrayColorMode*. Este modelo de color se usa en combinación con un array int de píxeles, para producir imágenes de 256 grises sin pérdida de datos. Con este GrayColorModel, también es posible meter el Window/Level usado en la producción de la imagen.

- ♣ Sintaxis de transferencia sin compresión, una imagen: un array de bytes que mantiene el formato original de los datos de píxel definido en el estándar DICOM, ejemplos:

- Imagen MONOCHROME2, 8 bits asignados, 8 bits almacenados: 1 elemento del array de bytes por píxel.
- Imagen MONOCHROME2, 16 bits asignados, 12 bits almacenados: 2 elementos consecutivos del array de bytes por píxel.
- Imagen MONOCHROME2, 12 bits asignados, 12 bits almacenados: 1 elemento y medio del array de bytes por píxel.
- Imagen RGB, configuración plana color por píxel: 3 bytes consecutivos representando 1 píxel.
- Imagen RGB, configuración plana color por plano: el array de bytes está dividido en 3 partes de igual longitud cada una representando un color de plano (Red-Green-Blue).

Los arrays de bytes se cogen e insertan con los métodos get(DDict.dPixelData) y set(DDict.dPixelData, byte array) respectivamente.

- ♣ Sintaxis de transferencia sin compresión, varias imágenes: un array de bytes que contienen marcos consecutivos.
- ♣ Sintaxis de transferencia con compresión, una imagen: un array de bytes en el formato original de compresión.

7.5. Creación de una asociación

Para establecer una asociación con una entidad del par DICOM, se hace uso de las clases del paquete *com.archimed.dicom.network*. Es necesario un iniciador de la asociación y un receptor de la asociación.

7.5.1. Iniciador de la asociación

Pasos a seguir para crear un iniciador de una asociación DICOM. El código correspondiente lo mostraremos en el capítulo de implementación de la aplicación cliente para no ser repetitivos.

1. Hacer una conexión TCP/IP con la entidad del par DICOM que hace uso de las clases estándares de java.net. Hay que indicar el servidor y el puerto por el que éste escucha, que para DICOM es el 104.
2. Crear un objeto "Association" con los Input y Output Streams derivados de Socket.
3. Preparar un objeto "Request" con los par ámetros 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 abstract syntax con una sintaxis de transferencia. Estos títulos no deben ser más largos de 16 caracteres.
4. Se envía la petición a la entidad del par DICOM y recibe la respuesta.
5. Analiza la respuesta. Se comprueba si se acepta, se rechaza o se aborta nuestra petición.
6. En este punto se tiene una asociación válida para el tipo de imagen que hayamos especificado y podemos enviar ese tipo de imágenes a la entidad del par DICOM.
7. Acabar asociación una vez que se hayan enviado las imágenes.

7.5.2. Receptor de la asociación

Cuando una aplicación actúa como receptor de la asociación, espera conexiones entrantes de TCP/IP con el método *accept()* de java*.net.ServerSocket* y dedica un hilo para cada conexión entrante. En este ejemplo se empieza con el punto donde una entidad del par DICOM ha hecho una conexión de TCP/IP, dando por resultado un objeto válido de *java.net.Socket* que es devuelto por el método accept().

Los pasos a seguir para crear un receptor de una asociación DICOM son los siguientes:

1. Obtener el InputStream/OutputStream del objeto de java.net.Socket devuelto por accept() y crear un nuevo objeto Association.
2. Se lee la petición de asociación y se crea la respuesta apropiada para la petición. No se comprueba el título de la entidad de aplicación del llamante (*calling title*).

3. Si la respuesta es un *Reject* (Rechazo) se cierra la conexión. Si la respuesta es *Acknowledge* (Reconocida) se entra en un bucle que reciba y procese la verificación relacionada con el comando DIMSE correspondiente. El lanzamiento de la asociación también se maneja en el bucle.

7.6. Estructura de JDT

7.6.1. Árbol de clases

- class java.lang.Object
- class com.archimed.dicom.network.Association
- class java.awt.image.ColorModel (implements java.awt.Transparency)
    - class com.archimed.dicom.image.GrayColorModel
- class com.archimed.dicom.codec.Compression
- class com.archimed.dicom.DDate
- class com.archimed.dicom.DDateRange
- class com.archimed.dicom.DDict
- class com.archimed.dicom.DDictEntry
- class com.archimed.dicom.Debug
- class com.archimed.dicom.network.DimseUtil
- class com.archimed.dicom.network.ExtendedNegotiation
- class com.archimed.dicom.GroupList
    - class com.archimed.dicom.DicomObject
        - class com.archimed.dicom.image.DicomImage
            - class com.archimed.dicom.image.SCImage
- class com.archimed.dicom.image.ImageIO
- class com.archimed.dicom.Jdt
- class com.archimed.dicom.Offsets
- class com.archimed.dicom.Person
- class com.archimed.dicom.network.Request
- class com.archimed.dicom.network.Response
    - class com.archimed.dicom.network.Abort
    - class com.archimed.dicom.network.Acknowledge
    - class com.archimed.dicom.network.Reject

- ♣ class com.archimed.dicom.network.ResponsePolicy
- ♣ class com.archimed.dicom.tools.Sequences
- ♣ class com.archimed.dicom.TagValue
- ♣ class java.lang.Throwable (implements java.io.Serializable)
    - class java.lang.Exception
        - class com.archimed.dicom.DicomException
        - class com.archimed.dicom.IllegalValueException
        - class com.archimed.dicom.UnknownUIDException
- ♣ class com.archimed.dicom.UID
    - class com.archimed.dicom.MetaSOPClass
    - class com.archimed.dicom.SOPClass
    - class com.archimed.dicom.SOPInstance
    - class com.archimed.dicom.TransferSyntax
- ♣ class com.archimed.dicom.UIDEntry
- ♣ class com.archimed.dicom.image.WL

7.6.2. Packages

Com.archimed.dicom.network

- ♣ *Abort* (Abortar): Representa el aborto de una asociación. Los dos parámetros son, la fuente y la razón del aborto.
- ♣ *Acknowledge* (Reconocer): Estos objetos tienen los parámetros de la asociación Acknowledge, contiene todos los contextos de presentación del Acknowledge y son numerados desde 0 hasta el número de contextos de presentación menos uno. Puede actuar como receptor de asociación o como iniciador de la asociación.
- ♣ *Association* (Asociación): Contiene métodos para construir y destruir una asociación entre dos entidades de aplicación DICOM y mandar/recibir comandos y datos (data sets) una vez que la asociación está establecida.
- ♣ *DimseUtil*: Las funciones DICOM que pueden hacerse.
- ♣ *ExtendedNegotiation*: Representa los datos de negociación para una sintaxis abstracta.

- *Reject* (Rechazo): Representa el rechazo de una asociación. Sus parámetros son fuente, resultado y razón.
- *Request* (petición): Representa todos los parámetros relevantes de una asociación Request. Contiene todos los contextos de presentación y son numerados desde 0 hasta el número de contextos de presentación menos uno. Puede actuar como receptor de asociación o como iniciador de la asociación.
- *Response* (Respuesta): Representa una respuesta de un par de entidades DICOM. Las clases implementadas son: Reject, Acknowledge y Abort.
- *ResponsePolicy*: contiene dos tipos de métodos:
    - Métodos para creación de objetos Response, basados en una petición dada y una política de aceptación de la asociación.
    - Métodos para el análisis de respuestas Acknowledge, en el contexto de una petición dada.

Com.archimed.dicom.codec

- *Compression* (compresión): Es una clase que proporciona métodos para la descompresión de datos de píxel. DICOM soporta muchos tipos de técnicas de compresión.

Com.archimed.dicom.image

- *DicomImage*: Esta clase provee de métodos para construir una imagen Dicom.
- *GrayColorModel*: Esta clase representa un ColorModel para el empleo con imágenes en escala de grises.
- *ImageIO*: Es una clase que proporciona métodos para convertir datos de imagen almacenados en un ImageProducer (Productor de imagen) y viceversa.
- *SCImage*: Esta clase proporciona métodos para construir una imagen SC.
- *WL*: Es un contenedor básico para un par Ventana/Nivel.

Com.archimed.dicom.tools

- *Sequences*: Estos objetos proveen los "atajos" para coger/poner valores dentro de secuencias. La conversión de valores está hecha usando el esquema de conversión Tipo Dicom-Tipo Java dado en la clase DicomObject.

Com.archimed.dicom

- *DDate*: Esta clase representa la fecha de un objeto, conteniendo el día, mes y año. Se ha desarrollado en correspondencia al tipo Dicom DA. Es útil, cuando se quiere buscar en una gama de fechas.
- *DDateRange*: Esta clase representa la gama de fechas.
- *DDict*: Esta clase provee de un diccionario (Diccionario de datos Dicom) para los VRs y todos los elementos de datos. Cada par (grupo,elemento) es accesible por una variable del tipo static int. Por ejemplo, (0010,0010) es representado por DDic.dPatientName.
- *DDictEntry*: Un objeto para una etiqueta que puede ser almacenado en el diccionario de datos.
- *Debug*: Proporciona una variable del tipo static int por si hay que imprimir información de reparación de los errores a System.err (salida estándar de errores).
- *DicomObject*: Esta es la clase base de todos los datasets. Los métodos de acceso proporcionados aquí son el modo de obtener/poner elementos de dato (data elements) dentro de un objeto Dicom (DicomObject).
- *GroupList*: Provee de métodos para el acceso de datos por grupo.
- *Jdt*: Para obtener información sobre este JDT
- *MetaSOPClass*: Extensión de la clase UID.
- *Offsets* (Compensación): Esta clase proporciona utilidades para calcular compensaciones en un objeto Dicom (DicomObject) que va a ser escrito.
- *Person*: Esta clase representa el PN (Person Name) en Dicom.
- *SOPClass*: Extensión de la clase UID.
- *SOPInstance*: Extensión de la clase UID.
- *TagValue*: Representa un par (grupo, elemento) y su valor.
- *TransferSintax*: Extensión de la clase UID.

- *UID*: Esta clase contiene un depósito de todos los UIDs Dicom certificados. Cada UID está en función de: Transfer Syntax, SOPClass, MetaSOPClass y SOPInstance.
- *UIDEntry*: Representa la entrada de un UID en el depósito, almacenándose.
- *DicomException*: Indica que la aplicación ha violado la estructura y codificación del Dicom Data.
- *IllegalValueException*: Indica que un valor ilegal es leído durante una asociación o cuando un valor ilegal es argumento de un método.
- *UnknownUIDException*: Indica que el UID incluido no está definido en la clase UID.

7.7. Instalación de JDT

Para poder utilizar estas librerías de Java es necesario hacer una instalación manual como la realizada con las JDK. También al realizar el programa se han utilizado las librerías JDT que se distribuyen de forma gratuita por lo que tienen un periodo de validez; este periodo se acaba el 23 de Abril del 2004, pues bien para poder utilizarlas, lo único que hay que hacer es retrasar el reloj del ordenador hasta el 23 de Abril del 2003 y así tenemos un año completo para poder utilizar las librerías.

Para realizar la instalación es común en todos los sistemas operativos el crear una carpeta que se llame como se quiera, en nuestro caso se llama *classpath*. La página para hacer la petición de las librerías es www.softlink.be. En la carpeta que hemos creado hay que guardar los archivos enviados por softlink que son *jdt.jar* y *jdt.key*.

En nuestro caso hemos creado una carpeta llamada classpath en el disco C:\, por lo que hay que copiar los archivos jdt.jar y jdt.key en *C:\classpath*.

7.7.1. Instalación en Windows 95/98

Para la instalación de las librerías JDT en este sistema operativo, hay que abrir el archivo *Autoexec.bat* que se encuentra en C:\ con un editor de texto (pinchamos con el botón derecho del ratón sobre el archivo mientras se deja pulsado "Shift", y elegimos el Bloc de Notas). En la variable classpath creada anteriormente cuando instalamos las

librerías JDK, hay que escribir el path o camino para llegar a los archivos de las librerías JDT. En nuestro caso es:

C:\classpath\jdt.key; C:\classpath\jdt.jar;

Una vez realizado esto, hay que hacer una copia del archivo "jdt.key" en los directorios donde haya archivos fuente que utilicen estas librerías. Por ejemplo, si tenemos en la carpeta C:\java un archivo fuente main.java que utilice estas librerías, hay que poner en la carpeta al archivo "jdt.key".

7.7.2. Instalación en Windows NT/2000/XP

Para la instalación de JDT, hay que entrar en las variables de entorno. Para llegar a las variables de entorno ir a Inicio • Configuración • Panel de Control • Sistema • Avanzado • Variables de entorno... y en variables del sistema pinchar en la variable classpath creada anteriormente, ahí añadir:

C:\classpath\jdt.key; C:\classpath\jdt.jar;

Una vez realizado esto, hay que hacer una copia del archivo "jdt.key" en los directorios donde haya archivos fuente que utilicen estas librerías. Por ejemplo, si tenemos en la carpeta C:\java un archivo fuente main.java que utilice estas librerías, hay que poner en la carpeta al archivo "jdt.key".

7.7.3. Incluir JDT en JBuilder

Como dijimos antes, la aplicación se desarrolló mediante el entorno JBuilder de Borland, de modo que vamos a explicar aquí la forma en que debemos proceder para que dicho entorno pueda hacer uso de las librerías JDT. Una vez instaladas las librerías en el PC, hay que incluirlas en el proyecto en el que se esté realizando una aplicación DICOM. Este proceso es el mismo para cualquier tipo de librerías que se quieran utilizar. Para ello hay que realizar los siguientes pasos:

1. En la barra de herramientas seleccionar Proyecto

2. Propiedades del proyecto. . .
3. Ahora en la pestaña de *Vías de acceso* y dentro de esta en la pestaña, en la de *Bibliotecas necesarias*, se pulsa *Añadir...* y seguidamente *Nuevo…* Mediante el asistente de bibliotecas se añaden las JDT:

   - ♣ Nombre: JDT
   - ♣ Ubicación: Directorio Inicial
   - ♣ Pulsar Añadir...
   - ♣ Seleccionar el camino hasta las librerías JDT que en nuestro caso son:
     - D:\classpath\jdt.jar
     - D:\classpath\jdt.key
     - D:\classpath\
   - ♣ Aceptar

4. Asegurarse que las bibliotecas se han añadido bien entrando en Herramientas
   - Configurar bibliotecas...
5. Hecho esto ahora podemos añadir las clases de estas bibliotecas a nuestro código fuente sin que dé error. Por ejemplo:

   import com.archimed.dicom.;*

Con esto podemos declarar nuevos objetos de las clases de JDT, por ejemplo:

DicomObject CT = new DicomObject();

Las librerías instaladas ya son utilizables en el proyecto y se puede trabajar con ellas.

7.8. Conclusiones

A la hora de elegir una herramienta para desarrollar nuestra aplicación, se presentaban a priori dos posibilidades, es decir, dos conjuntos de librerías para

implementar funciones acordes al estándar DICOM: las DCMTK[2] de *Offis* (dcmtk351) y las JDT de *Softlink.* Analizadas las posibilidades que ambas librerías ofrecían, se llegó a las siguientes conclusiones:

El toolkit dcmtk351 cuenta con la ventaja de ser un producto gratuito, pero si no se es un gran experto en materia DICOM y sobre todo en programación en C++, la profundización de su estudio es muy complicada y engorrosa, debido a la poca documentación proporcionada con dicha herramienta y a la gran dificultad y amplitud del código.

Por el contrario, JDT es un producto no gratuito, pero cuenta con las grandes ventajas que el toolkit dcmtk351 no tiene. Lo primero a señalar es que el lenguaje de programación es JAVA, con las ventajas que conlleva con respecto a C++. Por otro lado, contiene una documentación amplia y muy bien estructurada, gracias a la utilidad *javadocs*, lo que facilita enormemente el trabajo y su comprensión. JDT contiene un árbol de clases, información de los diferentes packages, información sobre las clases y sus métodos. Además, las aplicaciones Java para arquitectura de red en modo Cliente – Servidor resultan muy potentes y eficientes. Por todos los motivos que se han explicado, la elección final fue emplear el JDT para el desarrollo de la aplicación cliente.

[2] Dicom Toolkit