# APLICACIÓN MÓVIL PARA LA CONSULTA Y VISUALIZACIÓN DE IMÁGENES USANDO EL PROTOCOLO DE COMUNICACIONES DICOM.

.

JHON FREDDY MONTENEGRO ÁLVAREZ

UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

CALI

2014

# APLICACIÓN MÓVIL PARA LA CONSULTA Y VISUALIZACIÓN DE IMÁGENES USANDO EL PROTOCOLO DE COMUNICACIONES DICOM.

JHON FREDDY MONTENEGRO ÁLVAREZ

**Trabajo de grado para optar por el título de**

**Ingeniero Electrónico**

Director:

Profesor titular:

ING. FABIO GERMAN GUERRERO Msc.

UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

CALI

2014

# Tabla de contenido

CAPITULO 5



CAPITULO 6

## LISTA DE FIGURAS

**LISTA DE TABLAS**

# Capítulo 1

# INTRODUCCIÓN

## 1.1 Introducción

El acceso a la salud constituye uno de los factores fundamentales para garantizar condiciones de vida adecuadas a la población de cualquier país y en tal sentido el ofrecer servicios de salud con cobertura y calidad adecuados constituye una de las principales responsabilidades de los estados y de las sociedades hacia sus ciudadanos.

El cumplimiento de lo anterior atraviesa grandes dificultades, por la gran demanda de atención, que en muchos casos no alcanza a ser cubierta por el personal de salud; las distancias que deben recorrer los pacientes para acceder a los centros donde se localiza el personal médico con el conocimiento y tecnología necesarios para atenderlos, e incluso pacientes con discapacidades o personas de la tercera edad. El modelo de prestación de salud que hoy en día se tiene es el del personal médico concentrado en centros hospitalarios, donde cuentan con tecnologías avanzadas para diagnóstico y tratamiento de enfermedades, y con infraestructura de habitaciones y camas que hospedan a los pacientes que requieren un tratamiento con supervisión continua.

Sin embargo, el avance de las TIC está planteando un modelo alternativo, donde es la información, y no el paciente, la que se desplaza de un lugar a otro. Este concepto, que consiste en la prestación de servicios de salud a distancia, recibe el nombre de telemedicina, y permite, entre otras cosas, establecer canales de comunicación entre médico y paciente, entre médicos ubicados en lugares distintos, y que entre ellos puedan compartir información e imágenes de diagnóstico, que permitan hacer una evaluación, sin necesidad del contacto físico. De este modo y apoyado en dispositivos informáticos y de telecomunicaciones, el médico puede hacer diagnósticos, administrar tratamientos y hacer seguimiento a los mismos, sin que el paciente tenga que estar presente.

# Capítulo 2

# El Estándar DICOM

## 2.1. Introducción

DICOM está basado en conceptos característicos dentro del entorno de la programación orientada a objetos, cuyos estudios son representados como objetos con sus respectivos atributos y métodos denominados servicios, que definen los tipos de operaciones aplicables a dichos objetos, tales como almacenamiento, impresión, búsqueda en una base de datos e intercambio de datos entre diferentes medios.

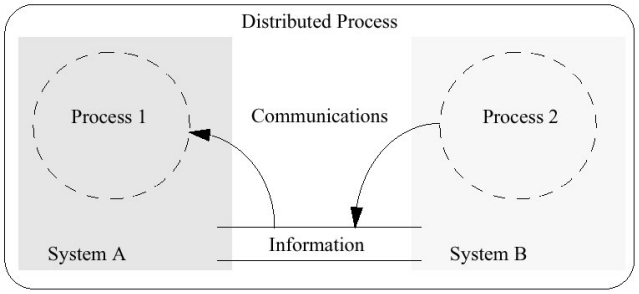
El estándar DICOM fue concebido como un documento multi-partes, las cuales se encuentran relacionadas pero con cierto grado de independencia, facilitando una posible actualización de cada una de ellas sin afectar considerablemente el contenido de las otras. DICOM fue desarrollado siguiendo las directrices de ISO/IEC de 1989 donde en la parte tres se define "el bosquejo y la presentación de estándares internacionales".

DICOM pertenece al campo de la medicina y facilita el intercambio de información digital entre equipos de imágenes médicas y otros sistemas al igual que la interoperabilidad de equipos médicos especificados; es de aclarar, que el estándar facilita pero por si solo no garantiza interoperabilidad, además, no especifica un procedimiento de validación para evaluar la conformidad de una implementación al estándar.

## 2.2. Procesamiento distribuido

Un simple modelo de un proceso distribuido servirá para explicar el mecanismo y terminología usada en el estándar DICOM. Figura 1.

Figura 1. Modelo simple proceso distribuido.



Un proceso distribuido está formado al menos por dos procesos que comparten información y confiando en la operatividad de uno en el otro. Un número de procesos distribuidos actuando juntos proporcionan ***servicios*** para sistemas en entornos como departamentos de radiología. Antes de que los procesos puedan actuar juntos una serie de temas tienen que ser tratados. Tienen que estar de acuerdo en el papel (rol) que cada uno desempeñará, tener una visión equivalente de la información y seleccionar las operaciones que cada parte realizará.

Primeramente, el papel de cada parte debe ser definido como ***cliente*** o como ***servidor***. La parte que utiliza la operatividad de la otra, tiene el papel de cliente. La parte contraria actuando sobre un modelo concertado tiene el papel de servidor. El funcionamiento de ambas partes viene definido por la relación que comparten. ***La relación*** define que parte y bajo que condición toma la iniciativa en el proceso. En muchos casos los clientes provocan el proceso, pero a veces lo hace el servidor.

Además de los papeles que desempeñan, ambas partes tienen que estar de acuerdo en la información que intercambian. ***La información*** está definida por el contexto del servicio que el proceso distribuido está realizando. Cada proceso individual tendrá una visión selectiva de esta información, pero la visión tiene que ser coherente en la totalidad del contexto.

***La operación*** define como debe ser procesada la información intercambiada en la otra parte, tal como almacenar información, devolver un resultado, etc.

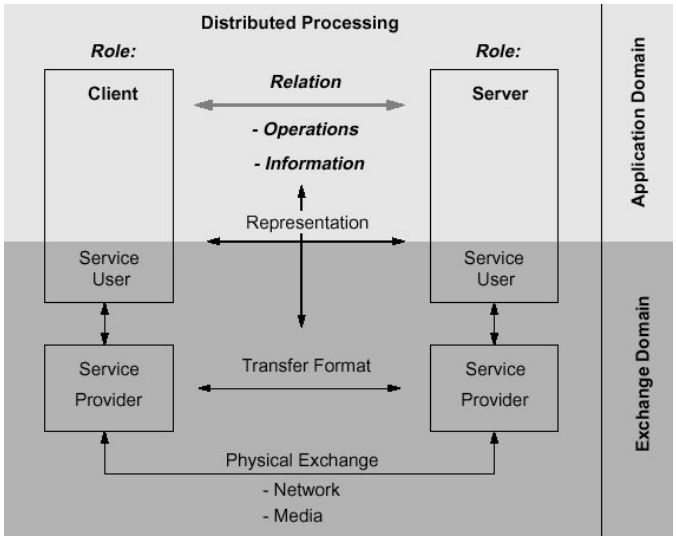
*La combinación del contexto, relación, operaciones e información es la piedra fundamental del procesamiento distribuido y tiene que ser definida antes de que una aplicación pueda ser realizada*. Todas estas cuestiones son parte del dominio de la aplicación (***application domain***) de los procesos distribuidos. Estos no se ocupan de la forma en que la información es intercambiada actualmente, pero cuentan con los servicios de menor nivel (p.e. TCP/IP) suministrados por el dominio del intercambio ( ***exchange domain***) para poder hacer frente al intercambio.

Ambas partes, cliente y servidor, tienen que ser capaces de emitir peticiones a los servicios de menor nivel. Los servicios de menor nivel llevarán el intercambio y están ocultos para el dominio de la aplicación del cliente o servidor. La parte que solicita los servicios es el usuario del servicio (***service user***). El equivalente es el proveedor del servicio (***service provider***).

El proveedor del servicio debe determinar en qué formato la información fue transferida y convertida a la representación esperada por el dominio de la aplicación. La representación es conocida entre el usuario y el proveedor del servicio en cada parte. Después del intercambio, la información presentada a los procesos utilizando la información es igual en ambas partes, independientemente de cómo fuera intercambiada.

El intercambio físico entre los proveedores del servicio puede ser vía network o media. Cada mecanismo tiene su propia forma de manejar el conocimiento de la representación.

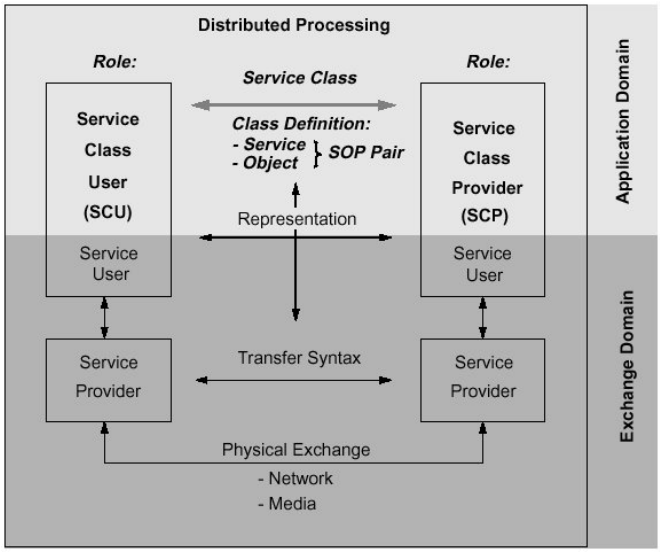
Figura 2. Modelo de un proceso distribuido DICOM.




## 2.3. Conceptos generales de Dicom

***Los objetos*** (pacientes, imágenes, reportes, etc.) y sus interrelaciones son descritos mediante modelos de entidad-relación. DICOM utiliza su propia terminología para describir el contexto, relaciones, asociaciones, etc. El primer paso es el mismo modelo que para el procesamiento distribuido con la transformación de la figura 2 en la figura 3 aplicando los términos equivalentes de DICOM.

Figura.3. Clases de Servicio DICOM.




Los objetos definidos en los modelos son llamados ***entidades***. Los ***atributos*** describen las características de un objeto. Cuando se dan valores a los atributos de un objeto, definiendo un

objeto real, existente, se habla de una ***instancia del objeto***. Los objetos se agrupan en clases, según su tipo. Las clases se comunican entre sí, mediante ***mensajes***. DICOM define las clases de objetos y sus mensajes permitidos en lo que es llamado **SOP** (***Service-Object Pair***). Cuando un equipo especifica que es compatible con una clase SOP, es posible saber de forma no ambigua como se entenderán sus datos, ya sea por el proveedor de los servicios de la clase: **SCP (Service Class Provider)** o por el usuario de los servicios de la clase: el **SCU (Service Class User).** Para realizar una asociación, lo primero es que SCU y SCP estén de acuerdo en utilizar una clase SOP.

### 2.3.1 Clases de Servicio (Service Classes) y Clases SOP (SOP Classes)

La relación entre ambas partes es definida por la descripción de la Clase de Servicio. La ***Clase de Servicio*** describe explícitamente los papeles que ambas partes desempeñan. Dependiendo de la Clase de Servicio el contexto del servicio está definido. Con DICOM ambos papeles son llamados: *Usuario de la Clase de Servicio* o **SCU** (***Service Class User***) (***cliente***) y *Proveedor de la Clase de Servicio* o **SCP** (***Service Class Provider***) (***servidor***). No hay que confundir SCU y SCP con el usuario del servicio y el proveedor del servicio del dominio del intercambio.

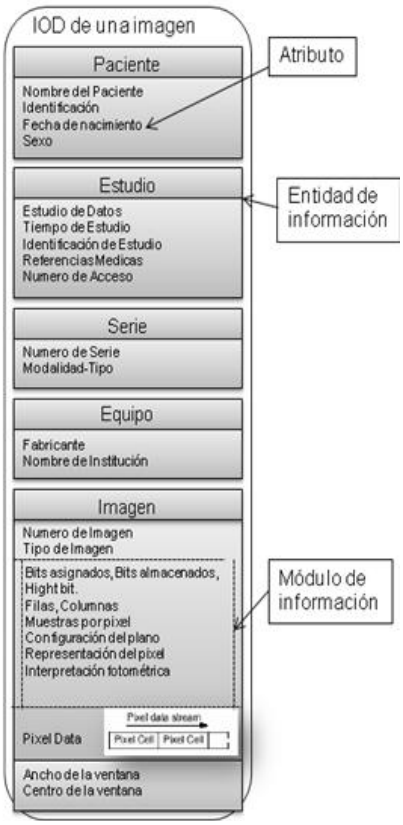
*Parte de la Clase de Servicio es la descripción de la información y operaciones*. En DICOM estas están combinadas con la definición de la clase, llamada **Clase de Servicio de Par Objeto** (Service Object Pair Class) o **Clase SOP** (SOP Class). *En cada definición de Clase SOP una única Definición de Objeto de Información (Information Object Definition) o IOD es combinado con uno o más servicios.*

Cuando se está estableciendo una conexión DICOM, por ejemplo para enviar imágenes de un dispositivo TAC a una estación de trabajo de propósito general, ambos dispositivos deben negociar primero sus respectivas prestaciones. Por ejemplo, la estación de trabajo podría soportar únicamente la recepción de imágenes de TAC y RMN, pero no de Ultrasonidos. Esto se debe a que se requieren aplicaciones software y a menudo hardware (por ejemplo, una pantalla en color) para poder mostrar imágenes de Ultrasonido. Así pues, durante este proceso de negociación quedan claras cuáles prestaciones soporta el dispositivo receptor y cuáles no. Cada prestación, tal como recibir imágenes de TAC, RMN o Ultrasonidos, y el soporte para impresión DICOM o consultar su base de datos, se identifica claramente y se especifica en el estándar como las llamadas ***Clases SOP***. De esa forma cada dispositivo tiene su propia "***lista de selección***" de prestaciones que soporta dependiendo de su funcionalidad. Por ello, el control de versiones es parte inherente del proceso de negociación DICOM, porque cuando un dispositivo no reconoce una nueva Clase SOP, devolverá un código de error. Adicionalmente, debe aparecer especificado en la ***Declaración de Conformidad DICOM***. **[1]**

### 2.3.2. Definiciones de Objeto de Información (Information Objects Definitions)

La parte de información de una Clase SOP es definida en los IODs. Un **IOD** es una colección de partes de información relacionada, agrupadas en *Entidades de Información* (Information Entities) o atributos. Cada entidad contiene información sobre un único objeto (mundo real) como un paciente, una imagen, etc. Dependiendo del contexto definido por la Clase de Servicio, un IOD consiste en una entidad de información única llamada ***IOD normalizado*** (normalized IOD) o una combinación de entidades de información llamada ***IOD compuesto*** (composite IOD). Las Clases de Servicio que llevan a cabo funciones de administración (en su mayor parte cuestiones simples) utilizan IODs normalizados, aquellas que manejan el flujo de imágenes (estructura compleja de información) utilizan IODs compuestos. Cada Clase SOP es definida con uno o más IODs que son combinados con uno o más servicios.

Figura 4. Estructura de un IOD compuesto.




Las entidades de información consisten en atributos, describiendo una única parte de información, por ejemplo, el nombre de un paciente. Los atributos que tienen una relación están agrupados en módulos de información de objetos o **IOMs** (***Information Object Modules***). Los IOMs están definidos de tal manera que pueden ser usados en más de un IOD. Estos IOMs también tienen la ventaja de que las descripciones semánticas de los atributos descritos pueden ser agrupados juntos.

### 2.3.3. Atributos

Los **atributos** son la entidad de información básica y tienen que ser descritos en detalle. Figura 5. Las siguientes características o campos de un atributo están definidas en el estándar DICOM:

Un único **Nombre de Atributo** (Attribute Name) (legible por el ser humano).
Una única **Etiqueta de Atributo** (Attribute Tag) (legible por los sistemas de información).
Una **Descripción de Atributo** (Attribute Description) (semántica).
Un **Valor de Representación** (Value Representation) (sintaxis).
Un **Valor de Multiplicidad** (Value Multiplicity).
**Tipo de clasificación**: 1, 1C, 2, 2C o 3 (usadas dependiendo del contexto de las Clases SOP, Clases de Servicio, papel que desempeña, etc.).

Figura 5. Atributo de información.

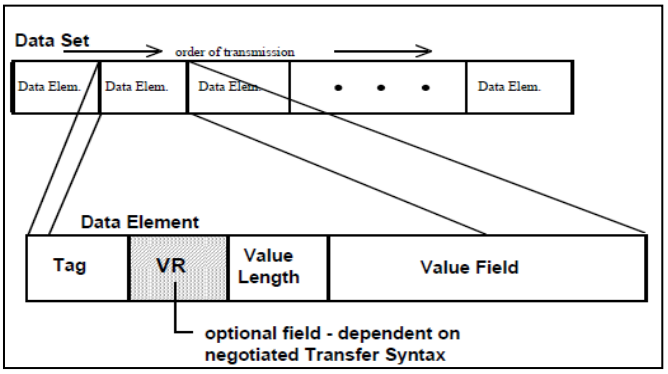
| Tag | Name | Keyword | VR | VM | |
|---|---|---|---|---|---|
| *(0008,0001)* | *Length to End* | *LengthToEnd* | *UL* | *1* | *RET* |
| (0008,0005) | Specific Character Set | SpecificCharacterSet | CS | 1-n | |
| (0008,0006) | Language Code Sequence | LanguageCodeSequence | SQ | 1 | |
| (0008,0008) | Image Type | ImageType | CS | 2-n | |
| *(0008,0010)* | *Recognition Code* | *RecognitionCode* | *SH* | *1* | *RET* |
| (0008,0012) | Instance Creation Date | InstanceCreationDate | DA | 1 | |

Tabla 1. Atributos del módulo Paciente.

| Attribute Name | Tag | Type |
|---|---|---|
| **Patient´s Name** | 0010, 0010 | 2 |
| **Patient´s ID** | 0010, 0020 | 2 |
| **Patient´s Birth Date** | 0010, 0030 | 2 |
| **Patient´s Sex** | 0010, 0040 | 2 |
| **Patient´s Birth Time** | 0010, 0032 | 3 |
| **Other Patient ID** | 0010, 1000 | 3 |
| **Other Patient Names** | 0010, 1001 | 3 |
| **Ethnic Group** | 0010, 2160 | 3 |
| **Patient Comments** | 0010, 4000 | 3 |

Tales atributos se codifican como ***elementos de datos*** (*data element*), que son unidades de información que poseen valores codificados para los diferentes atributos del IOD. Estos elementos de datos poseen campos de información que definen su estructura.

Figura 6. Estructura del Data Element.




- **Etiqueta (Tag):** Es un par ordenado de números de 16 bits codificados en hexadecimal. El primero de ellos hace referencia al *Grupo*, mientras el segundo al *Elemento*, y son leídos como enteros sin signo. Sirve para identificar cada elemento de datos de forma unívoca.

- **Representación del valor (VR):** Indica la forma en que se codifica el Valor del elemento o atributo. Por ejemplo el valor puede estar codificado como una cadena de caracteres o un entero sin signo. *El campo VR no siempre está codificado en el elemento de datos, sino que depende de la sintaxis de transferencia*.
- **Longitud del valor (Value Length):** Entero sin signo de 16 o 32 bits (según sea VR Explícito o Implícito). Contiene un número par que indica la extensión del campo Valor, donde se hallan contenidos los datos propiamente dichos.
- **Valor (Value):** Es el valor del elemento de datos, codificado según el campo VR y con la longitud que indica el campo Longitud del Valor.

En el documento número 6 del estándar están las descripciones de todos los elementos de datos, ordenadas según la etiqueta. En el documento número 3 del estándar se explica el propósito de cada uno de ellos, así como su obligatoriedad.

### 2.3.4. Elementos de Servicio (Service Elements)

Los **Elementos de Servicio** son las operaciones permitidas en los Objetos de Información (IODs) para una Clase SOP definida. El grupo de elementos de servicio pertenece a la Clase SOP y es llamada ***Grupo de Servicio*** (Service Group). Cada Clase SOP utiliza uno o más Elementos de Servicio tanto del grupo compuesto (C-XXXX) o del grupo normalizado (N-XXXX). El estándar dispone de los siguientes elementos de servicio:

| | |
|---|---|
| C-STORE | N-GET |
| C-FIND | N-SET |
| C-MOVE | N-ACTION |
| C-GET | N-CREATE |
| C-CANCEL | N-DELETE |
| C-ECHO | N-EVENT-REPORT |

Una Clase SOP utiliza uno o más elementos de servicio.

### 2.3.5. Instancias SOP (SOP Instances)

El esqueleto de las definiciones citadas anteriormente toma forma cuando se utilizan en un proceso distribuido. Después del acuerdo que mantienen las Clases SOP (e implícitamente la Clase de Servicio), y cómo los papeles de la SCU y la SCP están divididos, las instancias de la clase SOP pueden ser distribuidas entre las dos partes. Los Atributos tienen que ser proporcionados con los valores correctos y almacenado en la Instancia SOP como se especifica en la definición de los atributos.

Después de recopilar la información, esta será codificada a los formatos definidos por DICOM, utilizando la Representación del la Etiqueta (Tag) y del Valor (Value) para crear un DICOM data set, en el cual cada atributo es codificado en un data element. Este "***data set***" es manejado por el

proveedor del servicio de intercambio, el cual garantiza que la parte contraria recibe idéntico data set. Las diferencias en la representación del sistema especificado son tomadas en una cuenta durante el intercambio, asegurando que los significados semánticos permanecen intactos.
El receptor del data set decodificará este para extraer la información que necesita y actuar como acuerdo de la semántica de la Clase SOP.

Para facilitar el intercambio de información entre los sistemas digitales de imágenes en entornos médicos, la información es codificada en forma de Data set.

**El Data set** es la porción de un mensaje DICOM que transmite información acerca de objetos del mundo real gestionada sobre la red; consiste en la sucesión de una serie de campos con valores codificados de los atributos del objeto, incluyendo el propio objeto, estos campos son denominados como *elementos de datos* (*Data Elements*). En la figura 7 se puede observar la estructura del Data set.

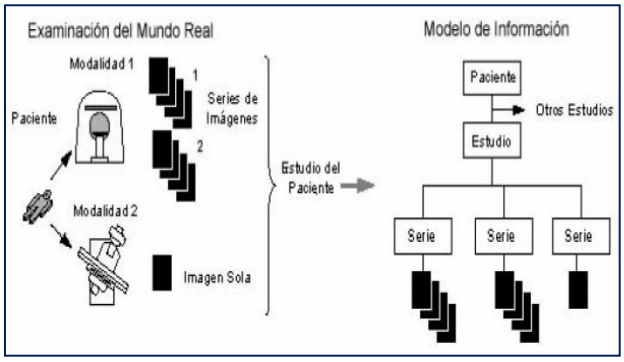
## 2.4. Modelo de información Dicom

El manejo electrónico de la información requiere un modelo para representar la forma en que la información está estructurada. Esta estructura es necesaria para tener instancias uniformes y para hacer posible la descripción de las relaciones entre instancias de forma clara. Un modelo de información de imagen deriva de la forma en que las imágenes se manejan en un departamento de radiología. Las imágenes recogidas de uno u otro aparato, son recopiladas en una carpeta perteneciente al paciente correspondiente. Las imágenes son ordenadas en la carpeta conforme al tipo de examen realizado (series de imágenes que están relacionadas).

Cuando los datos de las imágenes de diferentes fuentes tienen que ser recogidas en un ambiente único, debe ser posible ordenar los datos de las imágenes de diferentes fuentes. Esto es sólo posible cuando los datos están estructurados de acuerdo al mismo modelo de información.

El modelo de información de imagen DICOM está basado en suposiciones sobre la forma en que la información de diferentes aparatos están relacionados. Ver figura 8. Los cuatro niveles de este modelo de información son paciente, estudio, serie e imagen.

Figura 7. Modelo de información de imágenes DICOM.



- El **nivel de paciente** contiene la información de identificación y demografía del paciente al que pertenece el estudio. Un paciente puede tener uno o más estudios, por lo que este nivel se considera el más alto.

- El **nivel de estudio** es el más importante dentro del modelo de información. Un estudio es el resultado de una solicitud de cierto tipo de examen médico. En general, una solicitud puede involucrar procedimientos de exámenes médicos de diferentes modalidades. Los resultados se tendrán en series de una o más imágenes. Todos los datos de imágenes se colocan junto con el mismo estudio como raíz.

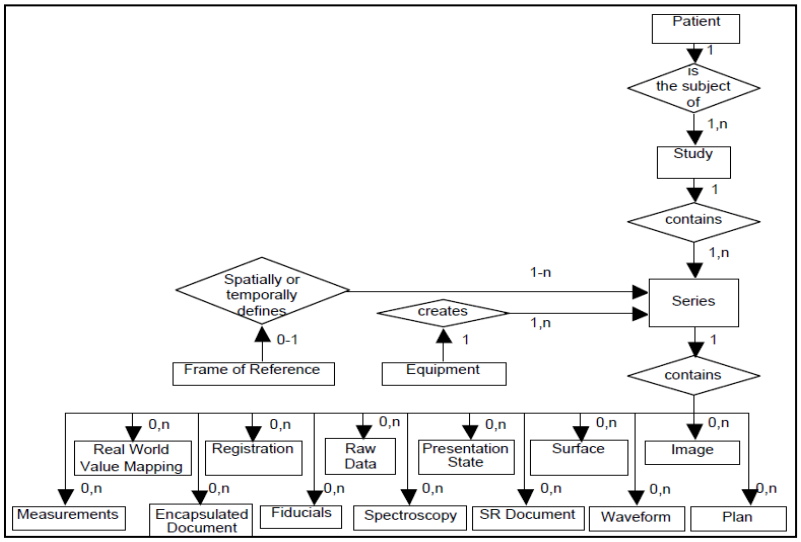
- **Nivel de serie:** abajo del nivel de estudio se reúnen todas las series de imágenes. En este nivel se identifica el tipo de modalidad de las imágenes, fecha y hora de creación, detalles acerca del tipo de examen médico y equipo utilizado. *Las series son un conjunto de imágenes relacionadas que provienen de una sola modalidad.*
  Cuando las adquisiciones en una secuencia tienen relación espacial o temporal, las imágenes resultantes de esta adquisición pueden ser agrupadas en series. Una serie nueva debe comenzar cuando la relación existente entre imágenes ya no existe.

- **Nivel de imagen:** el nivel más bajo del Modelo de Información es el nivel de imagen. Cada imagen contiene información de adquisición y posición, así como también los datos propios. Dependiendo del tipo de modalidad, el nivel de imagen contiene datos para una sola imagen, dos imágenes (sistema de dos planos) o una colección de imágenes tomadas a partir de una colección de datos (imágenes) en un período corto de tiempo (*multiframe images).*

### 2.4.1. Instancias imagen SOP

El modelo de información mostrado en la figura 8 es una simplificación del modelo de información completo de imagen DICOM de la figura 9. Cada bloque en este diagrama representa una entidad de información (ver IODs) del IOD compuesto. Las relaciones indican los puntos cardinales para cada relación de la entidad de información usada en una instancia SOP.

Figura 8. Modelo de información de un IOD compuesto.



Las entidades de información sobre el nivel de serie deberían contener información perteneciente al estudio y al paciente que debe ser comparable con la información de otros aparatos. La mayoría de esta información viene de fuentes externas como un sistema de planificación. Puede ser suministrado al aparato por la interfaz de usuario o mediante una conexión a un sistema de información.

### 2.4.2. Tipos de imágenes

DICOM define un número de tipos de clases de imágenes SOP, dependiendo del aparato médico que crea los datos de las imágenes. Cada tipo tiene su propio IOD para añadir información específica del aparato a la instancia de la imagen SOP.

Todas las instancias de las imágenes SOP comparten un mínimo juego de información que permite a una aplicación visualizadora manejar las imágenes independientemente de su tipo. Una clase de imagen SOP está disponible para encapsular las imágenes que no están disponibles en el formato digital y sí capturadas en formato de película o de video.

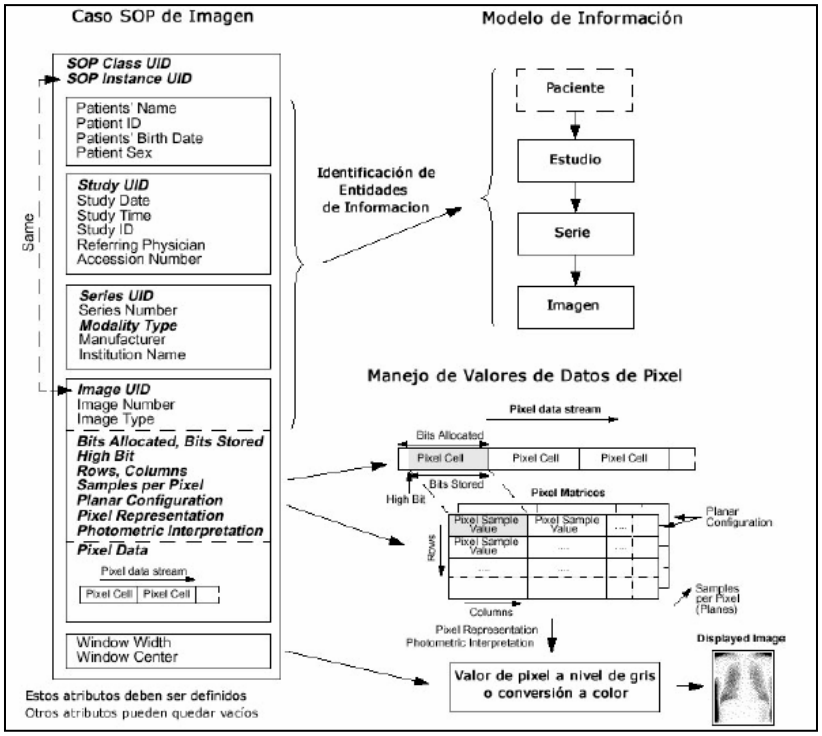
#### 2.4.2.1. Tipos genéricos de imágenes

Las instancias de las clases de las imágenes SOP tienen un conjunto básico de atributos; ver figura 10. El conjunto mínimo de atributos requeridos para una instancia de imagen SOP consiste en el siguiente grupo de atributos:

- *Atributos identificadores*: UID de clase SOP, UID de la instancia del estudio, UID de la instancia de la serie y UID de la instancia de la imagen (= UID de la instancia SOP).
- Tipo de aparato.
- Descripción de la matriz de píxeles: muestra por píxel, filas, columnas.
- Interpretación del valor del píxel: interpretación fotométrica.
- Codificación de los píxeles: bits asignados, bits almacenados, bit alto, representación de pixel, configuración plana.
- Matriz de píxeles: datos de píxel.

Este mínimo conjunto permite mostrar los datos de píxel y proporciona la identificación en el nivel de sistema, para el caso de la instancia SOP para adherirla al modelo de información. Añadiendo más información al menos para los tres primeros niveles del modelo de información, hace más entendible a la instancia SOP.

Figura 9. Conjunto mínimo de atributos para una instancia SOP de imágenes.




Los atributos que identifican la instancia SOP para seres humanos y permiten que la imagen sea mostrada con los correctos ajustes de ventana son:

- Nivel de paciente: nombre, ID, día de nacimiento, sexo.
- Nivel de estudio: fecha del estudio, hora, nombre del médico, ID del estudio, número de acceso.
- Nivel de serie: número de serie, fabricante, nombre de la institución.
- Nivel de imagen: número de imagen, tipo de imagen.
- Ajustes de presentación: ancho de ventana, centro de ventana.

Los atributos listados arriba son en la mayoría de los casos atributos del tipo 2 (deben ser suministrados, pero pueden faltar) o del tipo 3 (opcionales).

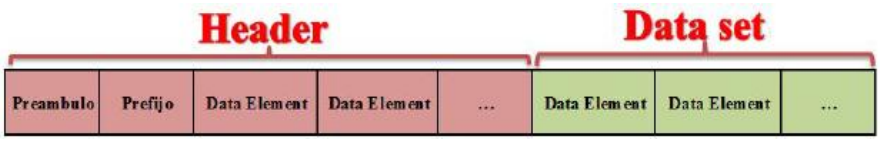
## 2.5. Estructura de un fichero Dicom

DICOM está basado en conceptos que son habituales en el entorno de la Programación Orientada a Objetos (OOP), en el estándar las imágenes o un estudio se representan como un objeto, los cuales tienen atributos y métodos (llamados servicios) donde cada uno de estos definen los tipos de operaciones que se pueden aplicar a dichos objetos, tales como almacenarlos, imprimirlos, buscarlos en una base de datos o moverlos de un lugar a otros.

Los objetos en DICOM se definen como Objetos de Información y las operaciones o servicios comúnmente llamados métodos en el entorno OO, son llamados ***Clases de Servicio***.

Un archivo en formato DICOM presenta en forma general la estructura mostrada en la siguiente figura. Consiste de dos partes: ***Header*** seguido por un ***Data Set*** de DICOM.

Figura 10. Estructura general de un archivo DICOM.

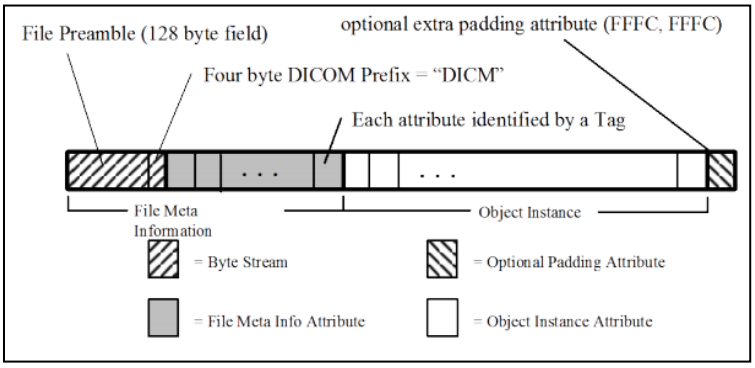



Un archivo DICOM consiste de:

- ***Preamble***: comprende 128 bytes, seguido por,
- ***Prefix***: comprende los caracteres "D", "I", "C", "M" seguido por,
- ***File Meta Header***: este comprende, entre otros, el Media SOP Class UID, Media SOP Instance UID y el Syntax Transfer UID. Por defecto, estos son codificados en explicit VR, Little Endian. Los datos son leídos e interpretados dependiendo del tipo VR.
- ***Data Set***: comprende un número de elementos DICOM, caracterizados por etiquetas y sus valores.

En la siguiente figura se observa la estructura del archivo Dicom en forma más detallada.

Figura 11. Estructura detallada de archivo DICOM.




El **Header** y el **Data Set** de un fichero DICOM consisten en una serie de campos con toda la información necesaria sobre la imagen en cuestión, incluyendo la propia imagen.
Al conjunto de toda la información codificada sobre un campo se le conoce con el nombre de ***Elementos de Datos*** (Data Element). Así, tanto el **Header** y el **Data Set** de un archivo DICOM consisten de una sucesión de Data Element.

El **Header** contiene datos generales sobre el archivo DICOM en cuestión. Estos datos están estructurados como Data Elements y deben ser codificados siempre usando la sintaxis de transferencia ***Explicit VR Little Endian***. Todos estos elementos de datos pertenecen al grupo 0002H.

El **Data Set** de un archivo DICOM, consiste de una serie de campos con toda la información necesaria sobre la imagen en cuestión, incluyendo la propia imagen. Entre estos campos se encuentra la información del paciente, del estudio realizado, series, sobre el tipo de imagen, la imagen y mucho más. Esta es la parte que va a proporcionar la gran mayoría de información que se necesita para visualizar la imagen correctamente, incluyendo los propios datos de la imagen. La información del Data Set está codificada según la sintaxis de transferencia indicada en la Cabecera.

Tabla 2. Cabecera de un archivo DICOM.

| Preamble | 128 bytes |
|---|---|
| **Prefix** | 4 bytes "D", "I", "C", "M" |
| **Group Length** | DATA ELEMENT |
| **File Meta Information Version** | DATA ELEMENT |
| **Media Storage SOP Class UID** | DATA ELEMENT |
| **Media Storage SOP Instance UID** | DATA ELEMENT |
| **Transfer Syntax UID** | DATA ELEMENT |
| **Implementation Class UID** | DATA ELEMENT |
| **Implementation Version Name** | DATA ELEMENT |
| **Source Application Entity Title** | DATA ELEMENT |
| **Private Information Creator UID** | DATA ELEMENT |
| **Private Information** | DATA ELEMENT |

Tabla 3. Data Set DICOM.

| | |
|---|---|
| PACIENTE | Nombre |
| | Identificación |
| | Fecha de nacimiento |
| | sexo |
| ESTUDIO | Identificador del estudio |
| | Fecha |
| | Hora |
| | Numero de acceso |
| SERIE | Identificador de la serie |
| | Numero |
| | Modalidad |
| EQUIPO | Fabricante |
| | Nombre de la institución |
| IMAGEN | Atributos de la adquisición |
| | Atributos de posición |
| | Numero de imagen |
| | Tipo de imagen |
| | Bits asignados, bits almacenados |
| | Bit superior |
| | Filas, columnas |
| | Muestras por pixel |
| | Configuración planar |
| | Representación del pixel |
| | Interpretación photometrica |
| | Data pixel |

La manera como están escritos los *Data Element* se denomina *sintaxis de transferencia*, la cual es generalmente igual para todos los elementos de un archivo. La sintaxis de transferencia determina si el Data Element está escrito en un ordenamiento ***Big*** o ***Little Endian***, si el valor de representación está o no incluido en el elemento (VR explicito o implicito) y el tipo de compresión de la imagen.

El estándar DICOM define las siguientes sintaxis de transferencia.

Tabla 4. Sintaxis de Transferencia.

| Formato | Sintaxis de Transferencia | Descripción |
|---|---|---|
| Nativo | 1.2.840.10008.1.2 | Implicito VR, Little Endian |
| | 1.2.840.10008.1.2.1 | Explicito VR, Little Endian |
| | 1.2.840.10008.1.2.2 | Explicito VR, Big Endian |
| JPEG | 1.2.840.10008.1.2.4.50 | Baseline (1) |
| | 1.2.840.10008.1.2.4.51 | Extended (2,4) |
| | 1.2.840.10008.1.2.4.57 | Lossless, non-hierar. (14) |
| | 1.2.840.10008.1.2.4.70 | Lossless JPEG Imagen Compression |
| | 1.2.840.10008.1.2.4.80 | JPEG-LS Lossless Imagen Compression |
| | 1.2.840.10008.1.2.4.81 | JPEG-LS Lossy (Near-Lossless) Image Compression |
| | 1.2.840.10008.1.2.4.90 | JPEG 2000 Image Compression (Lossless Only) |
| | 1.2.840.10008.1.2.4.91 | JPEG 2000 Image Compression |
| | 1.2.840.10008.1.2.4.92 | JPEG 2000 Part 2 Multi-component Image Compression (Lossless Only) |
| | 1.2.840.10008.1.2.4.93 | JPEG 2000 Part 2 Multi-component Image Compression |
| RLE | 1.2.840.10008.1.2.5 | Run Lenght Encoding, Lossless |

## 2.6. Sintaxis de transferencia

Un concepto importante es la **Sintaxis de Transferencia**. En términos simples, indica si un dispositivo puede aceptar la data enviada por otro dispositivo. Cada dispositivo viene con su propia *Declaración de Conformidad DICOM*, que lista todas las sintaxis de transferencia aceptadas por el dispositivo. Una sintaxis de transferencia dice cómo se codifican los datos y mensajes transmitidos. La Parte 5 del estándar DICOM da la sintaxis de transferencia como un conjunto de reglas de codificación que permiten a las *Entidades de Aplicación* negociar sin ambigüedades las técnicas de codificación (es decir, estructura de elementos de datos, ordenamiento de bytes, compresión) que son capaces de soportar, permitiendo de ese modo a estas Entidades de Aplicación comunicarse. Una Entidad de Aplicación es el nombre de un dispositivo DICOM o programa utilizado para identificarlo de manera única. [2]

### 2.6.1. Sintaxis de transferencia Little Endian VR implícito

Esta sintaxis de transferencia se aplica a la codificación de todo el conjunto de datos (Data Set) DICOM. Esto implica que cuando un Data Set DICOM está codificado con la sintaxis de transferencia DICOM *Implicit VR Little Endian* los siguientes requerimientos se deben cumplir:

a) Los ***Data Elements*** contenidos en la estructura del Data Set deben ser codificados con VR implícito (sin un campo VR).
b) La codificación de la estructura general del Data Set (etiquetas de elementos de datos, longitud de valor y valor) debe estar en ***Little Endian***.
c) La codificación de los Data Elements del Data Set será la siguiente de acuerdo a su VR (representación de valor):

- Para todos los VR definidos en esta parte, excepto para los VR **OB** y **OW**, la codificación será en Little Endian.
- Para los VR **OB** y **OW**, la codificación deberá satisfacer las especificaciones siguientes, dependiendo de la etiqueta del Data Element:
  - ✓ El Data Element (7FE0, 0010) ***Pixel Data*** tiene el VR **OW** y será codificado en Little Endian.
  - ✓ El Data Element (60xx, 3000) ***Overlay Data*** tiene el VR **OW** y será codificado en Little Endian.
  - ✓ El Data Element (60xx, 3000) ***Waveform Data*** tendrá el VR **OW** y será codificado en Little Endian.

La sintaxis de transferencia DICOM ***Implicit VR Little Endian*** será identificada por un UID de valor "**1.2.840.10008.1.2**".

## 2.7. Imagen en Dicom

Existe una gran variedad de *Data Elements* que no siempre estarán definidos en su totalidad dentro de un archivo, asimismo habrán *Data Elements* que no aporten información relevante para ciertas necesidades. Por esto es importante saber que como mínimo el archivo debe contener los *Data Elements* mencionados en la Tabla 1 para la adecuada lectura de la imagen:

Tabla 5. Data Elements necesarios para la lectura de imagen DICOM.

| TAG | DESCRIPTION | TYPE |
|---|---|---|
| **(0028, 0002)** | Samples per Pixel | Int |
| **(0028, 0008)** | Number of Frames | Int |
| **(0028, 0010)** | Rows | Int |
| **(0028, 0011)** | Columns | Int |
| **(0028, 0100)** | Bits Allocated | Int |
| **(0028, 0101)** | Bits Stored | Int |
| **(0028, 0102)** | Hight Bit | Int |
| **(0028, 0103)** | Pixel Representation | Int |
| **(0028, 1050)** | Window Center | Int |
| **(0028, 1051)** | Window Width | Int |
| **(0028, 1052)** | Rescale Intercept | Int |
| **(0028, 1053)** | Rescale Slope | Int |
| **(7FE0, 0010)** | Pixel Data | Byte[] o Uint16[] |

A continuación se detallan algunos de estos Data Elements:

- (0028,0002) Samples por Píxel (Samples per Pixel)
- (0028,0004) Interpretación Fotométrica (Photometric Interpretation)
- (0028,0010) Filas (Rows)
- (0028,0011) Columnas (Columns)
- (0028,0100) Bits Destinados (Bits Allocated)
- (0028,0101) Bits Almacenados (Bits Stored)
- (0028,0102) Bits Más Significativo (High Bit).
- (0028,0103) Representación de los Píxeles (Pixel Representation)
- (7FE0,0010) Datos de los Píxeles (Pixel Data)

**Samples por Píxel (Samples per Pixel)**

Samples por Píxel (0028,0002) es el número de planos separados de la imagen. Se han definido imágenes de uno y tres planos. Se permiten imágenes formadas por un número distinto de planos, pero su significado no está definido por el estándar.
Para imágenes monocromas (escala de grises) e imágenes de paleta de colores, el número de planos es uno. Para imágenes RGB u otros modelos de color de tres vectores, el valor de este atributo es tres.

Todos los planos de la imagen deben tener el mismo número de filas (0028,0010), columnas (0028,0011), Bits Destinados (0028,0100), Bits Almacenados (0028,0101), Bit mas significativo (0028,0102), Representación de Pixel (0028,0103) y Relación de Aspecto de Pixel (0028,0034).

**Interpretación Fotométrica**

El valor del elemento de datos Interpretación Fotométrica (0028,0004) especifica la interpretación de los datos de los píxeles que componen la imagen.

- **MONOCHROME1**

Los datos de los píxeles representan un plano de imagen simple monocromo. El mínimo valor de sample debe ser visualizado como color blanco. Solo se debe usar si Samples por Píxel tiene el valor uno.

- **MONOCHROME2**

Los datos de los píxeles representan un plano de imagen simple monocromo. El mínimo valor de sample debe ser visualizado como color negro. Solo se debe usar cuando Samples por Píxel vale uno.

- **PALETTE_COLOR**

Los datos de los píxeles describen una imagen en color con un solo sample por píxel (imagen de un solo plano). El valor del píxel es usado como un índice para las tablas de paletas de colores roja, verde y azul (Red, Blue, Green Palette Color Lookup Tables (0028,1101-1103 y 1201-1203). Este valor solo puede ser usado cuando Samples por Píxel es igual a uno. Cuando la interpretación fotométrica es PALETTE_COLOR, las tablas de paletas de colores roja, verde y azul deben estar presentes.

- **RGB**

Los datos de los píxeles representan una imagen en color descrita por planos de imagen rojo, verde y azul. El sample de valor mínimo de cada plano de color representa la intensidad mínima de ese color. Este valor debe ser usado solo cuando Samples Por Píxel vale tres.

**Filas**

El valor del elemento de datos Filas (0028,0010) representa el número de filas de la imagen. Es decir, la altura en píxeles de la imagen.

**Columnas**

El valor del elemento de datos Columnas (0028,0011) representa el número de columnas de la imagen. Es decir, el ancho en píxeles de la imagen.

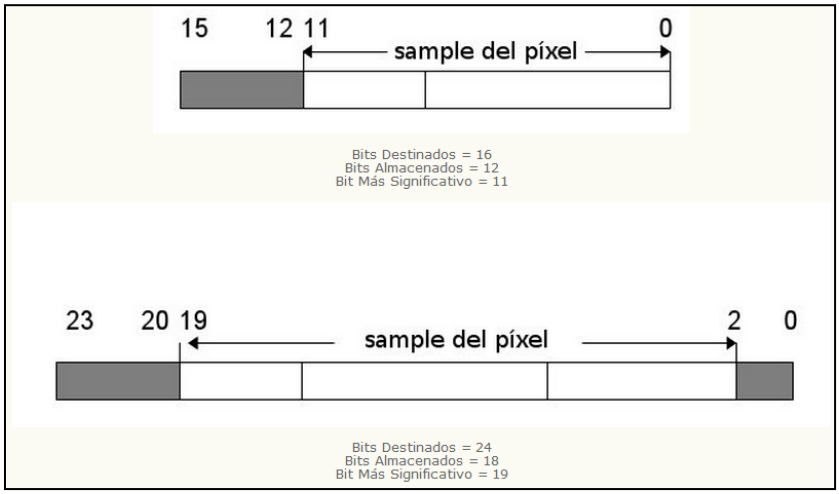
**Bits destinados, bits almacenados y bit más significativo**

El estándar permite utilizar cada valor del elemento de datos ***Pixel Data*** para almacenar información extra que no representa datos de los píxeles en sí. Por ejemplo, se pueden codificar los datos de los píxeles de forma que cada valor ocupe 16 bits, pero que sólo los últimos 12 de esos 16 bits representen el valor del píxel. *A aquellos bits que representan realmente el valor del píxel se les denomina* ***sample del píxel***. El resto de bits (los primeros 4) se pueden utilizar para temas específicos de la implementación o para planos de revestimiento (overlay planes).

Por lo tanto, es necesario especificar en la cabecera cuántos y cuáles de estos bits de los valores de los datos de los píxeles sirven para representar el *sample* y cuáles de ellos son para otra cosa.

- ***Bits Destinados*** es el tamaño de cada valor en bits de los datos de los píxeles, normalmente 8 ó 16.
- ***Bits Almacenados*** indica el número de bits de cada valor que se usan para representar el valor del sample.
- ***Bit Más Significativo*** indica la posición del bit más significativo de aquellos que se usan para representar el valor del sample.

Figura 12. Ejemplo de bits destinados, bits almacenados y bit más significativo.




**Representación de los Píxeles**

El valor del elemento de datos *Representación de los Píxeles* indica la representación de los samples de los píxeles.

Se pueden dar dos casos:

- 0000h = se representan como un entero sin signo.
- 0001h = se representan como un entero en complemento a dos.

**Datos de los Píxeles**

El orden de los píxeles para cada plano de imagen es de izquierda a derecha y de arriba a abajo. La forma de interpretar el formato de estos píxeles está definida por la interpretación fotométrica, y además por la sintaxis de transferencia en el caso de estar en algún formato encapsulado, como el JPEG. **[3]**

La siguiente tabla especifica un resumen de los atributos más comunes que describen los datos de pixel de la imagen.

Tabla 6. Módulo Pixeles de la Imagen.

| Nombre del atributo | Etiqueta | Descripción |
|---|---|---|
| Muestras por píxel | (0028,0002) | Numero de muestras (planos) en la imagen. Si la imagen es RGB este atributo tiene un valor de 3. |
| Interpretación fotométrica | (0028,0004) | Especifica como deben ser interpretados los datos de la imagen. Algunos de los valores en este elemento son:<br>MONOCHROME1<br>MONOCHROME2<br>PALETTE COLOR<br>RGB... etc. |
| Filas | (0028,0010) | Numero de filas en la imagen. |
| Columnas | (0028,0011) | Numero de columnas en la imagen |
| Bits Asignados | (0028,0100) | Numero de bits asignados a cada muestra (8, 16, etc). |
| Bits Almacenados | (0028,0101) | Numero de bits realmente utilizados por cada muestra. |
| Bit mas significativo | (0028,0102) | Bit más significativo en una muestra. |
| Representación de píxel | (0028,0103) | Representación de los datos en cada muestra. Puede ser:<br>0000H = entero sin signo.<br>0001H = complemento a dos |
| **Datos de Píxel** | **(7FE0,0010)** | **Imagen.** |
| Configuración Planar | (0028,0006) | Indica si la información de cada dato de píxel es enviada "color por plano" o "color por píxel". Estará presente si el dato muestras por píxel es mayor a 1. |
| Relación de aspecto | (0028,0034) | Relación entre el tamaño vertical y el horizontal de los píxeles de la imagen. |

La información referente a las características de la imagen y de la estructura de los pixeles se encuentra en los elementos pertenecientes al grupo 0028. En la tabla anterior se lista algunos de los atributos del modulo "***pixeles de la imagen***". Con esta información es posible interpretar correctamente los datos binarios contenidos en el elemento (7FE0, 0010). La mayor parte de estos elementos son datos tipo 1, por lo tanto es obligatorio que estén presentes en un set de datos correspondiente a una imagen. La lectura e interpretación de estos datos es de suma importancia en sistemas de procesamiento de imágenes ya que a partir de ellos es posible la correcta lectura y manipulación de las mismas.

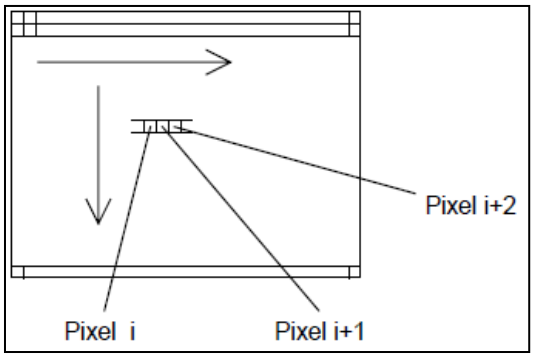
### 2.7.1. Codificación de los datos de pixeles

Los datos de pixeles de la imagen se almacenan dentro del campo *Valor* del elemento *Pixel Data* (7FE00, 0010). El orden en el cual el Pixel Data para una imagen plana es codificado es de izquierda a derecha y de arriba hacia abajo, una fila a la vez (figura 19).

Un pixel individual puede consistir de uno o más valores de Sample del Pixel (muestras de pixel) (es decir, imágenes en color o multiplanar). Cada valor del sample del pixel puede ser expresado ya sea como un entero complemento a dos binario o un entero sin signo binario, según lo especificado por el elemento *Pixel Representation* (0028, 0103). El número de bits en cada valor del Sample del Pixel se especifica por *Bits Stored* (0028, 0101). Para Sample del Pixel enteros complemento a dos el signo del bit es el bit más significativo del valor del Sample del Pixel.

Una celda de pixel es el contenedor para un valor del sample del pixel y opcionalmente bits adicionales. Estos bits adicionales pueden ser utilizados para superponer planos, o ubicar pixeles en ciertos límites (byte, Word, etc). El tamaño de las celdas de pixeles es especificado por *Bits Allocated* (0028, 0100) y es mayor que o igual a los *Bits Stored.* La ubicación de los valores del sample del pixel dentro de la celda de pixeles es especificada por *Hight Bit* (0028, 0102).

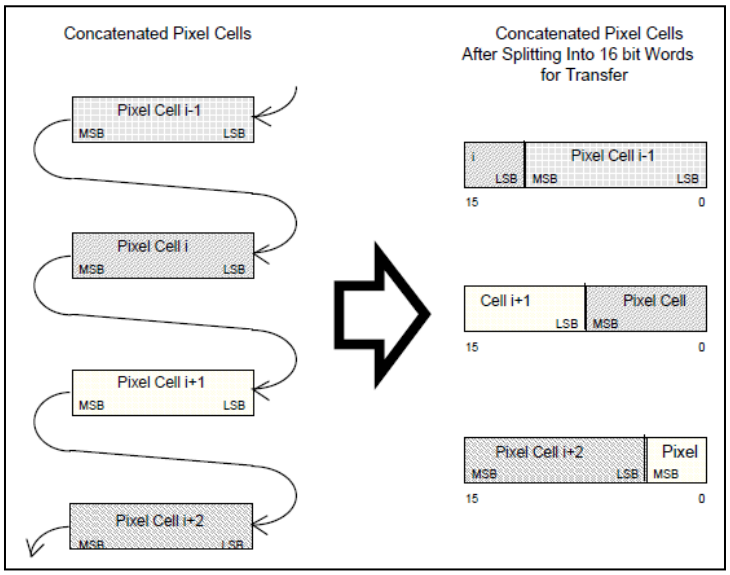
Figura 13. Imagen Plana.




El elemento Pixel Data, como el especificado por la *sintaxis de transferencia por defecto DICOM*, tiene un valor de representación de **OW** (Other Word String). El Pixel Data en DICOM 3.0, como lo fue en ACR-NEMA 2.0 está compactado (ver figura 20). Una manera de visualizar esta codificación compactada es imaginar que se codifican las celdas de píxeles como un flujo concatenado de bits desde el bit menos significativo de la primera celda de pixel hacia el bit más significativo de la última celda de pixel. Dentro de este flujo, el bit más significativo de cualquier celda de pixel es seguido por el bit menos significativo de la siguiente celda de pixel. *El Pixel Data puede entonces ser dividido en un flujo de palabras físicas de 16 bits, cada uno de los cuales está sujeto a las limitaciones de orden de bytes de la sintaxis de transferencia.*

*Todas las demás sintaxis de transferencia DICOM (no por defecto) hacen uso de la codificación de VR explicito*. Para estas sintaxis de trasferencia, todos los Pixel Data donde *Bits Allocated* es menor que o igual a 8 pueden ser codificadas con un VR explicito de OB. Al igual que en el caso OW, las celdas de píxeles se compactan juntas, pero en este caso, el Pixel Data se divide en un flujo de palabras físicos 8 bits.

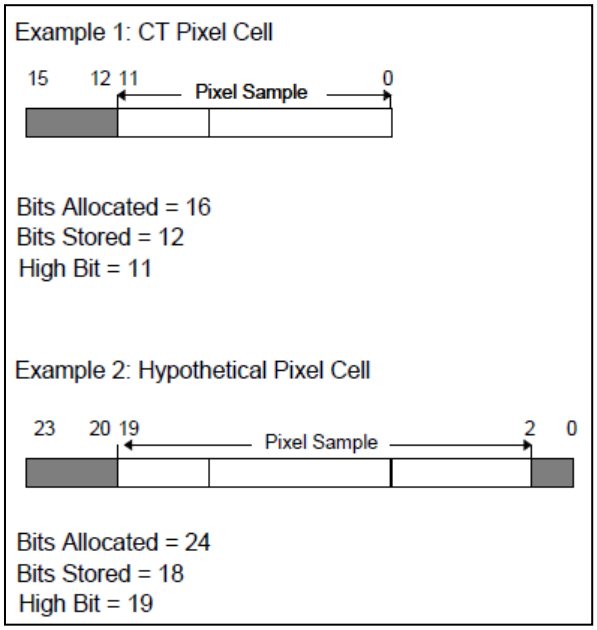
Figura 14. Codificación de Pixel Data con VR de OW.




**Nota**: para Pixel Data codificados con un VR explicito de OB, la codificación del Pixel Data no es afectada por el ordenamiento de bytes Big Endian o Little Endian.

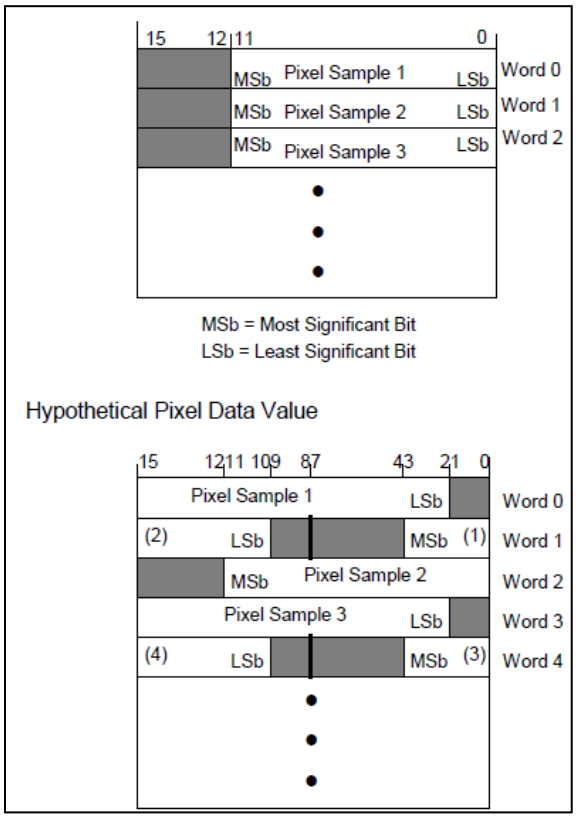
Ahora se realizan dos ejemplos de codificación de Pixel Data que utilizan el valor de representación **OW** a efectos de aclaración. El ejemplo 1 es un ejemplo valido para un IOD de imagen CT, mientras el ejemplo 2 será para un objeto de información hipotético (ver figura 21).

Figura 15. Ejemplo de celdas de pixeles.



La figura 22 muestra los Pixel Data construidos de los ejemplos de celdas de pixeles ya compactadas en un flujo de palabras de 16 bits.

Figura 16. Celdas de pixeles empaquetadas en Words de 16 bit VR = OW.




El ordenamiento de bytes es también especificado como parte de la sintaxis de transferencia negociada utilizado en el intercambio de un mensaje DICOM. Palabras de 16 bits son transmitidas a través de la red (***un byte a la vez***) primero el byte menos significativo en el caso de una sintaxis de transferencia Little Endian y el byte más significativo primero cuando utiliza una sintaxis de transferencia big Endian. En la figura 23 se puede observar como son transmitidos los bytes.

Como último par de ejemplos, para Pixel Data con la Representación de valores **OW** y los siguientes atributos: *8 bits destinados, 8 bits almacenados, y un bit alto de 7*; los flujos de bytes resultantes representados en la figura 24 son como se transmiten a través de una red y / o almacenados en los medios de comunicación. Para Píxel Data teniendo los mismos atributos, pero teniendo el valor de representación **OB**; los flujos de bytes resultantes no son afectados por el ordenamiento de bytes y se representan en la figura 25.

Figura 17. Flujos de Byte de los Pixel Data (VR=OW).

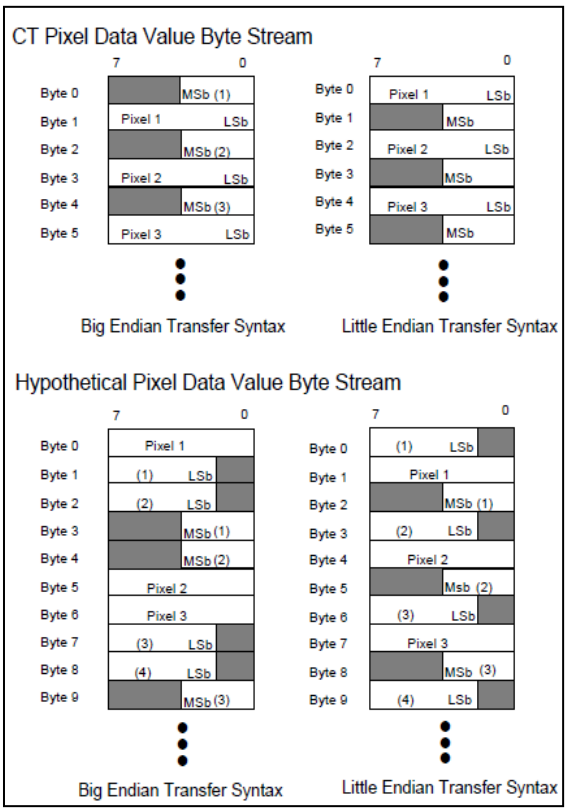



Figura 18. Flujos de Byte de los Pixel Data (VR=OW).

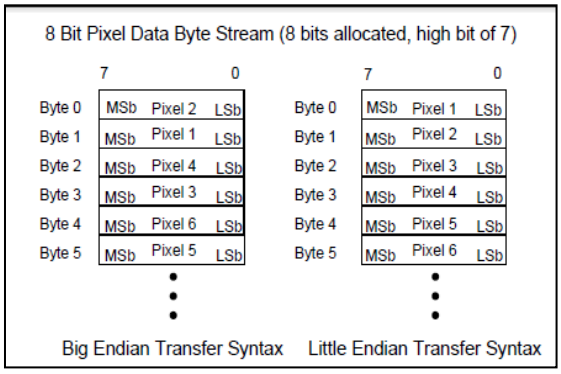
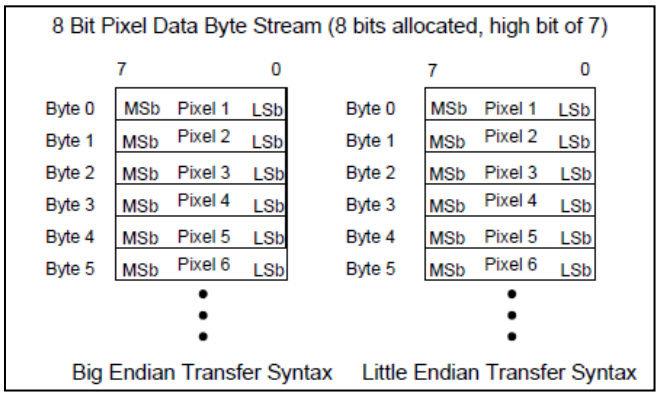


Figura 19. Flujos de Byte de los Pixel Data (VR=OB).



# Capítulo 3

# Sistema Operativo Android

## 3.1. Introducción

**Android** es un sistema operativo basado en el kernel de Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas, inicialmente desarrollado por Android, Inc. Google respaldó económicamente y más tarde compró esta empresa en 2005. Android fue presentado en 2007 junto la fundación del Open Handset Alliance: un consorcio de compañías de hardware, software y telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008. [4]

## 3.2. Arquitectura del sistema operativo Android

Los componentes principales del sistema operativo de Android:

- Aplicaciones: las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.

- Marco de trabajo de aplicaciones: los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

- Bibliotecas: Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android; algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.

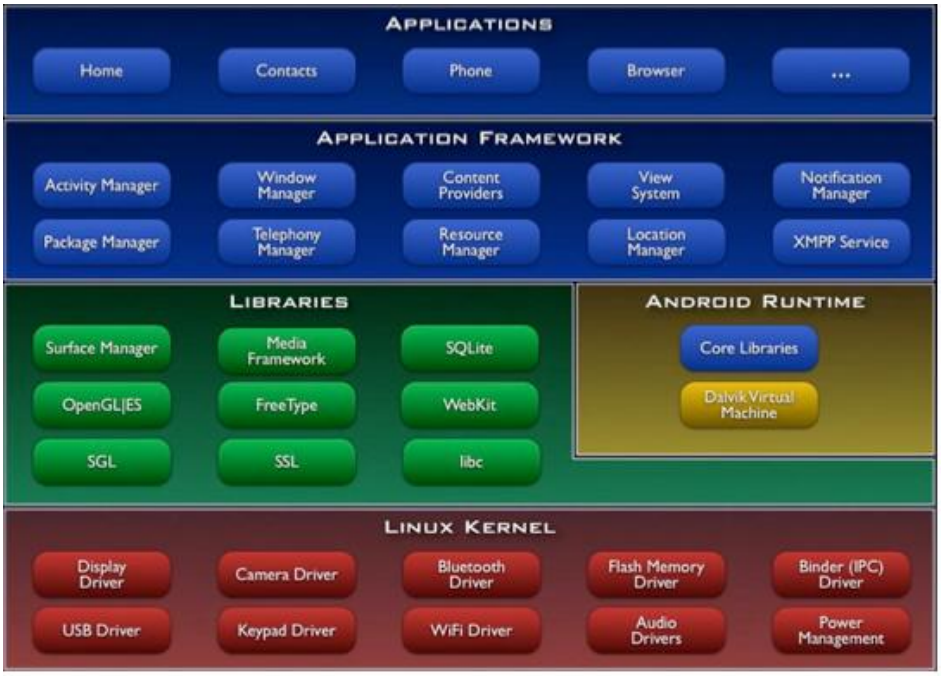
- Runtime de Android: Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en

registros y corre clases compiladas por el compilador de Java que han sido transformadas al formato.dex por la herramienta incluida "dx".

- Núcleo Linux: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

En la siguiente figura se observa el framework del sistema operativo Android

Figura 20. Arquitectura del SO Android.




Los dos últimos niveles de la arquitectura de Android están escritos enteramente en Java. El Framework de las aplicaciones representa fundamentalmente el conjunto de las herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o por terceras compañías o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo framework, representado por este nivel.

El último nivel del diseño arquitectónico de Android son las aplicaciones. Éste nivel incluye tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente. Todas estas aplicaciones utilizan servicios, las API y librerías de los niveles anteriores.

## 3.3. Componentes de aplicación

Los componentes de aplicación son los bloques de construcción esenciales de una aplicación Android. Cada componente es un punto diferente a través del cual el sistema puede entrar en su aplicación. No todos los componentes son puntos de entrada para el usuario y algunos dependen de unos y otros, pero cada uno existe como una entidad propia y desempeña un papel especifico – cada uno es una única pieza que ayuda a definir el comportamiento global de sus aplicaciones.

Hay cuatro tipos diferentes de componentes de aplicación. Cada tipo tiene un propósito diferente y un distinto ciclo de vida que define como el componente se crea y se destruye.

Estos son los cuatro tipos de componentes de aplicación:

- **Actividades**

Una actividad representa una pantalla única con una interfaz de usuario. Por ejemplo, una aplicación email debe tener una actividad que muestre una lista de nuevos emails, otra actividad para componer un email, y otra actividad para leer emails. Aunque las actividades trabajan juntas para formar una experiencia de usuario coherente en la aplicación email, cada una es independiente de la otra. Como tal, una aplicación diferente puede iniciar cualquiera de estas actividades (si la aplicación email lo permite). Por ejemplo, una aplicación de cámara puede iniciar la actividad en la aplicación email que compone un nuevo email, de forma que el usuario comporta una foto.

Una actividad es implementada como una subclase de **Activity** y puede obtener más información al respecto en la guía de desarrolladores de actividades (Activities developer guide).

- **Servicios**

Un servicio es un componente que se ejecuta en segundo plano para realizar operaciones de larga duración o para realizar trabajos para procesos remotos. Un servicio no proporciona una interfaz de usuario. Por ejemplo, un servicio puede reproducir música en segundo plano (fondo) mientras el usuario está en una aplicación diferente, o puede obtener datos mediante la red sin el bloqueo de la interacción con una actividad. Otra componente, tal como una actividad, puede iniciar el servicio y se deja correr o unirse a ella con el fin de interactuar con él.

Un servicio es implementado como una subclase de **Service** y puede obtener más información al respecto en la guía de desarrolladores de servicios (Services developer guide).

- **Proveedores de contenido**

Un proveedor de contenido gestiona un conjunto compartido de datos de aplicación. Puede almacenar los datos en el sistema de archivo, una base de datos SQLite, en la web, o cualquier otro lugar de almacenamiento persistente en la que su aplicación pueda acceder.

A través del proveedor de contenidos, otras aplicaciones pueden consultar o incluso modificar los datos (si el proveedor de contenidos lo permite). Por ejemplo, el sistema Android proporciona un proveedor de contenidos que gestiona la información de contacto del usuario. Como tal, cualquier aplicación con los permisos adecuados puede consultar parte de los proveedores de contenido (such as ContactsContract.Data) para leer y escribir información acerca de una persona particular.

Los proveedores de contenido también son útiles para lectura y escritura de de datos que es privada para su aplicación y no comparte. Por ejemplo la aplicación de muestra ***Note Pad*** utiliza un proveedor de contenido para guardar notas.

Un proveedor de contenidos es implementado como una subclase de ***Content Provider*** y debe implementar un conjunto estándar de APIs que permitan a otras aplicaciones realizar transacciones.

- **Receptores de mensajes de Distribución**

También llamados ***broadcast receiver*** o notificaciones, son los encargados de reaccionar ante los eventos ocurridos en el dispositivo, ya sean generados por el sistema o por una aplicación externa.
No tienen interfaz, pero pueden lanzar una *Actividad* por medio de un evento. La clase que defina estos componentes heredará de la clase *BroadCastReceiver*. Su ciclo de vida es muy corto, ya que solo están activos mientras se ejecuta el método onReceive (Context, Intent), que es equivalente al onCreate(Bundle) de otros componentes. El objeto Context nos pasa es estado actual, y el intent, nos permitirá lanzar el evento.
Muchas *broadcasts* se originan desde el sistema - por ejemplo, un broadcasts anunciando que la pantalla se ha apagado, que la batería está baja, o que una imagen fue capturada. Las aplicaciones también pueden iniciar los broadcasts, por ejemplo, dejar que otras aplicaciones sepan que algunos datos se han descargado al dispositivo y están disponibles para que los utilicen.
Aunque los *broadcast receiver* no muestran una interfaz de usuario, es posible crear una notificación de la barra de estado para alertar al usuario cuando ocurre un evento *broadcast.*

Cuando el sistema inicia un componente, este inicia el proceso para esa aplicación (si no está ya en ejecución) e instancia las clases necesarias para el componente. Por ejemplo, si su aplicación inicia una actividad en la aplicación cámara que captura una foto, esa actividad se ejecuta en el proceso que pertenece a la aplicación de la cámara, no en el proceso de su aplicación. Por lo tanto, a diferencia de las aplicaciones en la mayoría de los otros sistemas, las aplicaciones de Android no tienen un único punto de entrada (por ejemplo, no hay una función **main ()**).

Debido a que el sistema ejecuta cada aplicación en un proceso separado con permisos de archivos que restringen el acceso a otras aplicaciones, su aplicación no puede activar directamente un componente de otra aplicación. El sistema Android, sin embargo, puede. Por lo tanto, para activar un componente de otra aplicación, se debe entregar un mensaje al sistema que especifica su ***intención*** de iniciar un componente particular. El sistema entonces activa el componente por usted.

### 3.4. El archivo Manifest

Antes de que el sistema Android inicialice un componente de aplicación, el sistema debe saber que el componente existe leyendo el archivo ***AndroidManifest.xml*** de la aplicación. Su aplicación debe declarar todos sus componentes en este archivo, el cual debe estar en el directorio raíz del proyecto de la aplicación.

El manifiesto hace una serie de cosas, además de declarar los componentes de la aplicación, tales como:

- Identificar cualquier permiso de usuario que la aplicación requiera, tales como acceso a internet o acceso de lectura a los contactos de usuario.
- Declarar el nivel de API mínimo requerido por la aplicación, basado en que API utiliza la aplicación.
- Declarar las características de hardware y software utilizadas o requeridas por la aplicación, tal como cámara, servicios bluetooth o una pantalla multitouch.
- Y mas.

La tarea principal del Manifiest es informar al sistema acerca de los componentes de la aplicación. Por ejemplo, un archivo de manifiesto puede declarar una actividad de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
   <application android:icon="@drawable/app_icon.png" ... >
      <activity android:name="com.example.project.ExampleActivity"
            android:label="@string/example_label" ... >
      </activity>
      ...
   </application>
</manifest>
```

Se debe declarar todos los componentes de la aplicación de la siguiente forma:

- <**activity**> elementos para actividades
- <**service**> elementos para servicios
- <**receiver**> elementos para broadcast receivers
- <**provider**> elementos para proveedor de contenidos

En el elemento **<activity>,** el atributo **android:name** especifica el nombre de la clase completo de la subclase Activity y el atributo **android:label** especifica una cadena a utilizar como una etiqueta visible al usuario para la actividad.

Actividades, servicios, y proveedores de contenido que incluya en su código fuente, pero no se declaran en el manifiesto no son visibles para el sistema y, en consecuencia, no se puede ejecutar. Sin embargo, los *broadcast receiver* pueden ser declarados, ya sea en el manifiesto o creados de forma dinámica en el código (como objetos ***BroadcastReceiver***) y registrados en el sistema llamando a **registerReceiver ().**

### 3.5. Android NDK

El NDK es un conjunto de herramientas que permite implementar partes de su aplicación utilizando lenguajes de código nativo como C y C + +. Para ciertos tipos de aplicaciones, esto puede ser útil porque permite volver a utilizar bibliotecas de código existentes escritas en estos lenguajes, pero para la mayoría de las aplicaciones no es necesario el NDK Android.

Como desarrollador, necesita equilibrar sus ventajas frente a las desventajas en el uso del NDK. En particular, el uso de código nativo en Android en general no se traduce en una mejora de rendimiento notable, pero siempre aumentará su complejidad a la aplicación. Por lo general, sólo se debe utilizar el NDK si es esencial para la aplicación, no porque simplemente prefiere programar en C / C + +.

Los típicos casos para el uso del NDK son, operaciones intensivas de CPU que no necesitan mucha memoria, tales como procesamiento de señales, simulaciones físicas, y así sucesivamente. Al analizar si debe o no desarrollar en código nativo, piense en sus necesidades y ver si las APIs del framework de Android proporcionan la funcionalidad que necesita.
El NDK contiene las APIs, documentación y aplicaciones ejemplo que facilitan el escribir código nativo. Específicamente:

- Un conjunto de herramientas y archivos construidos utilizados para generar librerías de código nativo de fuentes C y C++.
- Una manera de integrar las correspondientes bibliotecas nativas en un archivo de paquete de aplicación (. Apk) que se puede implementar en los dispositivos Android.
- Un conjunto de cabeceras de sistema nativo y bibliotecas que serán soportadas en todas las futuras versiones de la plataforma Android, a partir de Android 1.5. Las aplicaciones que utilizan las actividades nativas se deben ejecutar en Android 2.3 o superior.
- Documentación, ejemplos y tutoriales.

# Capítulo 4
# Bibliotecas para procesamiento de imágenes

## 4.1. Introducción

En la actualidad existen diversas herramientas con funciones específicas para la manipulación de archivos DICOM, entre ellas podemos destacar según su función las bibliotecas de visualización, de comunicación y aquellas que integran estas dos funciones. Las herramientas de visualización son bibliotecas que permiten mostrar al usuario el contenido de los archivos DICOM, tanto los metadatos como la imagen que viene contenida en éste, estas bibliotecas se implementan en las estaciones de trabajo en la cual la interacción con el usuario se hace de una manera directa. También existen bibliotecas que están diseñadas con base en el protocolo de comunicaciones DICOM, y se especializan en el transporte de acuerdo a los atributos que especifique el cliente, estas bibliotecas cumplen con la función en el PACS de interconectar las diferentes partes que lo componen (servidor, estaciones de trabajo). En la actualidad existen herramientas que permiten manejar el módulo de comunicación y de visualización de una manera integrada.

## 4.2. Bibliotecas de imágenes médicas

- **Insight Segmentation and Registration Toolkit (ITK)**.

Es una biblioteca de código libre, utilizada principalmente para la ejecución del registro y segmentación en las imágenes. ITK está implementado en C++, y usa la herramienta CMAKE para manejar el proceso de compilación. Además presenta un proceso envolvente que permite interpretar los lenguajes de programación como Tcl, Java, Python.

ITK es un código abierto, sistema multiplataforma que proporciona a los desarrolladores un amplio conjunto de herramientas de software para el análisis de imágenes. Desarrollado a través de metodologías de programación extrema, ITK emplea algoritmos de vanguardia para el registro y la segmentación de datos multidimensionales. Las metas para ITK incluyen:

- La creación de un repositorio de algoritmos fundamentales.
- El desarrollo de una plataforma para el desarrollo de productos avanzados.
- El establecimiento de una base para futuras investigaciones.

ITK implementa una biblioteca llamada GDCM. Esta biblioteca de código abierto fue desarrollada por CREATIS Tim, y se distribuye con licencia LPGL. La librería GDCM sólo implementa la parte

5 del estándar DICOM, lo cual hace que ITK sea limitado y no acepte ningún formato comprimido de DICOM; entre sus principales aplicaciones están las de leer y escribir imágenes DICOM sin compresión, de manera individual y por series. [5]

- **Visualization Toolkit (VTK)**

El VTK es un sistema de software disponible gratuitamente, de código abierto para gráficos por ordenador en 3D, procesamiento de imágenes y visualización. VTK consiste en una biblioteca de clases C + + y varias capas de interfaz interpretadas incluyendo Tcl / Tk, Java y Python.

VTK soporta una amplia variedad de algoritmos de visualización incluyendo: escalar, vectorial, tensor, textura, y métodos volumétricos; y técnicas avanzadas de modelado como: modelado implícito, reducción de polígonos, el suavizado de malla, corte, contorneado, y la triangulación de Delaunay.

En las últimas versiones se han incluido clases que permiten visualizar Dicom, pero no soporta formatos comprimidos. [6]

- **Java Dicom Toolkit (JDT)**

Es una biblioteca de desarrollo para archivos DICOM que está implementada totalmente en JAVA. JDT funciona sobre JDK 1.1.X y JAVA 2. JDT no es un producto gratuito, pero cuenta con la ventaja de abrir cualquier tipo de formato DICOM.

- **DCMTK**

Es una colección de bibliotecas y aplicaciones implementando gran parte del estándar Dicom. Este incluye software para examinar, construir y convertir archivos de imagen Dicom, enviar y recibir imágenes sobre una conexión de red, etc. DCMTK está escrito en una mezcla de ANSI C y C++. Se presenta en el código fuente completo y está disponible como software de "código abierto".

El software DCMTK puede ser compilado bajo Windows y un amplio rango de sistemas operativos Unix incluyendo Linux, Solaris, HP-UX, IRIX, FreeBSD, OpenBSD y MacOS. Todos los scripts de configuración necesarios y makefiles del proyecto son suministrados.[7]

- **PixelMed Java DICOM Toolkit**

Es un conjunto de herramientas DICOM independiente que implementa código para la lectura y la creación de datos de DICOM, red DICOM y soporte de archivos, una base de datos de objetos DICOM, soporte para la visualización de directorios, imágenes, informes y espectros, y la validación de objetos DICOM. [8]

- **OpenCV**

OpenCV es liberado bajo una licencia BSD y por lo tanto es libre para uso académico y comercial. Cuenta con interfaces de C + +, C, Python y Java y es compatible con Windows, Linux, Mac OS, iOS y Android. OpenCV fue diseñado para la eficiencia computacional y con un fuerte enfoque en las aplicaciones en tiempo real. Posee más de 500 funciones, que incluye funciones en lenguaje nativo para aplicaciones de visión artificial. [9]

OpenCV es la biblioteca idónea para el tratamiento de imágenes, ya que se diseñó para el procesamiento de imágenes, cada función y estructura de los datos se diseñó con el codificador de procesamiento de imágenes en mente, lo que supone una mayor rapidez para ejecutar acciones de este tipo. El hecho de que esté desarrollada en C/C++ supone una mayor rapidez en la ejecución de las operaciones, que comparado con otros lenguajes y herramientas con las que poder realizar tales acciones supone un mejor rendimiento. Todo esto y el hecho del gran desarrollo de esta librería, la han convertido en una librería fundamental para el procesamiento de imágenes. [10]

- **Imebra C++ DICOM SDK**

Es una biblioteca Dicom C++ de código abierto multiplataforma con envoltorios Java que pueden ser utilizados en Windows, Linux, Android, IOS y SO X.

- La biblioteca puede manejar DICOM 3, NEMA y archivos jpeg
- La biblioteca también soporta el estándar UNICODE

Algunas características

- Multiplataforma: Linux, Windows, IOS, Android, OS X
- El código fuente puede ser compilado con gcc, clang, Visual Studio
- El código fuente y la documentación están disponibles bajo la licencia GPL [11]

# Capítulo 5

# Metodología

## 5.1. Introducción

En el presente proyecto la idea es desarrollar una aplicación médica implementando partes del estándar Dicom para visualizar imágenes médicas en dispositivos móviles, en especial para el sistema operativo Android. En primer lugar, se utilizó la librería DCM4CHE debido a la madurez del proyecto, el número de aplicaciones que lo utilizan en el ámbito clínico [12], la continua expansión y actualización del toolkit y el importante grupo de desarrolladores que dan soporte al proyecto y brindan su continuidad en el tiempo.

## 5.2. DCM4CHE

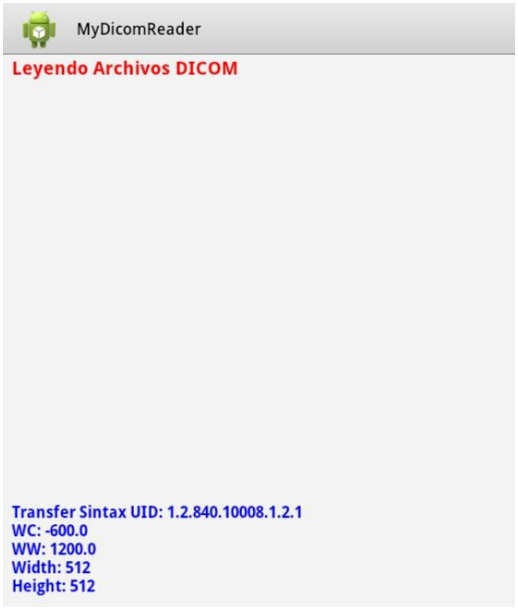
Dcm4che es una colección de aplicaciones de código abierto y utilidades para las empresas del sector de la salud. Estas aplicaciones se han desarrollado en el lenguaje de programación Java por motivos de rendimiento y portabilidad. En el núcleo del proyecto dcm4che hay una implementación robusta de la norma DICOM. El dcm4che-1.x es un conjunto de herramientas DICOM que se utiliza en muchas aplicaciones de producción en todo el mundo, mientras que la corriente versión 2.x versión ha sido rediseñada para un alto rendimiento y flexibilidad. [13]

Sin embargo, al trabajar con la biblioteca en un ambiente móvil se presentaron problemas ya que la biblioteca utiliza APIs Java que el sistema operativo Android no implementa, razón por la cual se generaban errores, como por ejemplo no encontrar clases de determinados métodos. Se optó por hacer una codificación básica que permitiera leer la metadata de un archivo Dicom, pero sin visualizar la imagen. A continuación se presenta un trozo de código que permite dicha lectura:

```
try {

        File myDicom = new File("/mnt/sdcard/download/sample.dcm");
        din = new DicomInputStream(myDicom);
        dcmObj = din.readDicomObject();
        obteniendoData();


}catch(IOException e) {
        e.printStackTrace();
}
```

```
private void obteniendoData() {

                sintaxis = dcmObj.getString(Tag.TransferSyntaxUID);

                //I get Window Center
                float wc = dcmObj.getFloat(Integer.parseInt("00281050", 16));

                //I get Window Width
                float ww = dcmObj.getFloat(Integer.parseInt("00281051", 16));

                //I get Width
                int columnas = dcmObj.getInt(Tag.Columns);

                //I get Height
                int filas = dcmObj.getInt(Tag.Rows);

tv2.setText("Transfer Sintax UID: "+sintaxis+"\nWC: " +wc+ "\nWW: "+ww+"\nWidth: "
                                        +columnas+"\nHeight: "+filas);          }
}
```

Figura 21. Captura de pantalla con la biblioteca DCM4CHE.




Se trató de muchas maneras utilizar esta biblioteca, o trozos de código de sus aplicaciones Java que permitieran visualizar una imagen Dicom en el SO Android, razón por la cual se descartó al cabo del tiempo sin mayores éxitos.

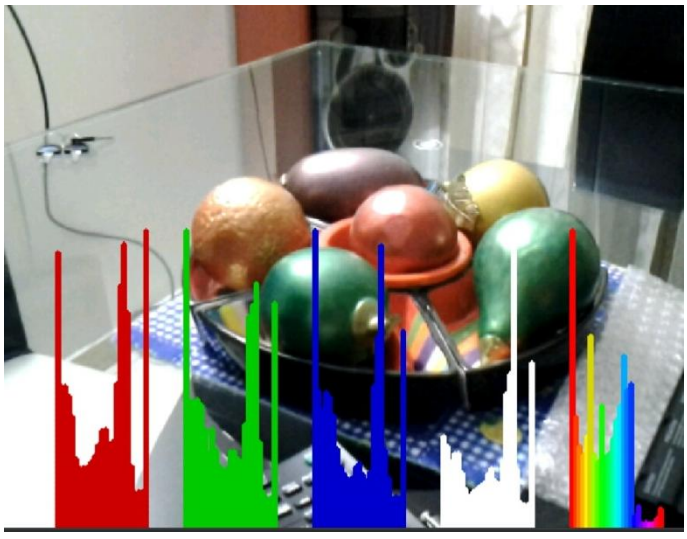
## 5.3. OpenCV

A esta biblioteca se llegó por el trabajo "RECONOCIMIENTO DE IMÁGENES UTILIZANDO REDES NEURONALES ARTIFICIALES", realizado por Pedro Pablo García García. Este trabajo describe el proceso de extracción de patrones característicos de imágenes, mediante la ayuda de Redes Neuronales Artificiales. La información de la Red Neuronal junto con datos adicionales de las imágenes, serán almacenados en una base de datos y consumidos por un servicio web. Un teléfono móvil con sistema operativo Android consumirá la información almacenada en el servicio web. Posteriormente al realizar una captura de imagen con la cámara del teléfono, este procesará la imagen y junto con los datos consumidos por el servicio web será capaz identificar de qué imagen se trata. Para el tratamiento de las imágenes se utilizarán librerías OpenCV, tanto en el servidor como en el teléfono móvil.

Para este proyecto se necesitó implementar algoritmos de manipulación de imágenes, por lo que es conveniente buscar librerías adecuadas que contengan la mayor parte de las funciones que se van a utilizar, además por las características del proyecto en concreto, estas librerías deberían poder ser utilizadas desde diferentes entornos de programación y diferentes lenguajes. Por otro lado y puesto que es un proyecto de investigación es conveniente que sean de uso libre, altamente eficientes y de fácil utilización. Todas estas características las reúnen las librerías **OpenCV** ya que son multiplataforma, existiendo versiones para Linux, Mac y Windows. Estas librerías incorporan más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión. Por otro lado están distribuidas bajo licencia BSD (Berkeley Software Distribution) que permite ser utilizadas libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.[14]

Con respecto a esta librería se está trabajando y analizando el código fuente de las aplicaciones ejemplo que trae consigo y conociendo las clases básicas para el manejo de imágenes. A continuación se muestra un pantallazo de una de sus aplicaciones:

Figura 22. Captura de pantalla para OpenCV.



**C++.** Lenguaje de programación que surge ante la necesidad de extender el exitoso lenguaje de programación C, introduciendo la programación basada en objetos. Desde el punto de vista de lenguajes orientado a objetos es un lenguaje híbrido, al que además se le añadieron facilidades de programación genérica, que sumada a los paradigmas ya presentes (programación estructurada y orientada a objetos) lo convierten en un lenguaje multiparadigma.

Para poder utilizar la gran cantidad de funciones disponibles en la biblioteca, se debe importar compilada al proyecto y crear una clase java auxiliar en la que se cargarán las funciones de la librería.

```
public class OpenCV {
/**
* Libreria para cargar los metodos nativos.
*/
	static {
		System.loadLibrary("opencv");
	}

/*Métodos que deseamos importar*/
public native static byte[] getSourceImage();
}
```

### 5.4. Imebra C++ DICOM SDK

Imebra para Android es una colección de wrappers JNI para la biblioteca Dicom Imebra. El paquete también contiene los binarios pre-construidos para diferentes arquitecturas.

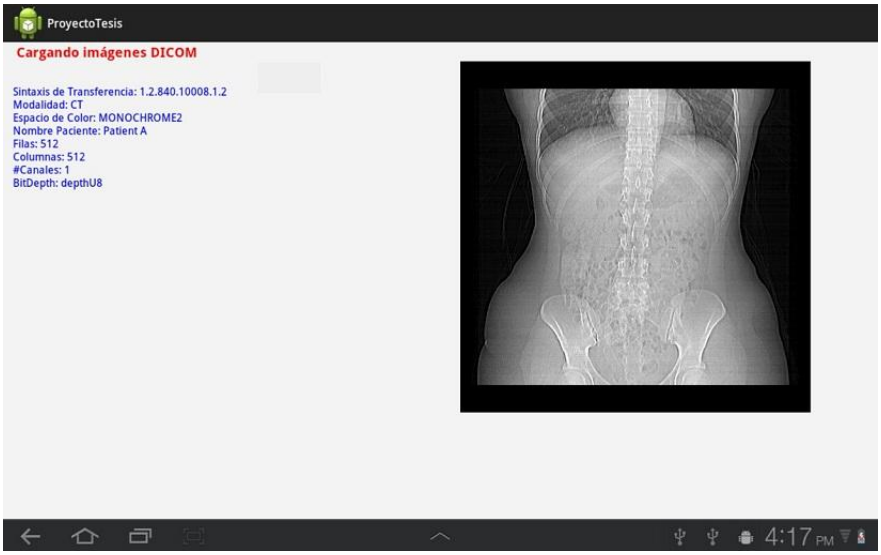
```
public class MainActivity extends ActionBarActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    System.loadLibrary("imebra_lib");
  }


public void onStart(){
    super.onStart();

    Stream stream = new Stream();
    stream.openFileRead("/mnt/sdcard/download/dicom.dcm");

    DataSet dataSet = CodecFactory.load(new StreamReader(stream), 256);
    Image image = dataSet.getImage(0);
```

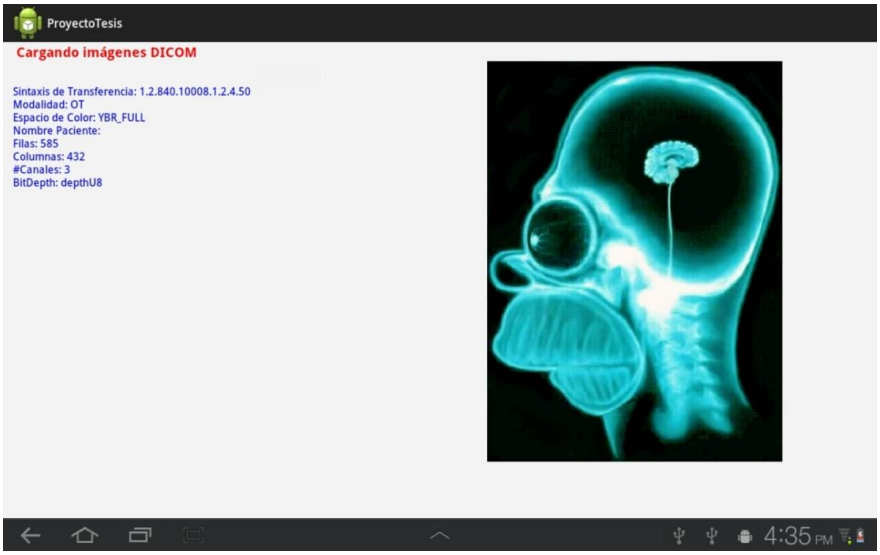
Los trozos de código anteriores permiten cargar la biblioteca Imebra y luego abrir un archivo Dicom y visualizarlo. Esta biblioteca es la utilizada para llevar a cabo la implementación de la aplicación y como entorno de desarrollo se utiliza el IDE Android Studio ya que es el que utilizan en el ejemplo mostrado en la página principal.

### 5.4.1 Implementando con Imebra C++ DICOM SDK

En primer lugar se empieza conociendo un poco las clases más importante de esta biblioteca para luego con el código base mostrado en la página principal empezar todo el proceso que llevará a visualizar dichas imágenes en el dispositivo móvil.
En las siguientes figuras se observa una captura de pantalla de algunos datos de un archivo Dicom y la respectiva imagen asociada. Se puede observar que la aplicación permite cargar imágenes con diferentes sintaxis de transferencia.

Figura 23. Capturas de pantalla de imagen DICOM.



Se presentaron problemas cargando imágenes con sintaxis de transferencia **1.2.840.10008.1.2.4.91** que se refiere a la sintaxis ***JPEG 2000 Image Compression***. La aplicación lanzaba la siguiente excepción:

java.lang.IllegalArgumentException: Unknown Transfer Syntax

Las pruebas se hicieron con imágenes Dicom con las siguientes sintaxis de transferencia:

| | |
|---|---|
| 1.2.840.10008.1.2 | Implicit VR, Little Endian |
| 1.2.840.10008.1.2.1 | Explicit VR, Little Endian |
| 1.2.840.10008.1.2.5 | Run Lenght Encoding, Lossless |
| 1.2.840.10008.1.2.4.50 | Baseline(1) |
| 1.2.840.10008.1.2.4.70 | Lossless JPEG Image Compress |

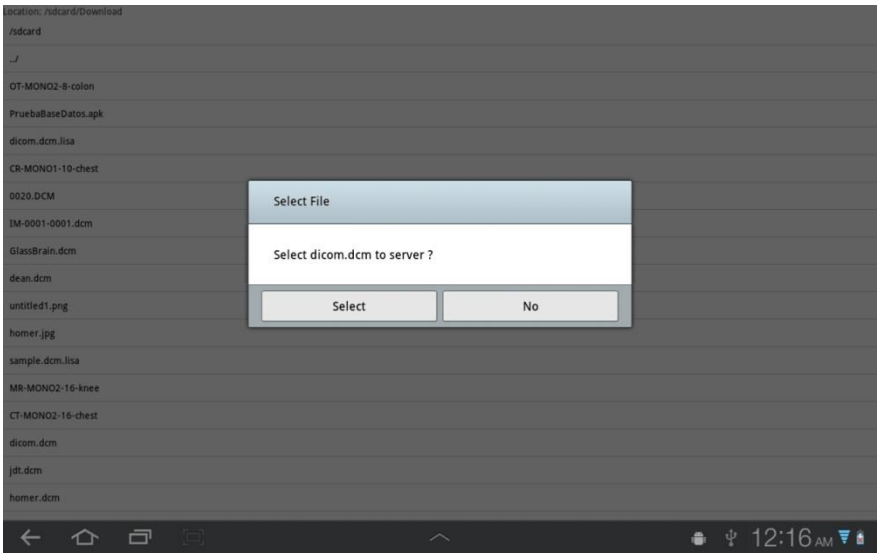
El proyecto se basa en cuatro módulos principales que permitió una mejor forma de trabajar y organizar todo el proceso del proyecto:

- Explorador de archivos.
- Conversor de imagen Dicom a Bitmap.
- Manipulación de la imagen Bitmap.
- Almacenamiento en base de datos.

**Explorador de archivos**

La idea principal de este modulo es permitir navegar hacia la tarjeta SD en la cual se tienen las imágenes Dicom (extensión .dcm) y de esta forma poder seleccionar la imagen deseada. En la figura 24 se muestra una captura de pantalla en ejecución del modulo.

Figura 24. Capturas de pantalla módulo Explorador de Archivos.



**Conversor de imagen Dicom a Bitmap**

La función principal de este modulo es convertir el archivo .dcm a bitmap. Esto se logra gracias a la clase DrawBitmap de la biblioteca Imebra. Es importante mencionar que en este módulo se hace un escalado de la imagen, ya que ésta es grande y puede provocar una excepción de tipo de asignación de memoria. En la figura 25 se observa la captura de pantalla al cargar el bitmap.

Figura 25. Capturas de pantalla módulo Conversor a Bitmap.


**Manipulación de la imagen Bitmap**.

En este módulo se implementa la función zoom de las imágenes después de la conversión y se le agrega la funcionalidad de poder manipular el brillo y el contraste de las imágenes. En la figura 26 se observa la respectiva captura de pantalla.

Figura 26. Capturas de pantalla módulo Manipulación Bitmap.


**Almacenamiento en base de datos**

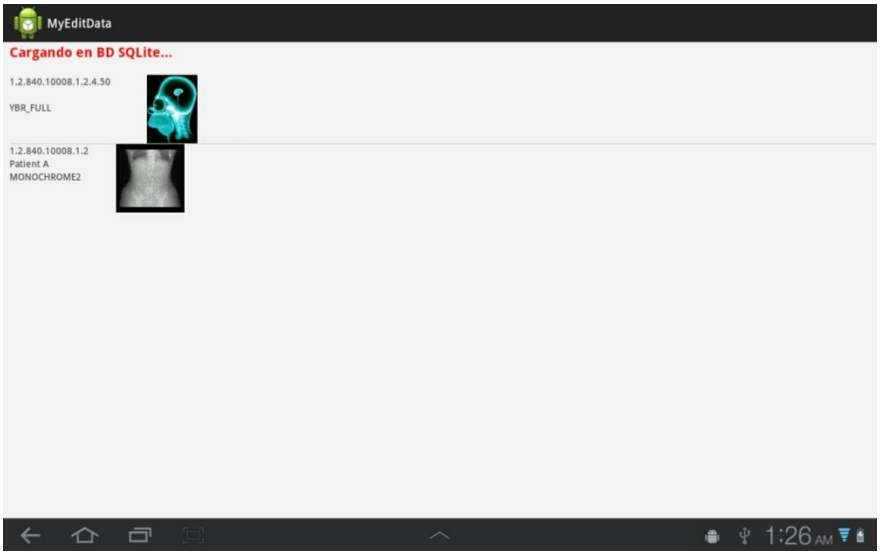
En este modulo se busca guardar en una base de datos parte de los atributos de la imagen y la imagen en sí. La plataforma Android proporciona dos herramientas principales para el almacenamiento y consulta de datos estructurados:

- Bases de Datos SQLite
- Content Providers

SQLite es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y por supuesto ser de código libre.

Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, y entre ellas una completa API para llevar a cabo de manera sencilla todas las tareas necesarias. En la figura 27 se observa una captura de pantalla de la carga de las imágenes en la base de datos.

Figura 27. Capturas de pantalla módulo Almacenamiento en BD.




### 5.4.2 Resultados

La aplicación desarrollada en este proyecto permite la visualización de imágenes médicas que se basan en el estándar Dicom. La aplicación se probó con algunas sintaxis de transferencia, entre las cuales se encuentran las nativas y algunas en el formato JPEG. Al cargar una imagen con la sintaxis de transferencia **1.2.840.10008.1.2.4.91** la aplicación lanza siguiente excepción: java.lang.IllegalArgumentException: Unknown Transfer Syntax

En la aplicación la imagen puede ser manipulada de tal forma que se le puede hacer zoom, movimiento y variar brillo y contraste. Se presentaron problemas al querer medir longitud entre dos puntos en la imagen ya que no se puede hacer en la misma actividad en la cual se hace el manejo del zoom que era lo ideal que se pretendía en el proyecto, razón por lo cual la imagen se carga en otra actividad y en ésta se hace la respectiva medición.

La actividad encargada de la medición de longitud en la imagen, presenta el inconveniente de que al cargar la imagen y no poderle hacer zoom resulta incomodo en algunas imágenes hacer la medición.

La aplicación permite cargar gran parte de la metadata asociada a la imagen. Se seleccionaron los atributos que se consideraron más importantes.

En la aplicación las imágenes se cargan en una base de datos que viene implícita con el sistema operativo Android. La idea es simplemente organizarlas para de esta forma visualizar las que se han manipulado. Esta actividad permite eliminar las imágenes que consideremos.

# Capítulo 6

# Conclusiones

El inicio en el entendimiento del estándar DICOM resulta al principio en una tarea con cierto grado de dificultad ya que gran parte de su contenido es complejo y con mucho concepto técnico que involucra muchos campos; sin embargo la implementación del estándar en el campo medico ha llevado a la creación de nuevas aplicaciones que han facilitado y mejorado los servicios básicos y complejos que se presentan en el diario vivir de la comunidad médica y también ha permitido a la comunidad científica mejorar y crear muchos herramientas que llevan a evaluar de una manera más precisa a los pacientes.

Las bibliotecas de desarrollo en base al estándar, presentan una ventaja para el desarrollador de aplicaciones DICOM, ya que son implementaciones robustas, generalmente de acceso libre, disponibles para diferentes lenguajes de programación, con manejo de valores y codificaciones que en muchos casos es una tarea compleja y molesta. El acceso a este tipo de librerías brinda las herramientas necesarias para que cualquier persona interesada genere sus propias soluciones DICOM, lo que propicia la aparición de múltiples implementaciones en la atención a la salud, y de esta forma contribuir a un mejor servicio a la comunidad.

Las aplicaciones móviles en salud se han incrementado debido en gran parte a los centros de investigación en salud y a los gobiernos interesados en ofrecer de una manera más amplia servicios de salud básicos de tal forma que sean beneficiadas una mayor población y en especial las más vulnerables. Esto se logra de una manera conjunta donde se involucra gobierno, instituciones de salud y desarrolladores de tal forma que se mire hacia un mismo horizonte y se comprometan llegar a una población más distante.

Las aplicaciones móviles en salud también ha permitido facilitarles en gran parte el trabajo a enfermeros, médicos y especialistas ya que ofrecen una cantidad de herramientas lo que conlleva a unos diagnósticos más precisos y por ende una mejor atención a los pacientes.

La aplicación desarrollada en el proyecto no pretende ser una herramienta que permita a personal especializado utilizarla como soporte. En el mercado se encuentran gran cantidad de aplicaciones que sirven como herramientas para visualizar imágenes basadas en el estándar Dicom pero en la gran mayoría se advierte que no están aprobadas por las respectivas autoridades de salud. Esto conlleva a realizar un trabajo de investigación más profundo y de esta forma garantizar un excelente producto.

# REFERENCIAS


**[1] DICOM básico segunda edición. Herman Oosterwijk. http://www.otechimg.com/product.cfm?prd=DICOM%20Basics%20Book%20Spanish**

**[2] diseño e implementación de un sistema telamatico para equipos médicos usando el protocolo de comunicaciones dicom**

[3] http://www.revistaesalud.com/index.php/revistaesalud/article/view/261/607

[4] http://en.wikipedia.org/wiki/Android_(operating_system)

[5] http://www.itk.org/

[6] http://www.vtk.org/

[7] http://dicom.offis.de/dcmtk.php.en

[8] http://www.pixelmed.com/

[9] http://opencv.org/

[10] TRATAMIENTO DE IMÁGENES, EXTRACCIÓN DE CARACTERÍSTICAS Y "*MATCHING*" EN PLATAFORMAS ANDROID

**[11] http://imebra.com/**

**[12].** http://www.41jaiio.org.ar/sites/default/files/5_CAIS_2012.pdf

**[13]**Carlos Perez Castillo. IMBIS – Sistema de Información para Biomarcadores de Imagen. Universidad Politecnica de Valencia

**[14]** Pedro Pablo Garcia Garcia. Reconocimiento de Imágenes utilizando Redes Neuronales Artificiales. Universidad Complutense de Madrid