Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Valencia

Proyecto Final de Carrera:

Diseño, desarrollo e implementación de una plataforma modular y multifabricante para la visualización y procesado de imágenes médicas funcionales

Autor:

Adrián Colomer Granero

Directores:

Valery Naranjo Ornedo
Departamento de Comunicaciones
Universidad Politécnica de Valencia

Ángel Alberich Bayarri
Ingeniería Biomédica
Grupo Hospitalario Quirón

Julio 2013

“Nuestra recompensa se encuentra en el esfuerzo y no en el resultado. Un esfuerzo total es una victoria completa”

Mahatma Gandhi

Agradecimientos

A Valery, por haber confiado en mí, por toda la ayuda y consejos que me ha ofrecido. Te agradezco enormemente la oportunidad que me brindas, como sabes, la afronto con muchísimas ganas e ilusión.

A Ángel, por su apoyo y total dedicación en este trabajo. Sin importar la hora y a través de cualquier medio, siempre he recibido ayuda instantánea que me ha permitido avanzar en el proyecto.

A Pablo y Fernando, siempre dispuestos a ayudarme, sin perder la paciencia conmigo.

Este proyecto pone punto final a seis años de gran esfuerzo, en los que sin el apoyo de los míos hubiera sido imposible llegar a escribir estas líneas.

Por todo el interés que han mostrado durante mi trayectoria académica y los ánimos que me han transmitido, quiero dar las gracias a mis abuelos, Mª Carmen y Francisco.

Agradezco de todo corazón a mi madre Jacqueline y a mi hermano Fran todo lo que han hecho por mí. Vuestro cariño me ha ayudado a afrontar cada día. Sin vosotros esto no hubiera sido posible.

Agradecimiento especial merece Patri, mi novia, que ha mostrado un gran interés en que este trabajo saliese bien, leyéndoselo y corrigiendo detalles del mismo. La persona con la que he compartido todos mis problemas diarios y la que me ha ayudado a hacerlos más llevaderos. Gracias por tu ayuda y por hacerme tan feliz.

A Luis, mi padre, gracias por los todos los consejos que me has dado, de una forma u otra siempre me han ayudado.

No puedo olvidarme de aquellos que hace seis años comenzaron siendo compañeros de clase y ahora son grandes amigos. Gracias a vosotros todas esas horas de esfuerzo y estudio han sido mucho más amenas.

Resumen del Proyecto

Debido a la digitalización de la imagen ya es posible tratarla y procesarla mediante métodos de ingeniería, con el objetivo de mejorar su calidad y obtener información que no es visible a simple vista. Esta característica ha sido de vital importancia en múltiples campos, pero sin duda, uno de los más importantes es el de la medicina.

En las unidades de radiología de cualquier hospital, se hace imprescindible un software capaz de visualizar las imágenes que se obtienen mediante cualquiera de las técnicas de adquisición existentes.

El presente proyecto final de carrera, es fruto de la colaboración entre el departamento de comunicaciones de la Escuela Técnica Superior de Telecomunicaciones de la Universidad Politécnica de Valencia con la Unidad de Ingeniería Biomédica del Grupo Hospitalario Quirón y su hospital de Valencia.

El objetivo final de este trabajo, es el de crear una plataforma de arquitectura modular, que permita la visualización de imágenes obtenidas mediante Resonancia Magnética y esté provista de las herramientas principales para tratar y procesar dichas imágenes, facilitando así, el diagnóstico final al facultativo. Para ello, se hace uso de librerías orientadas a la imagen médica y a la creación de intuitivas interfaces de usuario, como son ITK y Qt respectivamente.

Índice general

Índice de figuras

Capítulo 1: Introducción

Capítulo 2: Entorno de desarrollo

Capítulo 3: Arquitectura de la plataforma

Capítulo 4: Desarrollo e implementación de GHQdicom

Capítulo 5: Resultados

Anexo I

Anexo II

Índice de tablas

Capítulo 4: Desarrollo e implementación de GHQdicom

Capítulo 1

Introducción

1.1 Marco del proyecto

1.1.1 Ámbito del proyecto

El presente proyecto se ha realizado en el marco del departamento de comunicaciones de la Escuela Técnica Superior de Telecomunicaciones de la Universidad Politécnica de Valencia y en colaboración con la Unidad de Ingeniería Biomédica de Grupo Hospitalario Quirón (Fig. 1.1). El objetivo común del proyecto ha sido el de desarrollar un visor de imagen médica, que incluya las herramientas básicas necesarias en la rutina clínica, y sea capaz de discernir entre los diferentes tipos de secuencias de imágenes que se adquieren mediante los equipos de resonancia magnética del hospital.

Figura 1.1: Logotipo Grupo Hospitalario Quirón Valencia.

La directora del proyecto por parte de la Universidad Politécnica de Valencia, Valery Naranjo es coordinadora científica de la línea de investigación de tratamiento de señal e imagen médica en LabHuman (Fig. 1.2), un grupo que forma parte del Instituto

Interuniversitario de Investigación en Bioingeniería y Tecnología Orientada al Ser Humano (i3BH), perteneciente a la Universidad Politécnica de Valencia.

Figura 1.2: Logotipo laboratorio LabHuman.

Ángel Alberich-Bayarri es el Director Corporativo de Ingeniería Biomédica de Grupo Hospitalario Quirón. La Ingeniería Biomédica en Grupo Hospitalario Quirón se ha centrado principalmente en la imagen médica, a través de la Unidad de Cuantificación. Este grupo de cuantificación tiene su principal sede en el Hospital Quirón Valencia, desde donde se analizan estudios de imagen de los diferentes hospitales de la compañía.

Desde su creación, el grupo ha centrado sus esfuerzos en el diseño de métodos para la obtención de biomarcadores que sirvan como apoyo al diagnóstico en diversos ámbitos médicos.

1.1.2 Organización de la memoria

La memoria del trabajo está organizada de la siguiente forma: En primer lugar el capítulo actual de introducción, donde se expone el contexto y objetivos de este proyecto final de carrera, se describen la resonancia magnética y el estándar DICOM (*Digital Imaging and Communications in Medicine*), así como las técnicas de imagen avanzadas por RM, y por último se hace un repaso del estado del arte en visores de imagen médica. Posteriormente, en el segundo capítulo se presentan las herramientas

software con las que se ha desarrollado la aplicación. En una nueva sección, se detalla la estructura interna de la aplicación. Es en el cuarto capítulo donde se expone la aplicación desarrollada, proporcionando una guía para el programador, cumpliendo así con uno de los objetivos primordiales de este proyecto, la posibilidad de ampliar el software mediante la inserción de *plugins* que realicen una función específica. Seguidamente en el capítulo de resultados, se muestra y detalla la aplicación desde el punto de vista del usuario, con capturas de pantalla que ayuden al mejor entendimiento de los menús y submenús que la conforman. Por último, se aborda la discusión sobre la imagen médica y las metodologías de visualización actuales y se comentan las conclusiones así como líneas futuras del trabajo realizado. Al final del documento se pueden encontrar los anexos del trabajo.

1.1.3 Objetivos del proyecto

El objetivo primordial de este proyecto es el de crear una plataforma de visualización y procesado de imágenes médicas, de código abierto, independiente de fabricantes, que permita la inclusión de módulos de procesado de imagen y de biomarcadores de imagen.

Son varios los objetivos específicos que persigue este proyecto y quedan detallados a continuación:

- Comprensión de los fundamentos de la resonancia magnética así como los aspectos teóricos de las técnicas avanzadas de adquisición de imagen (difusión y perfusión).
- Desarrollo de una parte software de lectura de volúmenes de imágenes: los resultados de exploraciones no invasivas, en concreto, resonancia magnética

(RM) se presentan en formato de imagen médica DICOM. La aplicación debe ser capaz de leer este formato.

- Capacidad de la aplicación para visualizar los diferentes *frames* de la serie dependiendo el tipo. Series estáticas se visualizaran mediante un slider mientras que series dinámicas lo harán por medio de dos sliders.
- Desarrollo de una parte software que permita técnicas de tratamiento de imagen para facilitar el diagnóstico del radiólogo.
- Implementación de una herramienta con la que se pueden generar y registrar los informes pertinentes sobre cierto paciente.
- Capacidad del visor de cargar estudios adquiridos de cualquier fabricante de equipos de resonancia magnética.

1.2 La ingeniería y la imagen médica

La medicina está cambiando, hoy en día los avances hacia la medicina personalizada se basan en el trabajo de equipos multidisciplinares compuestos por físicos, matemáticos, ingenieros y por supuesto médicos. Uno de estos avances es el de la biopsia virtual con biomarcadores de imagen. Gracias a que la imagen médica ya es digital esta ya no es un producto final y es posible tratarla y procesarla mediante métodos ingenieriles, de esta manera es posible proporcionar información de manera objetiva a las unidades de diagnóstico por imagen y hospitales de cualquier lugar del mundo.

Los biomarcadores de imagen nacen gracias a la sinergia entre las imágenes digitales y su procesado computacional. Es posible extraer biomarcadores a partir de cualquier modalidad de imagen para diagnóstico, pero de entre todas las existentes, la

resonancia magnética destaca por su gran versatilidad para estudiar distintos tejidos y procesos así como por sus características no ionizantes [1].

Para aplicar adecuadamente los biomarcadores de imagen se establecen unos pasos a seguir, desde su concepción teórica hasta su implantación asistencial [2]:

- Prueba de concepto: Debe verificar que un proceso biológico determinado, entendido como una cadena de causas y efectos, se puede estudiar con las pruebas disponibles de diagnóstico por imagen y computación.
- Prueba de mecanismo: Se basa en la demostración de la interrelación entre el biomarcador, el parámetro que se analiza y la enfermedad objeto de estudio.
- Adquisición de imágenes: La imagen es crucial para la extracción de biomarcadores, es por ello que en la adquisición se deben cuidar aspectos como la relación señal-ruido (RSR) y la relación contraste-ruido (RCR).

  Además el órgano bajo estudio, debe analizarse con la suficiente resolución espacial, definida ésta por el tamaño del vóxel.
- Procesado de imágenes: Es necesario garantizar que los datos adquiridos sean óptimos para el análisis, para ello, se debe incrementar la RSR tanto como sea posible por medio de filtros. Es recomendable también, homogenizar la imagen para eliminar señales y brillos artefactuales, cabe además, maximizar la resolución espacial sin artefactar la señal mediante algoritmos de super-resolución.
- Análisis y modelado: Proceso en el cual se extraen de las imágenes médicas digitales la información de los biomarcadores mediante los procesos computacionales adecuados. Los métodos existentes se agrupan en función de los parámetros que se analizan (métodos para calcular el volumen y forma

de un tejido, su topología y algunas propiedades físicas, químicas, biológicas y funcionales).

- Medidas: A partir de la representación de la distribución espacial del biomarcador mediante imágenes paramétricas, se pueden obtener mediciones bien de todo el tejido u órgano estudiado, bien solo de aquellas áreas que se consideren más representativas o anormales. A partir del histograma es posible obtener resultados como el valor medio, la desviación típica y el rango.
- Prueba de principio: Proceso necesario en el cual se realiza una experiencia piloto que valide el proceso en un grupo de sujetos, con y sin enfermedad, con un tamaño muestral pequeño. Se centra en demostrar la validez de las pruebas de concepto y de mecanismo, ambas teóricas.
- Prueba de eficacia: Permite valorar la capacidad de un biomarcador de medir adecuadamente el concepto. Se realiza con series de un tamaño muestral adecuado.
- Prueba de efectividad: Analiza en condiciones habituales de uso, la bondad de un biomarcador para medir la variable clínica relevante a la enfermedad.

1.3 La RM y el estándar DICOM

1.3.1 Propiedades magnéticas de los núcleos

El principio de la resonancia magnética es el de aprovechar las propiedades mecánico-cuánticas de los núcleos atómicos. Todos los núcleos que posean un número impar de protones o neutrones tienen un espín (campo magnético debido a la rotación

del núcleo) diferente a cero, es decir, se caracterizan por un momento magnético y un momento angular intrínseco. Esta propiedad es fundamental para poder aplicar la RM.

Los núcleos comúnmente más empleados en resonancia magnética son los de los átomos de hidrogeno que encontramos en el agua presente en el cuerpo humano, pero también son de utilidad los de otros muchos elementos, como por ejemplo el flúor, fósforo.

Para aplicar la técnica de resonancia magnética es imprescindible que los espines que giran alrededor del núcleo atómico lo hagan de forma alineada (Fig. 1.3). Es por ello, que se requiere de un campo magnético externo que sea estacionario (1 Tesla, 1.5 Teslas, 3 Teslas).

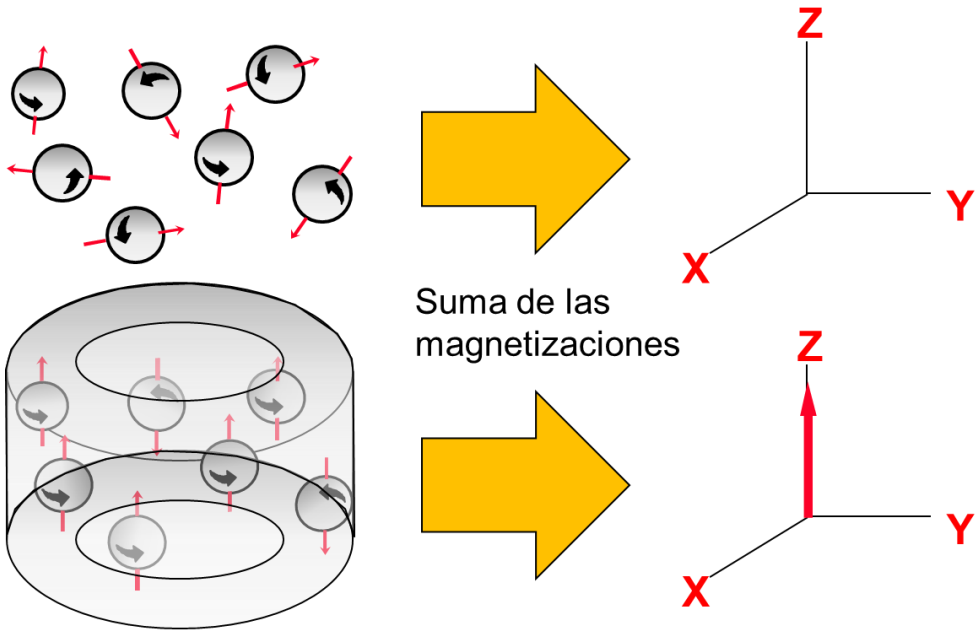


Figura 1.3: Espines en orientación arbitraria (Superior) y spines alineados con el campo magnético, magnetización resultante positiva en eje z (Inferior).

Las frecuencias a las cuales resuena un núcleo atómico son directamente proporcionales a la fuerza del campo magnético estacionario aplicado, de acuerdo con la ecuación de la frecuencia de precesión de Larmor la cual queda reflejada a continuación:

$$fo = \frac{\omega_0}{2\pi} = \frac{\gamma}{2\pi} B_0$$

donde γ se corresponde con el ratio giromagnético (cuyo valor en caso de excitar protones es γ = 42.58 MHz/T) y B0 hace referencia a la inducción magnética aplicada [3].

1.3.2 Principios físicos de la Resonancia Magnética Nuclear

Una vez que se dispone de hidrógeno con los espines en posición de equilibrio, precesando de forma alineada alrededor del eje longitudinal, es cuando se procede a excitarlos, a la frecuencia de Larmor, mediante un campo magnético de radiofrecuencia transversal al campo magnético estacionario (Fig. 1.4). A consecuencia de este fenómeno se produce una inducción de un par de fuerzas que desplaza la rotación de los espines a estados diferentes a los de equilibrio.

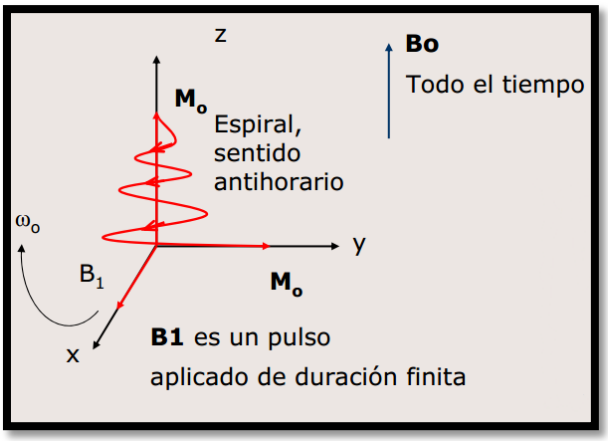


Figura 1.4: Excitación de los espines en reposo mediante un pulso de RF de duración finita [3].

La aplicación de este pulso de RF a la frecuencia de Larmor hace que la magnetización neta de los espines gire en torno al eje en que se aplica pulso con un ángulo que depende de la duración de dicho pulso y de la intensidad del campo:

$$\boldsymbol{\alpha = \omega_1 \tau = \gamma B_1 \tau}$$

donde γ se corresponde con el ratio giromagnético (cuyo valor en caso de excitar protones es γ = 42.58 MHz/T), B1 hace referencia a la intensidad de campo magnético aplicada y τ es la duración del pulso de radiofrecuencia.

Tras cesar la excitación con el pulso de radiofrecuencia, los espines vuelven al estado de equilibrio. Gracias a esta vuelta al estado de equilibrio, es posible medir ciertos parámetros mediante los cuales se obtiene la imagen por resonancia magnética [3].

El parámetro T1 es el tiempo que tarda la magnetización longitudinal en recuperar el 63% de su estado de equilibrio. Depende de la estructura molecular del tejido y del campo magnético aplicado (B0). El parámetro T1 mide el retorno longitudinal de los protones para su alineación con el campo magnético externo después que se ha interrumpido el pulso de RF. Si el tejido está formado por agua pura o líquido (ej. líquido cefalorraquídeo, saliva, humor vítreo, etc.), las pequeñas moléculas de agua, tardan bastante tiempo en transferir su energía. Esto significa que dichos líquidos presentan un T1 prolongado y aparecen de un color ennegrecido en las imágenes de RM (Fig. 1.5). Las moléculas de mayor tamaño, como las del tejido graso, transfieren la energía más rápidamente. La grasa presenta un T1 corto y tiene apariencia blanquecina o brillante en las imágenes de RM ponderadas en T1. Esta secuencia es útil para evaluar la morfología.

El parámetro T2 es el tiempo que transcurre hasta que la magnetización transversal posee el 37% del valor en que cesa el pulso de RF. El parámetro T2 depende únicamente de la estructura molecular del tejido y no del campo magnético aplicado. Con respecto al agua pura y otros líquidos, estas moléculas permanecen al paso durante un largo período de tiempo, por lo que la secuencia T2 es prolongada y aparecen blancas o brillantes en las imágenes ponderadas en T2 (Fig. 1.6). En cambio, la imagen de la grasa es de menor señal.

Como la mayor parte de los procesos patológicos dan lugar a un incremento en la cantidad de agua libre o de volumen, las imágenes en T2, se utilizan con mayor frecuencia para detectar cuadros patológicos.

El concepto de secuencias o imágenes ponderadas en T1 y en T2 sirve para la comprensión de la escala de grises de las imágenes de RM y cabe destacar que dichas secuencias ocurren simultáneamente de manera independiente [4].

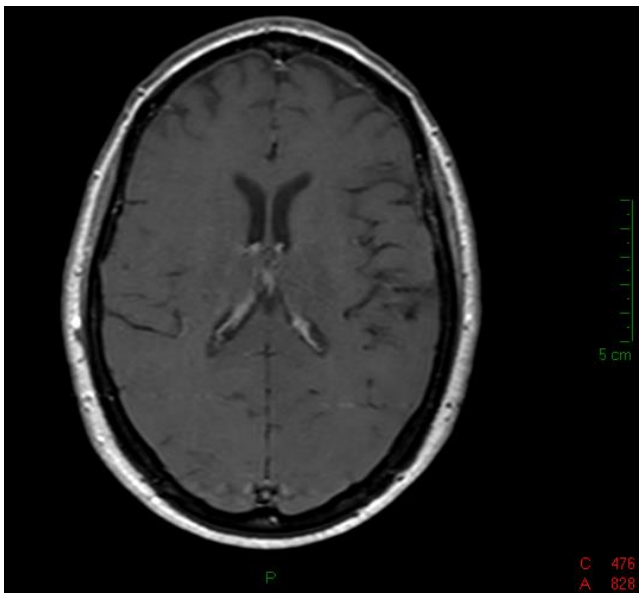


Figura 1.5: Imagen ponderada en T1.

Figura 1.6: Imagen ponderada en T2.

1.3.3 El estándar DICOM

Figura 1.7: Logotipo del estándar DICOM.

DICOM es un estándar creado por ACR (*American College of Radiology*) y NEMA (*National Electrical Manufacturers Association*), destinado al almacenamiento, intercambio, impresión y transmisión de imágenes médicas (Fig. 1.7). El estándar DICOM establece la forma de almacenar las imágenes digitales adquiridas sobre un paciente, así como los datos asociados a dicha adquisición. Otra característica importante es la posibilidad de interconectar sistemas informáticos de diferentes fabricantes y haciendo posible la comunicación entre sí [5].

Los ficheros DICOM se pueden diferenciar claramente en cuatro partes (Fig. 1.8).

- Preámbulo: Tamaño fijo de 128 bytes, pensado para tener un uso definido por la implementación. Puede incluir el nombre de la aplicación utilizada para la creación del fichero o incluso información que permita a cierta aplicación acceder directamente a los datos de la imagen almacenada en el fichero.
- Prefijo: Seguidamente a los 128 bytes de preámbulo encontramos 4 bytes, estos contienen sucesivamente los siguientes caracteres ‘D’,’I’,’C’,’M’ y son un indicador para las aplicaciones de forma que permite conocer si el formato del fichero es DICOM.

- Cabecera y Metacabecera: En la cabecera se especifican datos administrativos (datos del paciente, equipo de adquisición, tipo de estudio, condiciones en las que se tomó la imagen, datos del hospital, etc.) organizados en elementos de datos.
- Imagen: Una imagen se define como una matriz bidimensional compuesta por sus coordenadas espaciales (x,y) y su intensidad o nivel de gris. La imagen consiste en un mapa de densidades. Esta información, puede interpretarse como una imagen en escala de grises si se hace una superposición de la escala de grises sobre la escala de densidades (unidades Hounsfield) mediante la transformación definida por el estándar DICOM. Cabe destacar que aunque se ha separado la imagen como parte independiente del fichero DICOM, ésta se encuentra como un elemento de datos más de la cabecera (Fig. 1.9).

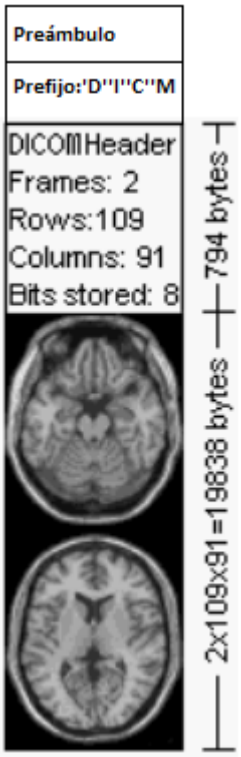


Figura 1.8: Estructura de un fichero

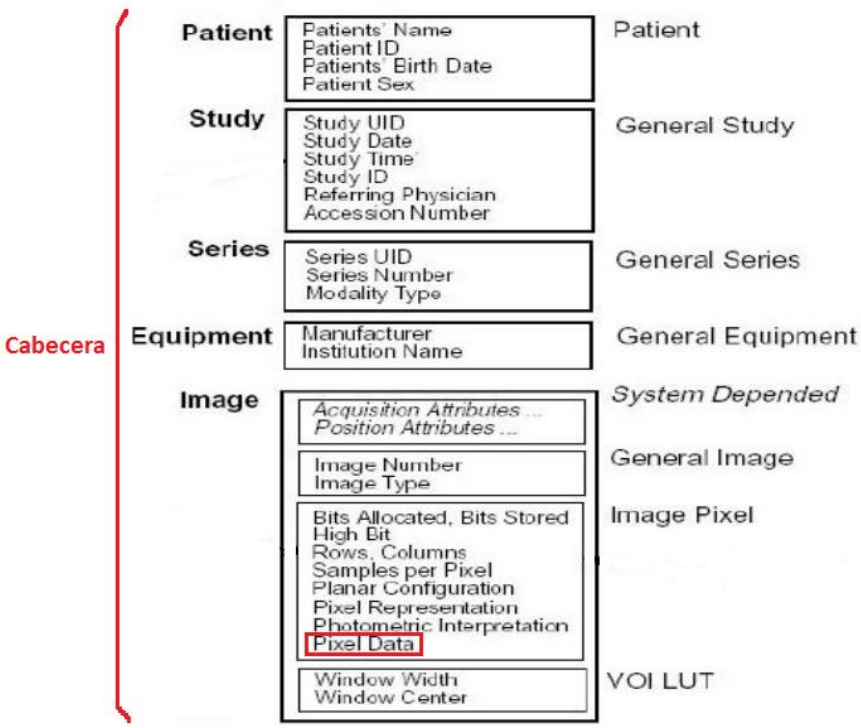


Figura 1.9: Estructura de la cabecera DICOM.

1.3.4 Técnicas de imagen avanzadas por RM

Los estudios dinámicos en imagen médica, son aquellos que se encargan de visualizar procesos fisiológicos que varían a lo largo del tiempo. Estos procesos, suelen estar ligados al movimiento propio de los fluidos corporales, como la sangre y el agua. Gracias a dicho movimiento, es posible la cuantificación de parámetros relevantes para el diagnóstico clínico. Existen parámetros fácilmente cuantificables como la velocidad o el flujo, basados en la cinética de fluidos. Por el contrario, existen otros con características dinámicas más complejas, por lo que no son tan obvios de extraer, como por ejemplo, la dirección de difusión principal de las moléculas de agua en los tejidos. La complejidad de la cuantificación vendrá determinada por varios factores:

- Será necesario utilizar la técnica de adquisición de imagen más adecuada con suficiente resolución espacial y temporal para así observar el proceso y seguirlo adecuadamente a lo largo del tiempo, disponiendo así, de toda la información necesaria para la cuantificación.
- Se debe tener en cuenta la región anatómica a estudiar, así como las relaciones existentes con estructuras anatómicas adyacentes que puedan resultar determinantes para una correcta cuantificación.
- Procesado de las imágenes que mejore en lo posible la calidad de las mismas, puesto que dicha calidad se puede ver afectada por la alta probabilidad de movimientos involuntarios por parte del paciente durante la adquisición. Este procesado suele consistir en la aplicación de filtros de reducción y/o realce de bordes y algoritmos de estimación de movimiento [6].

1.3.4.1 Cuantificación de la difusión del agua

El movimiento o difusión molecular en el cuerpo humano tiene lugar especialmente entre las moléculas de agua. En los tejidos de nuestro cuerpo, las moléculas de agua pueden presentar un movimiento sin barreras (difusión libre o desplazamiento molecular isotrópico), o un movimiento limitado por el entorno, por ejemplo, una maraña densa de membranas celulares (difusión restringida o desplazamiento molecular anisotrópico).

En términos generales, las medidas de difusión expresan el desplazamiento efectivo en el espacio de las moléculas de agua en un intervalo de tiempo. La temperatura tiene cierta contribución y modula el movimiento molecular de las moléculas de agua (aproximadamente un 2.4% por grado Celsius), sin embargo, esta influencia térmica no es determinante en el estudio de la difusión, existen otras propiedades biofísicas que presentan un efecto más relevante en la movilidad del agua tisular.

Suponiendo que en un instante temporal tenemos una agrupación de moléculas concentradas en un punto determinado, si dejamos pasar un tiempo t, sin ejercer ninguna acción sobre las mismas, éstas se habrán expandido según la ecuación de difusión en tres dimensiones de Einstein:

$$\boldsymbol{r^2 = 6 \cdot D \cdot t}$$

donde t es el intervalo de tiempo transcurrido y r el radio promedio de la distribución. Como podemos observar, el coeficiente de difusión D, se expresará mediante las unidades de distancia al cuadrado por unidad de tiempo. Para su aplicación en medicina, se suele expresar en mm^2/s.

La RM es la única modalidad de imagen que permite la visualización y el cálculo de la difusión molecular "in vivo" directamente a partir del movimiento aleatorio o browniano de las moléculas de agua presentes en cierto tejido, pues detecta movimientos microscópicos de dichas moléculas (Fig. 1.10).

Durante el desfase de los espines posterior a la finalización del pulso de RF ocurren incoherencias en la relajación de los espines debidas a la agitación térmica de la molécula de agua, lo que produce que la pérdida de sincronismo de los espines se produzca más rápidamente y se acelere el proceso de relajación. Por otra parte, los repetidos desplazamientos de las moléculas de agua hacen que los espines nucleares difundan a otras regiones en las que el campo magnético difiere del original, provocando una modulación en la frecuencia de relajación.

Para poder cuantificar el movimiento de desplazamiento por parte de los espines de manera independiente, se aplica un gradiente en una determinada dirección justo después de aplicar el pulso de radiofrecuencia. En esta situación, las moléculas de agua que se hayan desplazado en la dirección del gradiente, estarán bajo un campo magnético que distará más del original y, por tanto, la caída de señal será más rápida [6].

Suponemos en primer lugar que adquirimos una imagen sin la aplicación de gradientes de difusión y obtenemos una señal S_0, si ahora aplicamos un gradiente de difusión y obtenemos una señal S, la relación entre ambas señales vendrá dada, de manera general, por la expresión:

$$\boldsymbol{S = S_0 \cdot e^{-b \cdot D}}$$

donde D es el coeficiente de difusión, y b depende de los gradientes de difusión aplicados. Los valores de D expresan la distancia promedio que recorren las moléculas

de agua en el vóxel y se relaciona con la densidad celular, la permeabilidad e integridad de sus membranas y la tortuosidad del intersticio intercelular. Cuanto mayor sea el factor ‘b’, mayor será la sensibilización al movimiento y menos influencia tendrá la potenciación T2 en la señal de la imagen.

Una de las características más llamativas de las imágenes de difusión es su alta sensibilidad para detectar lesiones. Los tumores son frecuentemente más celulares que el tejido del que provienen, y por tanto, aparecen con una intensidad de señal mayor (difusión restringida) en las imágenes de difusión isotrópica. Conforme los tumores se desdiferencian y aumenta su proliferación y agresividad biológica, la difusión estará más restringida.

También se utiliza la difusión como marcador de la eficacia del tratamiento quimio y radioterápico, dado que la necrosis celular, la reducción de densidad celular y el daño a la membrana ocasionan un aumento de la difusión previamente restringida [7].

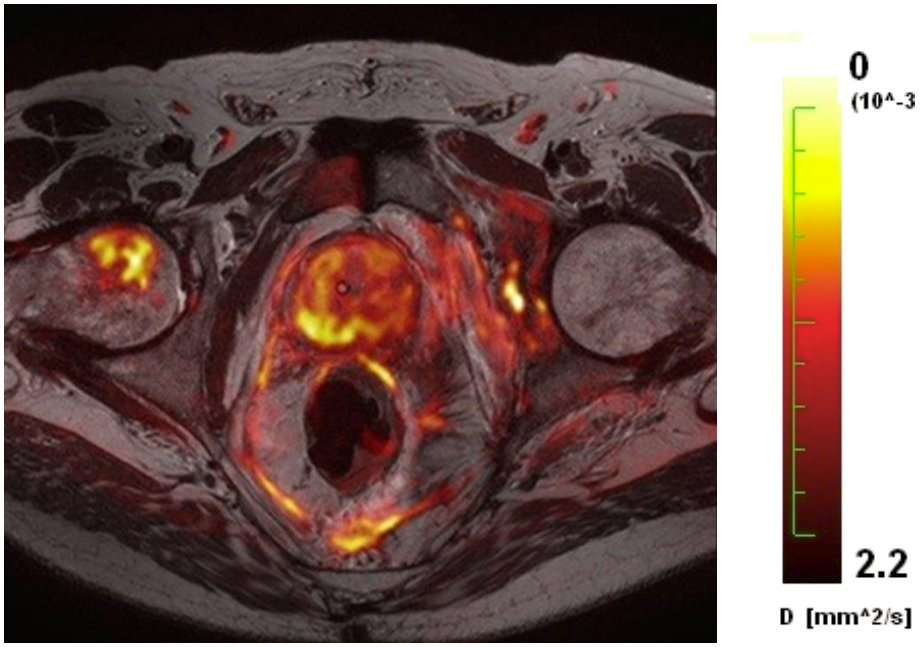


Figura 1.10: Mapa de difusión de la región pélvica.

1.3.4.2 Cuantificación de la perfusión

La perfusión se define como el proceso por el que la sangre de los vasos arteriales se difunde en un tejido a través de los capilares. La perfusión viene determinada por la siguiente relación:

$$\boldsymbol{P = (P_{art} - P_{ven})/R}$$

donde P_{art} y P_{ven} corresponden a la presión arterial y venosa y R viene dada por la resistencia vascular, es decir, el nivel de resistencia que ofrece el vaso al paso de la sangre.

En los estudios de perfusión por RM se suele utilizar una sustancia denominada medio de contraste, que se inyecta por vía intravenosa y modifica propiedades magnéticas de la sangre, de manera que a medida que se adquieren las imágenes del estudio dinámico se va produciendo un aumento de la diferencia de intensidad entre las regiones perfundidas y el resto de los tejidos, hasta alcanzar un nivel máximo cuando la concentración de contraste es máxima.

La cuantificación se basa en la caracterización de unas curvas temporales que se obtienen observando regiones en la imagen, es decir, promediando la intensidad de dichas regiones para cada imagen del estudio dinámico; o realizando un análisis píxel a píxel, de manera que para cada píxel se cuantifica una curva, con lo cual podemos generar mapas coloreados superpuestos a la imagen anatómica con información regional de cada parámetro cuantificado (Fig. 1.11).

Son diversas las metodologías que se utilizan para la cuantificación de la perfusión y el flujo sanguíneo: la aproximación semicuantitativa, la aproximación

compartimental mediante modelos, la técnica *Arterial Spin Labeling* (ASL) y la técnica *Intravoxel Incoherent Motion* (IVIM) entre otras [6].

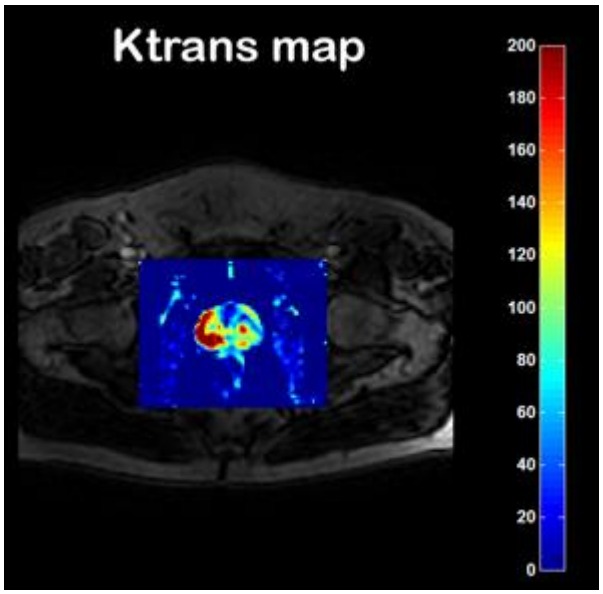


Figura 1.11: Mapa paramétrico coloreado de la permeabilidad vascular de la próstata.

1.4 Estado del arte en visores de imagen médica

La forma de visualizar y tratar las imágenes médicas precisa de un ordenador que incorpore un software capaz de cargar las series de imágenes adquiridas sobre cierto paciente y que incluya herramientas para su procesado con el objetivo de obtener un mejor diagnóstico.

Son muchos los visores de imágenes DICOM que existen en la actualidad, pero no todos ellos están orientados a las tecnologías de la información en el campo de la Medicina, es decir, existen muchos otros que precisan de herramientas de imagen en general, herramientas que no son de utilidad en el ámbito biomédico, sin embargo, carece de algunas que resultan esenciales en este campo. Son varias las características

que se requieren en un software de imagen médica para el uso diario en el entorno clínico, por ello, vamos a hacer un repaso de los visores actuales, que más se ajustan al entorno biomédico, destacando los pros y contras de cada uno de ellos.

1.4.1 ImageJ

ImageJ es un programa de procesamiento de imagen digital de dominio público programado en Java, desarrollado en el National Institutes of Health. ImageJ fue diseñado con una arquitectura abierta que proporciona extensibilidad vía *plugins* Java y macros (macroinstrucciones) grabables. Se pueden desarrollar *plugins* de escaneo personalizado, análisis y procesamiento usando el editor incluido en ImageJ y un compilador Java. ImageJ puede ejecutarse en un *applet* en línea, como aplicación ejecutable, o en cualquier computadora con Máquina virtual Java 5 o superior. Hay también distribuciones descargables para Microsoft Windows, Mac OS, Mac OS X, Linux, y Sharp Zaurus PDA [8].

Una de las propiedades básicas de las que dispone este software es en primer lugar la capacidad de lectura de imágenes en formato DICOM, PNG, JPEG y BMP. Además nos ofrece una serie de herramientas de procesado de imagen digital como la obtención de histogramas de densidad, gráficos línea perfil, operaciones lógicas y aritméticas entre imágenes, variación contraste/brillo, filtrado de mediana y transformaciones geométricas (zoom, rotación, etc.).

Entre los aspectos negativos del software se puede destacar que se trata de una plataforma sin un estándar definido, donde coexisten aplicaciones de todo tipo, no solo de imagen biomédica. Además el software es incapaz de diferenciar el tipo de estudio

que se carga, permitiendo visualizarlo solo mediante un slider que no separa funcionalmente los diferentes parámetros de interés de dicho estudio. Otro aspecto negativo de la aplicación, es la falta de herramientas para el uso clínico, como por ejemplo, una regla que permita la medición sobre la imagen, imposibilidad de escribir un informe clínico sobre un determinado paciente, etc.

Todos estos aspectos hacen de ImageJ un software que no se ajusta a las necesidades diarias de un hospital.

1.4.2 Gimias

Gimias (interfaz gráfica para el análisis de imágenes médicas y simulación) es una plataforma de software, diseñada por la Universisad Pompeu Fabra (UPF) de Barcelona, como una herramienta integradora de aplicaciones médicas. Se trata de un flujo de trabajo orientado al entorno de computación avanzada de imagen biomédica y simulaciones, donde un problema específico se puede ampliar mediante el desarrollo de plugins.

Gimias está construido sobre C + + y bibliotecas como VTK (kit de herramientas de visualización), ITK (*Insight Toolkit*), MITK (*Medical Imaging Interaction Toolkit*) y el kit de herramientas Dicom (DCMTK) entre otras. Este software esta liberado bajo una licencia BSD para permitir la cooperación entre los desarrolladores y gupos de investigación. Gimias está disponible para la plataforma Windows tanto en modalidad de 32 como 64 bits y existen versiones beta para Linux y MAC [9].

En cuanto a los aspectos positivos del software cabe destacar su conexión directa con los sistemas PACS (*Picture Archiving and Communication System*). Los formatos de entrada y salida son los propios de imagen médica (Dicom, Nifti y Analyze). Por otra parte, es de vital importancia, la inclusión de herramientas propias de uso diario en un hospital, como elementos para tomar medidas sobre la imagen, visualización en modo cine de las series de imágenes cargadas (incluyendo control de velocidad) y visualización de imágenes 3D, con opción de mallado superficial y volumétrico.

La desventaja más importantes de Gimias, desde el punto de vista del uso clínico, es la imposibilidad de diferenciar el tipo de estudio que se carga, permitiendo visualizarlo solo mediante un slider que no separa funcionalmente los diferentes parámetros de interés de dicho estudio. Cabe destacar también que se trata de una aplicación muy centrada en el área de cardiología y neurología, dejando otros campos más descubiertos.

Después de un estudio a fondo del software podemos concluir afirmando que se trata de una herramienta muy potente, pero es tal su complejidad, que se hace inviable su uso en el día a día del hospital.

1.4.3 Weasis

Weasis es un software construido en Java, basado en arquitectura modular, diseñado para satisfacer las expectativas de varios sistemas de información clínica que marca la evolución en lo respectivo a la imagen médica. Proporciona acceso web a las imágenes radiológicas y ofrece capacidades multimedia. Esta aplicación puede conectarse fácilmente a cualquier tipo de PACS a través de un portal web [10].

Este software se caracteriza por su fácil e intuitivo manejo (sliders y botones del ratón configurables). Posee herramientas como variación brillo/contraste, zoom, rotación, filtros y LUTS de color, elementos de medida y capacidad de visualizar los datos cargados en modo cine, que hacen que esta aplicación sea idónea para el uso en el entorno clínico.

Como aspecto negativo solo se encuentra (al igual que en los dos software anteriormente comentados) la imposibilidad de diferenciar el tipo de estudio que se carga, permitiendo visualizarlo solo mediante un slider que no separa funcionalmente los diferentes parámetros de interés de dicho estudio.

Debido a sus excelentes características, el estudio de Weasis se ha realizado más a fondo llegando a compilar su código fuente con la intención de observar su estructura y efectuar ligeras modificaciones para comprender mejor el funcionamiento.

La construcción y compilación de Weasis se puede llevar a cabo mediante terminal bash o a través de interfaz gráfica. En el Anexo I se reflejan las dos formas de compilar el código fuente, adquirido de cierto repositorio web, y se explica la estructura de la aplicación.

Después de este repaso a través de los visores de imagen médica que mejor se adaptan a las necesidades hospitalarias, se han extraído ciertas consideraciones que serán de gran utilidad a la hora de desarrollar nuestra propia aplicación. Se implementarán las características básicas que reúnen la mayoría de las aplicaciones de esta índole y además se tratarán de cubrir los aspectos negativos o que no contemplan, con el objetivo de desarrollar un visor DICOM con una base robusta, que pueda crecer mediante la inserción de módulos independientes capaces de resolver cierta necesidad.

Capítulo 2

Entorno de desarrollo

El desarrollo de un visor de imagen médica precisa en primer lugar de la implementación de una interfaz de usuario que albergue todas las necesidades que se requieran en un entorno clínico. En dicha interfaz debe primar la facilidad y el rápido manejo de la herramienta ya que se deben analizar multitud de casos cada día. Por este motivo, se ha escogido para implementarla el software Qt, mediante el que se pueden insertar multitud de elementos configurando una ventana muy intuitiva para el usuario.

Posteriormente se requiere del desarrollo de una parte software de lectura de volúmenes de imágenes DICOM. Para realizar esta tarea, se hace uso de librerías de procesado de imagen médica. Dichas librerías, contienen multitud de clases y métodos con los que se pueden desarrollar algoritmos que contribuyan a la mejora del diagnóstico clínico.

Por último, es necesario el uso de un software que actúe como nexo de unión entre la interfaz de usuario y los algoritmos que se implementen. Para este proyecto, se ha escogido Visual Studio 2008 por diversos motivos: su sencillez, su capacidad para integrar multitud de software y librerías en él y por la posibilidad de utilizar como lenguaje de programación C++.

A continuación, se detalla el procedimiento a seguir para establecer el entorno de trabajo integrando todas las herramientas necesarias. Es importante destacar que los

requisitos de dichas herramientas son mínimos por lo que es posible trabajar desde cualquier ordenador.

2.1 Visual Studio 2008

Visual Studio 2008 (Fig. 2.1) es un entorno integrado de desarrollo que permite la programación orientada a objetos. Entre los lenguajes que soporta se encuentran: Visual Basic, Visual C# y Visual C++, siendo éste último, el elegido para la programación de esta herramienta.

Figura 2.1: Logotipo de Visual Studio 2008.

El entorno de desarrollo integrado de Visual Studio ofrece un conjunto de herramientas destinadas a ayudar al programador a escribir y modificar código, así como a detectar y corregir errores.

Los tres pilares básicos de Visual Studio 2008 son: el desarrollo rápido de aplicaciones, la colaboración eficiente entre equipos y la innovación en experiencias de usuario.

2.1.1 Instalación de Visual Studio 2008

VS2008 se instalará en el ordenador con las opciones por defecto (Fig. 2.2).

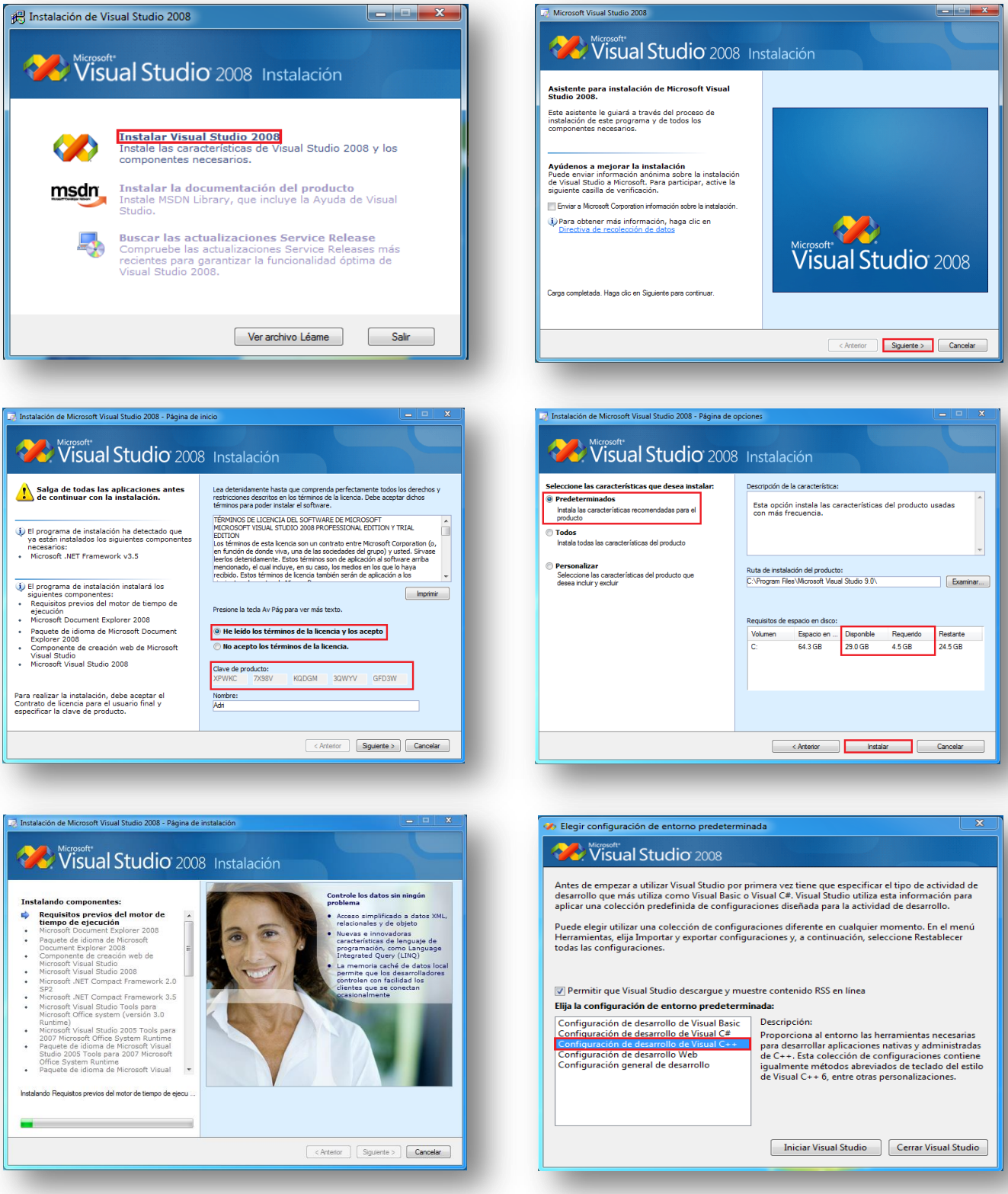


Figura 2.2: Pasos a seguir en el proceso de instalación de Visual Studio 2008.

2.2 Qt

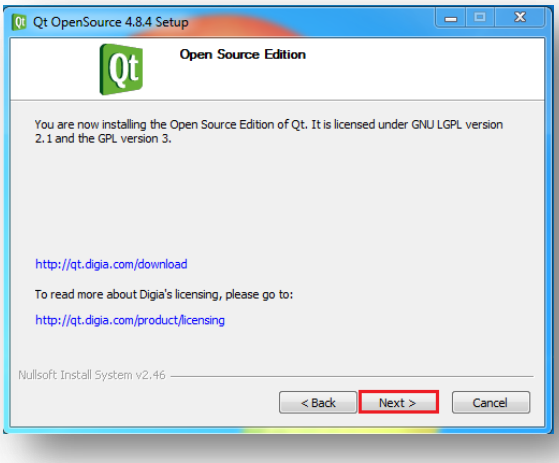
Qt (Fig. 2.3) es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario, programación web, bases de datos y también para el desarrollo de programas sin interfaz gráfica como herramientas de la consola y servidores. Qt se ha desarrollado como un software libre y de código abierto a través de Qt Project, donde participa tanto la comunidad, como desarrolladores de Nokia, Digia y otras empresas. Este software utiliza el lenguaje de programación orientado a objetos C++, aunque adicionalmente puede utilizarse con otros lenguajes de programación.

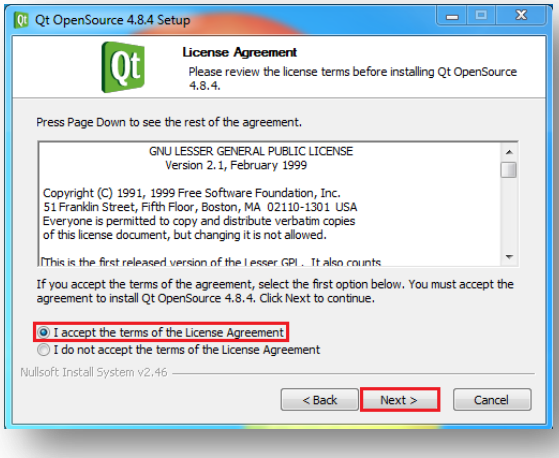
Figura 2.3: Logotipo de Qt.

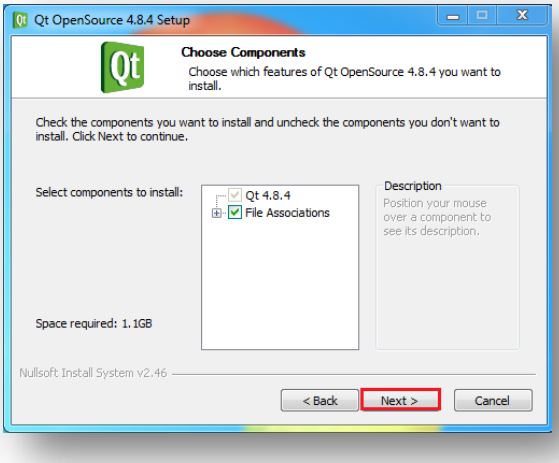
Permite realizar una programación visual y dirigida por eventos. En el caso de la programación visual, el programador centra su atención en diseñar el aspecto gráfico de la aplicación, la distribución de los elementos visuales llamados "*widgets*", la interacción entre los mismos y los distintos tipos de ventanas existentes. Por lo que respecta a la programación dirigida por eventos, el programador escribe el código que se ejecutará en respuesta a determinados eventos. No existe la idea de un control de flujo secuencial en el programa, sino que el programador toma el control cuando se dispara un evento. La labor del programador, es por tanto, asociar a cada evento el comportamiento adecuado.

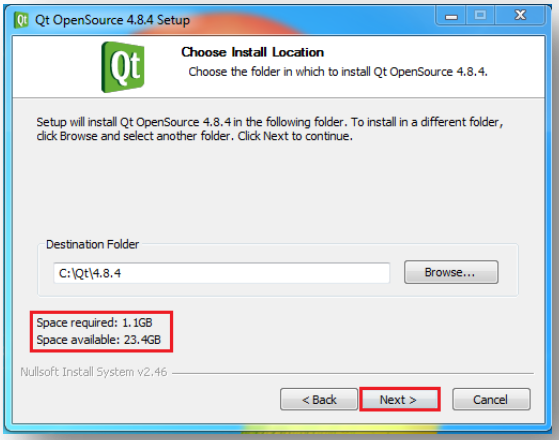
2.2.1 Instalación de Qt

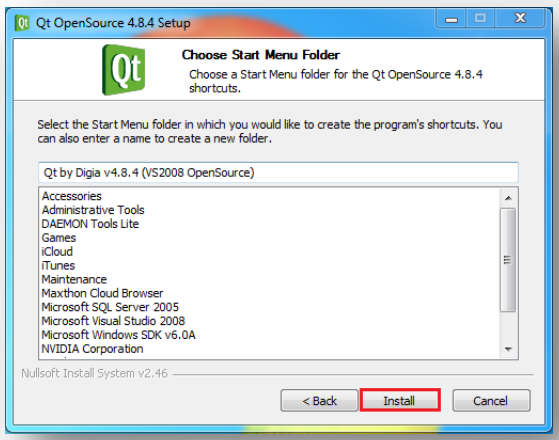
En el presente proyecto se ha utilizado la versión 4.8.4 de Qt. Para su instalación se ejecutará el fichero con extensión ".exe", de la versión comercial o de la versión de prueba proporcionada en la página web (http://qt-project.org/downloads) y se instalarán las opciones por defecto (Fig. 2.4).











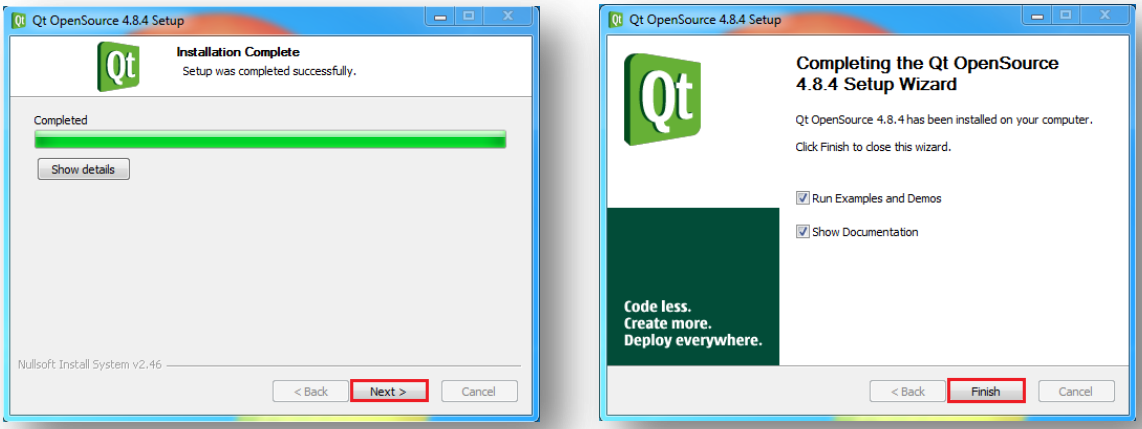
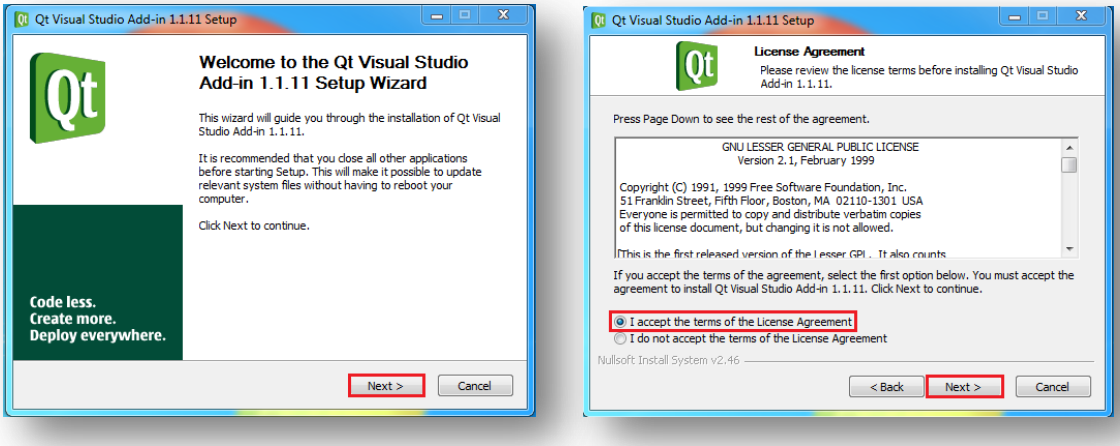


Figura 2.4: Pasos a seguir en el proceso de instalación de Qt.

2.2.2 Integración Qt con Visual Studio 2008

Se ha elegido utilizar Qt con Visual Studio 2008 aunque es compatible con todas las principales plataformas. Para la integración de Qt con Visual Studio 2008 hay que ejecutar el fichero "qt-vs-addin-1.1.11.exe" (o versiones posteriores) que proporciona Qt en la página web (http://qt.digia.com/) y se instalarán las opciones por defecto (Fig. 2.5).



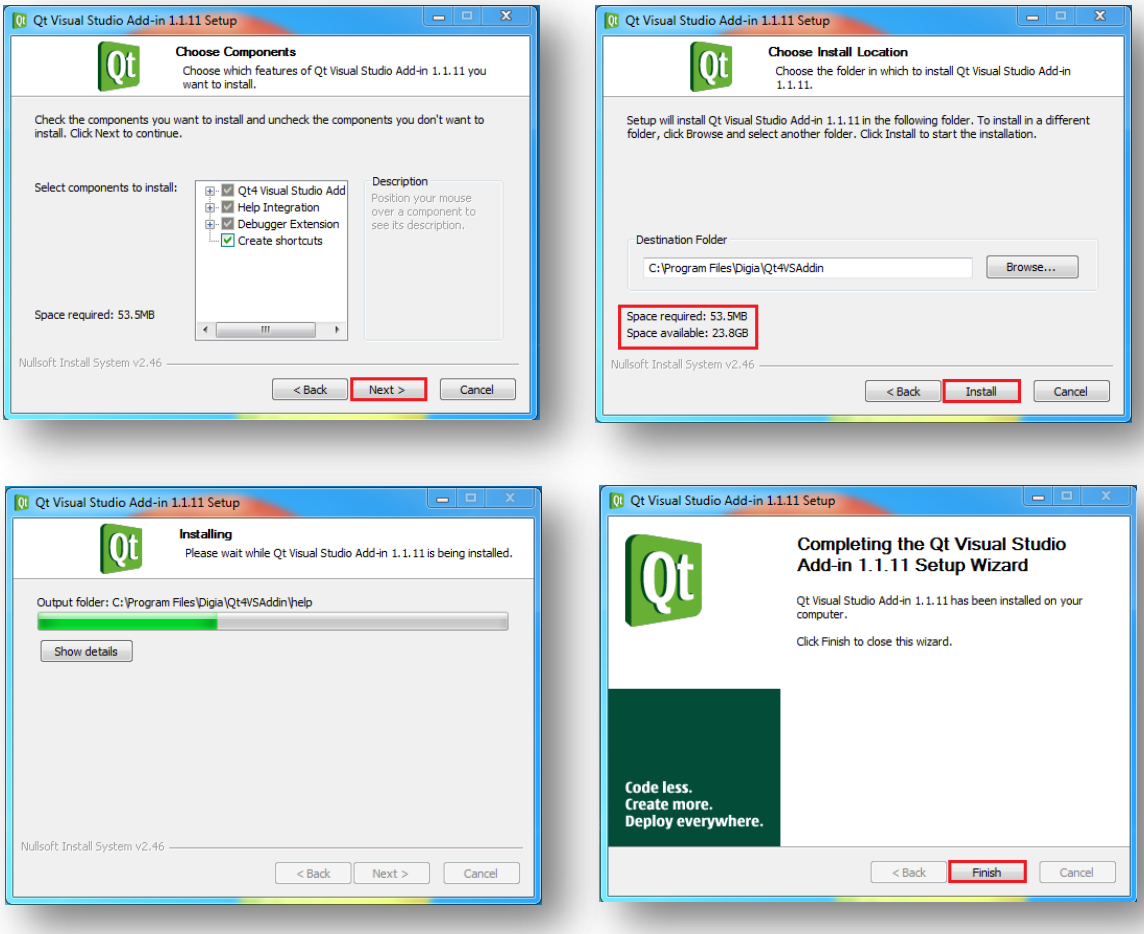


Figura 2.5: Pasos a seguir en el proceso de integración de Qt en Visual Studio.

Para comprobar que se ha instalado correctamente entramos en Visual Studio 2008 y vemos como aparece un menú de Qt (Fig. 2.6).

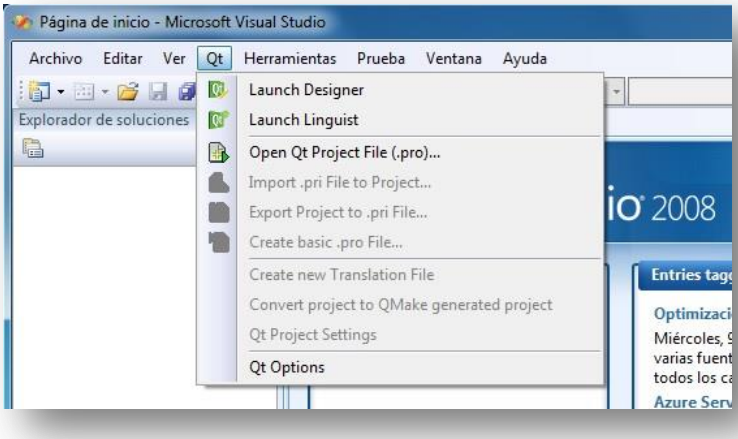


Figura 2.6: Comprobación Qt en Visual Studio 2008.

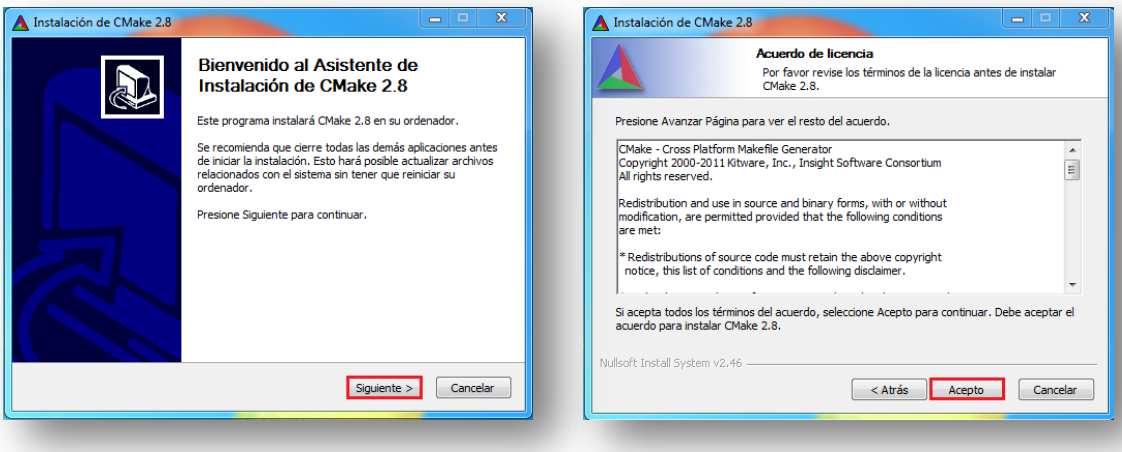
2.3 CMake

CMake (Fig. 2.7) es una herramienta multiplataforma de generación o automatización de código. Para realizar dicho proceso, se utilizan ficheros de configuración sencillos e independientes de la plataforma. Este software genera *makefiles* nativos y espacios de trabajo que pueden usarse en el entorno de desarrollo deseado.

Figura 2.7: Logotipo de CMake.

2.3.1 Instalación de CMake

Al tratarse de un compilador libre, este se puede obtener de la página web oficial (http://www.cmake.org/cmake/resources/software.html), una vez obtenido se procederá a su instalación con las opciones por defecto (Fig. 2.8).



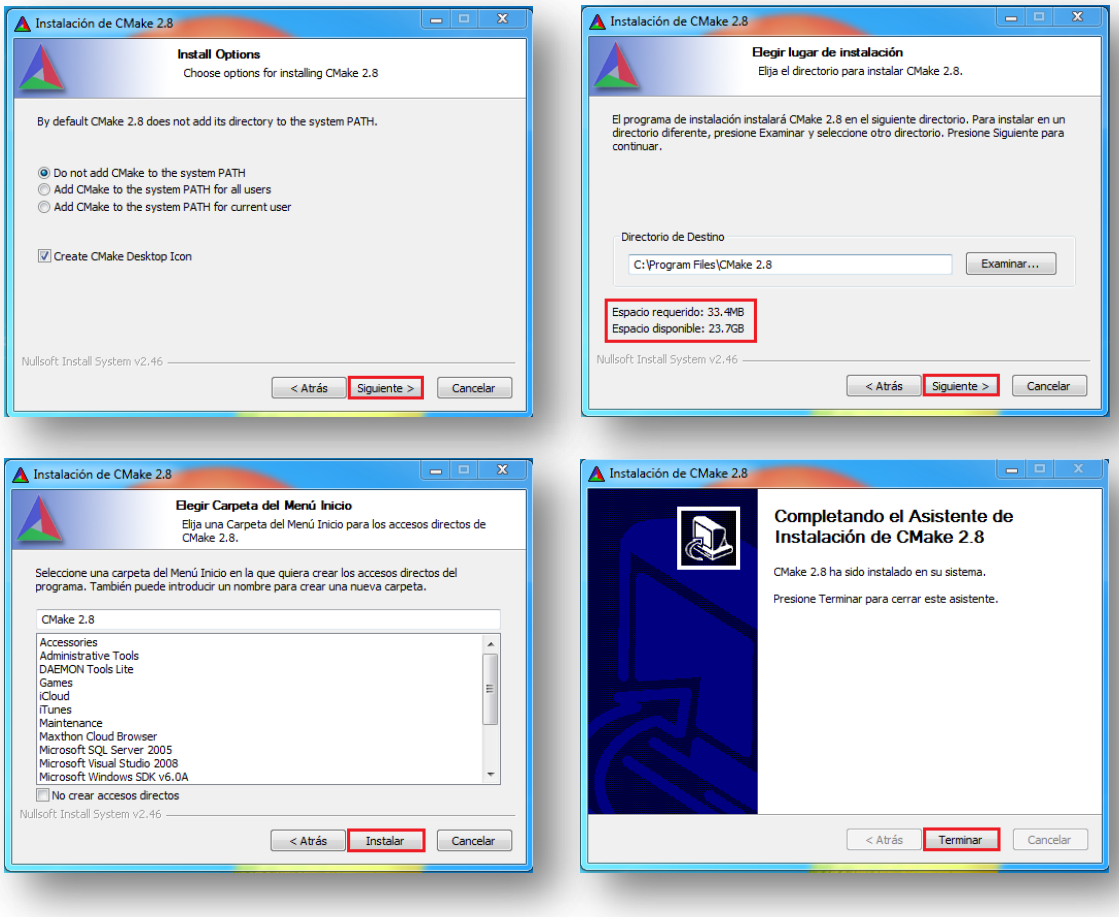


Figura 2.8: Proceso de instalación de CMake.

2.4 ITK

ITK (*Insight Segmentation and Resgistration Toolkit*) (Fig. 2.9), es un sistema multiplataforma que proporciona ciertas herramientas para el análisis y procesado de imágenes biomédicas. ITK está implementado en C++, y utiliza CMake para el proceso de compilación de manera independiente a la plataforma. ITK consiste en una serie de librerías de código abierto con capacidad de integración en herramientas como Qt.

Figura 2.9: Logotipo de ITK.

2.4.1 Descarga de las librerías ITK

Como ITK son librerías de código abierto se pueden obtener en la página web oficial (www.itk.org), concretamente en el apartado de descargas (http://www.itk.org/ITK/resources/software.html). Se deben descargar las *Insight Toolkit* y las *InsightAplications*. Se tratan de dos archivos comprimidos, por lo que el primer paso a realizar es descomprimirlos en cierto directorio. En este proyecto, se ha utilizado la versión de ITK 3.20.1 que es la más consolidada y la que mejores resultados proporciona en este momento.

2.4.2 Compilación de las librerías ITK

La compilación de las librerías ITK, se lleva a cabo a través del compilador CMake. Al ejecutar dicha aplicación, aparece la pantalla principal (Fig. 2.10) donde se deben rellenar ciertos campos. En el que indica *Where is the source code* se debe introducir el directorio donde se hayan descomprimidos las *Insight Toolkit*. Mientras que en *Where to build the binaries* habrá que poner una carpeta vacía creada por el usuario donde se guardaran las ITK compiladas que se van a crear.

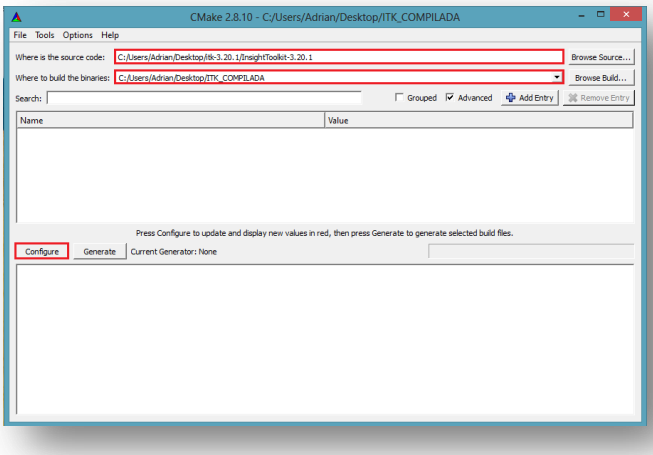


Figura 2.10: Compilando ITK en CMake.

Una vez realizado este proceso, se debe presionar el botón *Configure.* Al realizar esta acción se abre un cuadro de dialogo donde hay que introducir la plataforma en la que se va a hacer la compilación de las librerías (Fig. 2.11). En este proyecto se ha marcado la opción Visual Studio 9 (correspondiente a la versión Visual Studio 2008).

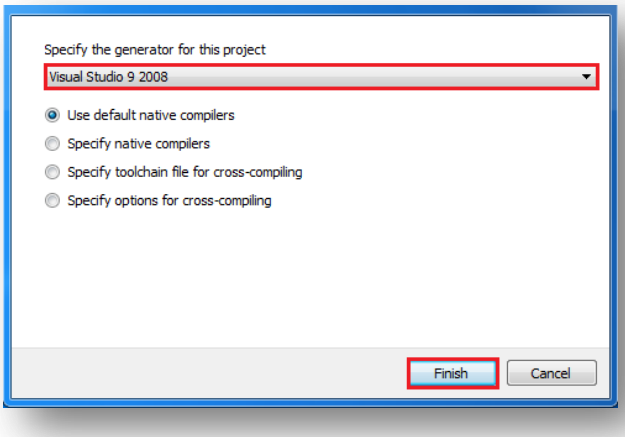


Figura 2.11: Selección de la plataforma.

Después de pulsar el botón *Finish*, se regresa a la pantalla principal de CMake, en la cual, ahora aparecen una serie de variables en rojo que pueden ser modificadas, aunque se recomienda no cambiar el valor de ninguna de ellas. Se presiona en *Configure* una vez más. Las variables pasan de estar en rojo a estar en gris y como último paso se presiona el botón *Generate*, el programa se cerrará por si solo al finalizar (Fig 2.12).

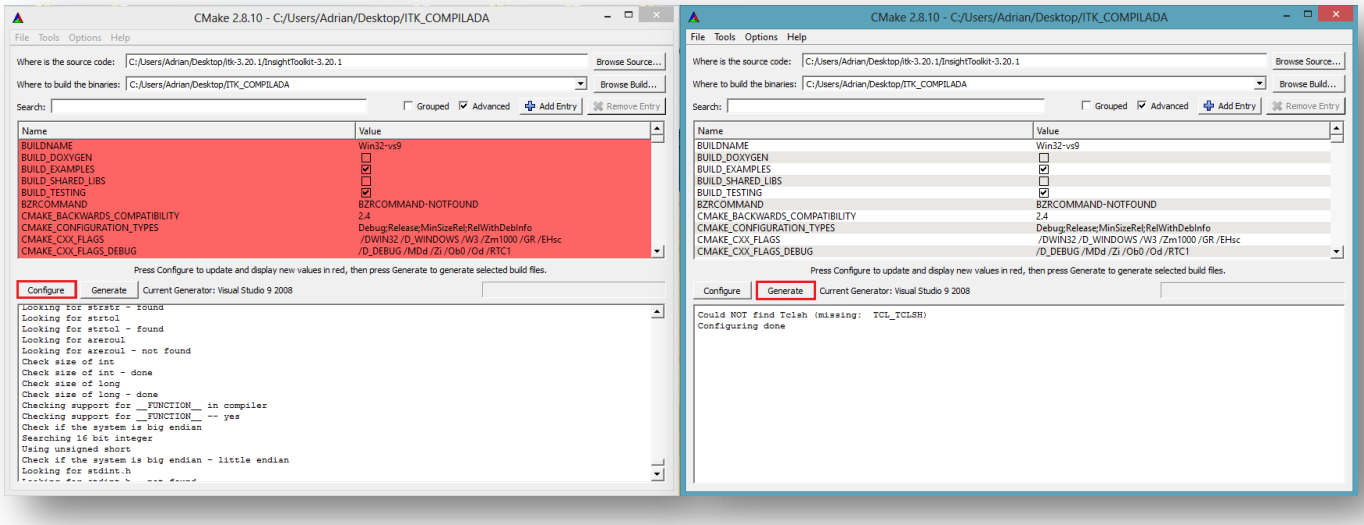


Figura 2.12: Compilación de ITK.

CMake ha convertido los archivos binarios C++ a Visual C++. Al abrir la carpeta creada por el usuario donde estarán las ITK compiladas (el directorio que se introdujo en *Where to build the binaries*) aparece un proyecto llamado "ITK.sln" y que se debe ejecutar en Visual Studio 2008.

Al abrir "ITK.sln" en Visual Studio y pulsar botón derecho sobre "ALL_BUILD", se debe pulsar en "Generar" (Fig. 2.13). Este proceso tarda alrededor de una hora (dependiendo de las características del ordenador donde se ejecute). Una vez hecho esto, se podrán incluir las librerías ITK en cualquier proyecto.

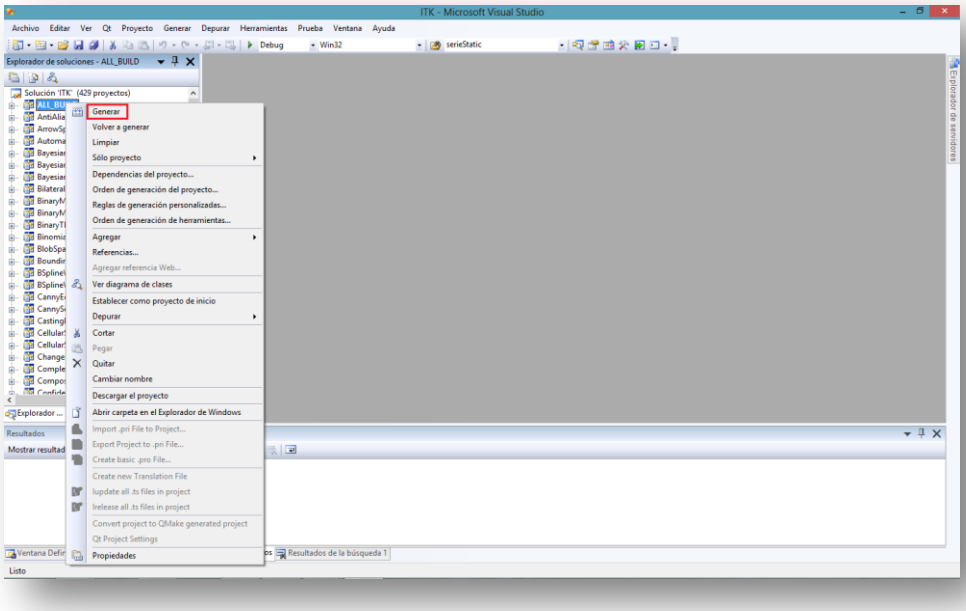


Figura 2.13: "ITK.sln" en Visual Studio 2008.

2.4.3 Integración de las librerías ITK en Visual Studio 2008

En este momento ya es posible añadir las librerías ITK al proyecto de Visual Studio 2008 en el que se vaya a trabajar. Para ello, se debe pulsar el botón derecho sobre el nombre del proyecto (Fig. 2.14) y seleccionar la opción "Propiedades" que se muestra en el menú desplegable.

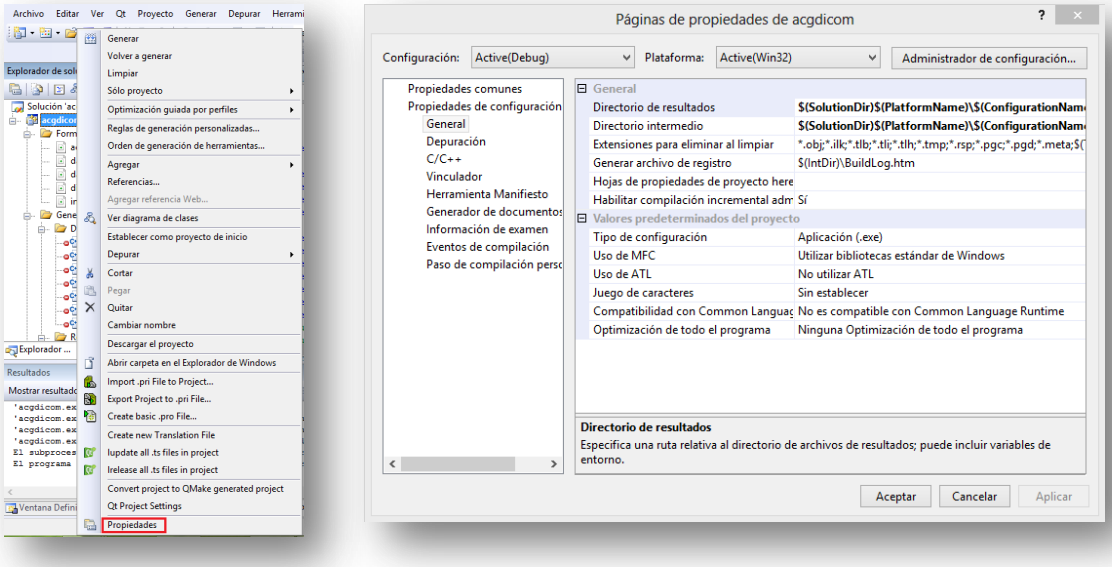


Figura 2.14: Propiedades del proyecto en Visual Studio 2008.

Las librerías se añaden en dos pasos que se detallan a continuación:

1. En primer lugar, se deben añadir los directorios de inclusión adicionales, para ello, en el menú de propiedades pulsamos la pestaña desplegable “C/C++” y escogemos la opción “General” (Fig. 2.15).

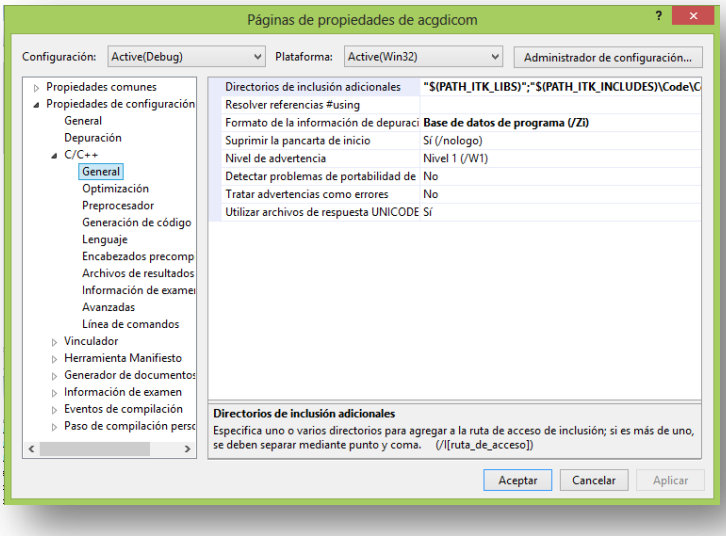


Figura 2.15: Pestaña “General” en propiedades del proyecto.

A continuación en “Directorios de inclusión adicionales” se deben añadir los siguientes:

- $(PATH_ITK_INCLUDES)\Code\Algorithms

- $(PATH_ITK_INCLUDES)\Code\BasicFilters
- $(PATH_ITK_INCLUDES)\Code\Common
- $(PATH_ITK_INCLUDES)\Code\IO
- $(PATH_ITK_INCLUDES)\Code\Numerics
- $(PATH_ITK_INCLUDES)\Code\Numerics\FEM
- $(PATH_ITK_INCLUDES)\Code\Numerics\NeuralNetworks
- $(PATH_ITK_INCLUDES)\Code\Numerics\Statistics
- $(PATH_ITK_INCLUDES)\Code\SpatialObject
- $(PATH_ITK_INCLUDES)\nifti\znzlib
- $(PATH_ITK_INCLUDES)\nifti\niftilib
- $(PATH_ITK_INCLUDES)\Utilities\vxl\core
- $(PATH_ITK_INCLUDES)\Utilities\vxl\vcl
- $(PATH_ITK_INCLUDES)\Utilities\vxl\v3p\netlib
- $(PATH_ITK_INCLUDES)\Utilities
- $(PATH_ITK_INCLUDES)\Utilities\DICOMParser
- $(PATH_ITK_INCLUDES)\Utilities\expat
- $(PATH_ITK_INCLUDES)\Utilities\gdcm\src
- $(PATH_ITK_INCLUDES)\Utilities\itkExtHdrs
- $(PATH_ITK_INCLUDES)\Utilities\NrrdIO
- $(PATH_ITK_INCLUDES)\Utilities\MetaIO
- $(PATH_ITK_LIBS)\Utilities
- $(PATH_ITK_LIBS)\Utilities\DICOMParser
- $(PATH_ITK_LIBS)\Utilities\expat
- $(PATH_ITK_LIBS)\Utilities\gdcm
- $(PATH_ITK_LIBS)\Utilities\NrrdIO

- $(PATH_ITK_LIBS)\Utilities\vxl\core
- $(PATH_ITK_LIBS)\Utilities\vxl\vcl
- $(PATH_ITK_LIBS)\Utilities\vxl\v3p\netlib

Nota: Las variables $(PATH_ITK_INCLUDES) y $(PATH_ITK_LIBS) hacen referencia a la ruta donde se encuentra el código fuente de las librerías ITK y las librerías compiladas, respectivamente. En el apartado siguiente, se explicará cómo crear dichas variables.

2. En segundo lugar, se deben añadir los directorios de bibliotecas adicionales, para ello, en el menú de propiedades pulsamos en la pestaña desplegable "Vinculador" (Fig. 2.16).

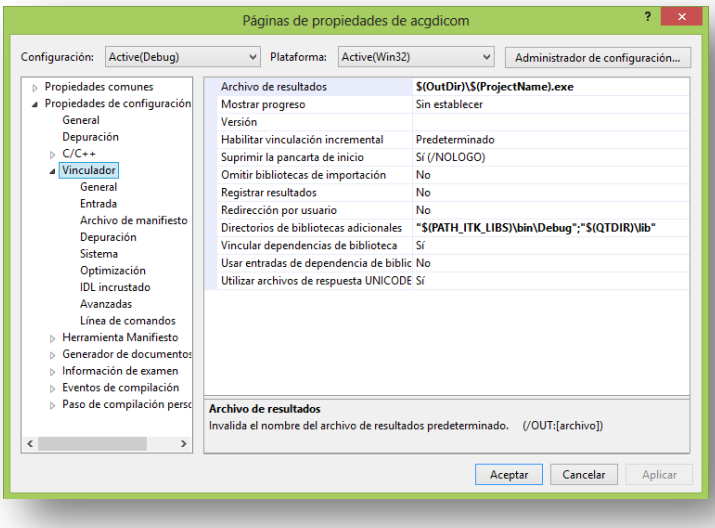


Figura 2.16: Pestaña "Vinculador" en propiedades del proyecto.

- En la opción "General": $(PATH_ITK_LIBS)\bin\Debug y $(PATH_ITK_LIBS)\bin\Release.

- En la opción "Entrada": ITKStatistics.lib, ITKFEM.lib, ITKAlgorithms.lib, itkNetlibSlatec.lib, ITKNumerics.lib,

ITKBasicFilters.lib, ITKIO.lib, ITKNrrdIO.lib, itkgdcm.lib, itkjpeg12.lib, itkjpeg16.lib, itkopenjpeg.lib, itkpng.lib, itktiff.lib, itkjpeg8.lib, ITKSpatialObject.lib, ITKCommon.lib, itkvnl_inst.lib, itkvnl_algo.lib, itkv3p_netlib.lib, itkvnl.lib, itkvcl.lib, ITKMetaIO.lib, itksys.lib, ITKDICOMParser.lib, ITKEXPAT.lib, ITKniftiio.lib, ITKznz.li, itkzlib.lib.

2.4.4 Creación de variables de entorno

Una variable de entorno es un valor que se almacena en memoria y tiene como objetivo simplificar el valor al que pertenece. Dicha variable, puede ser utilizada en varios procesos que funcionan simultáneamente. La utilidad que tienen en nuestro proyecto es la de simplificar la ruta absoluta donde se encuentran tanto el código fuente de las librerías ITK como las librerías ITK compiladas. Para crear estas variables debemos acceder al apartado "Editar las variables de entorno del sistema" en el Panel de Control de nuestro ordenador. En la pestaña "Opciones avanzadas" se pulsa el botón "Variables de entorno". A continuación en el cuadro de dialogo emergente, se escoge la opción "Nueva" y se procede a rellenar dos campos, en el superior se introduce el nombre de la variable y en el inferior su respectivo valor (Fig. 2.17).

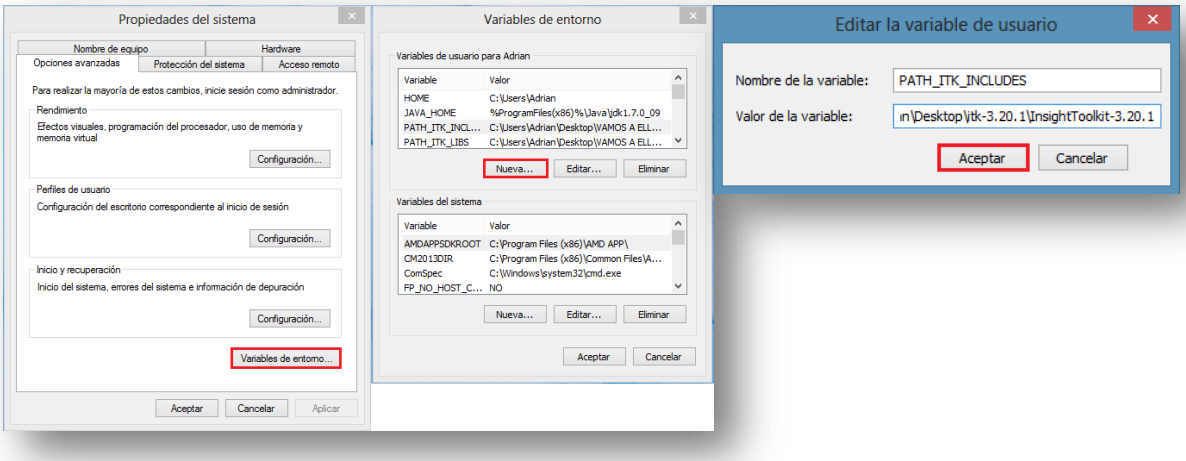


Figura 2.17: Pasos a seguir para crear una nueva variable de entorno.

Capítulo 3

Arquitectura de la plataforma

Una de las características principales que cumple la herramienta desarrollada es la de poseer una estructura totalmente modular. La interfaz de usuario, la lectura de imágenes DICOM y por descontado cada una de las funcionalidades de las que se ha provisto el visor, se han implementado en módulos independientes (Fig. 3.1). Este hecho ofrece una alta escalabilidad de la herramienta ya que en un futuro se pueden desarrollar *plugins* de filtrado, segmentación, detección automática y clasificación que contribuyan al diagnóstico basado en biomarcadores de imagen.

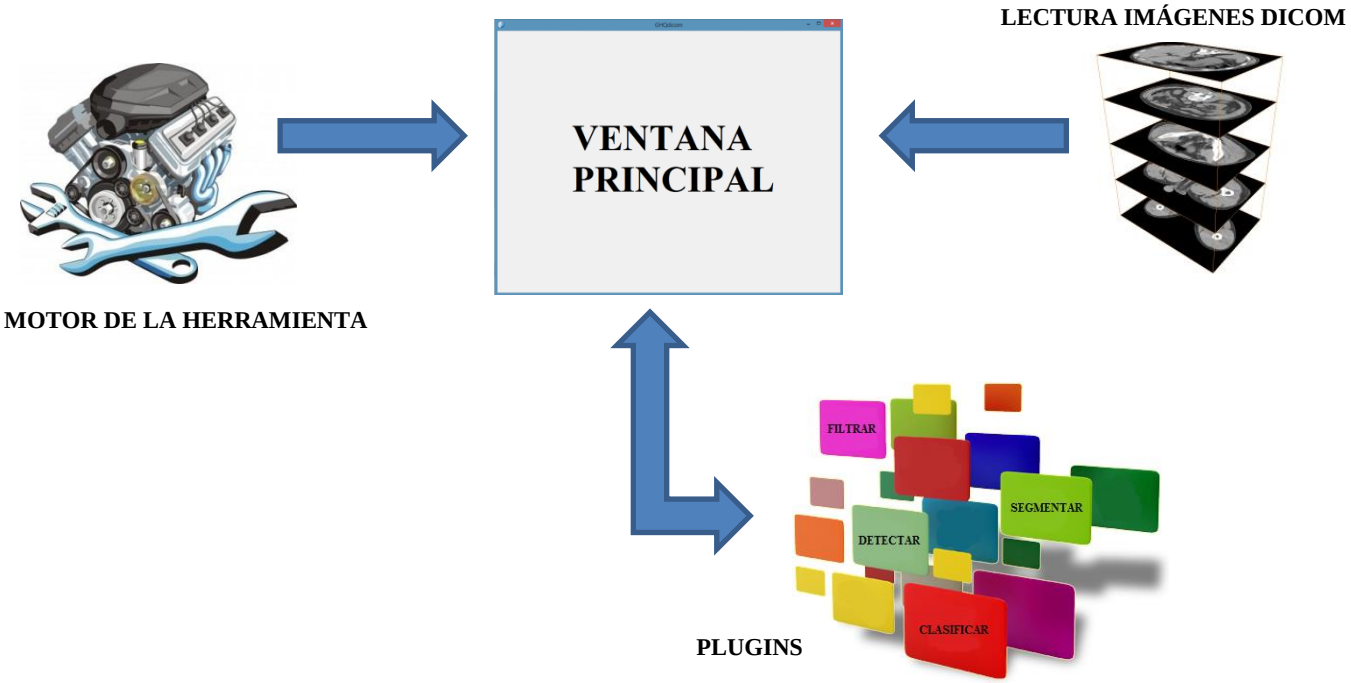


Figura 3.1: Esquema intuitivo de la arquitectura de la aplicación.

A continuación se muestra un esquema (Fig. 3.2) con las diferentes clases que componen la arquitectura de la aplicación. Como se puede observar, el motor de la aplicación genera una instancia de la ventana principal, a través de la cual se cargan y visualizan series de imágenes DICOM gracias a la comunicación con el módulo de lectura de imágenes. Una vez el usuario está visualizando la imagen, puede escoger diversas herramientas para trabajar en ella, algunas de estas herramientas requieren de comunicación con el usuario que se realiza por medio de ciertos cuadros de diálogo.

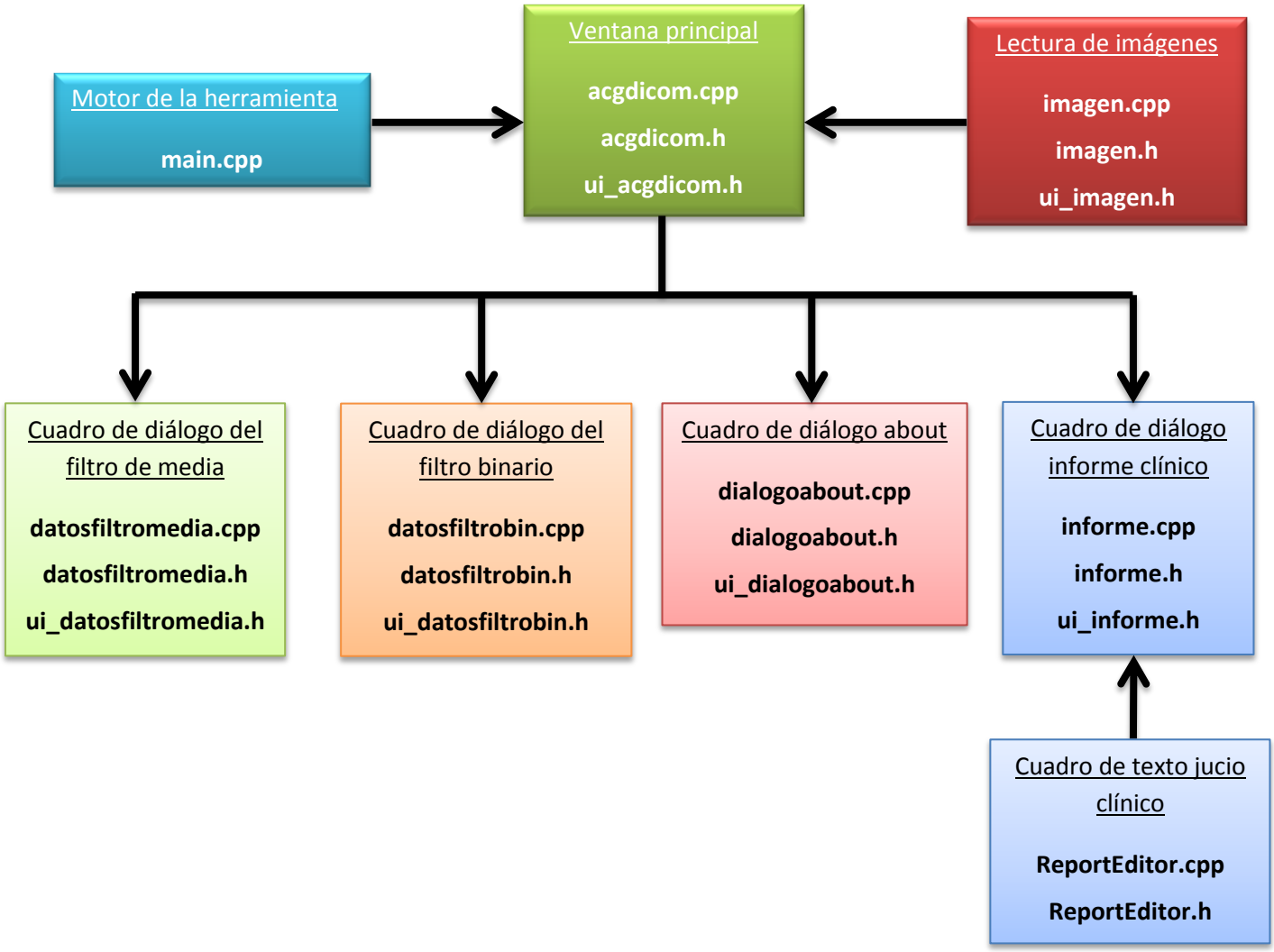


Figura 3.2: Esquema de clases que conforman la arquitectura de GHQdicom.

- **Motor de la herramienta**

El corazón de la aplicación desarrollada se encuentra en el archivo main.cpp. Dicho fichero (Fig. 3.3) contiene el código para inicializar la aplicación y los archivos de cabecera que contienen la lógica de la aplicación personalizada y los componentes de la IU (Interfaz de Usuario). Se realiza sólo una instancia por aplicación y debe crearse antes de cualquier objeto relacionado con la IU.

En primer lugar, se crea un objeto de la clase *QApplication* llamado "a". La clase *QApplication* administra el flujo de control de la aplicación de la IU. Dicha clase, contiene el bucle de eventos principal, donde todos los eventos del sistema de ventanas y otras fuentes son procesados y enviados. Asimismo se ocupa de la inicialización y finalización de la aplicación [12].

A continuación, se define la ventana principal del tipo acgdicom llamada "w", se muestra dicha ventana "w" y se ejecuta la aplicación "a". Por último, se hace una llamada al método *exec()* de la clase *QApplication* para iniciar el ciclo de eventos de Qt, pasándole de esta forma el control de la aplicación.

main.cpp

(Ámbito global)

```
#include "acgdicom.h"
#include <QtGui/QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    acgdicom w;
    w.show();
    return a.exec();
}
```

Figura 3.3: Archivo "main.cpp" de la aplicación.

- **Lectura de imágenes**

La clase Imagen se ha creado con la finalidad de visualizar las imágenes DICOM. En la ventana principal se crea una instancia de esta clase en un elemento *QFrame*, estableciendo así, un marco o *frame* en el que visualizar las imágenes (En el capítulo posterior, se explica detalladamente, el método de creación de *widgets* propios para insertar cualquier elemento, heredando las propiedades de una clase implementada por el programador).

Para visualizar una imagen DICOM se deben seguir los siguientes pasos:

1. Leer la imagen DICOM mediante las librerías ITK:

Para leer la imagen se recorrerán todos los píxeles que la constituyen y sus valores se almacenarán en un buffer, al cual tendremos acceso mediante un puntero creado en la clase Imagen. Este paso está ampliamente explicado en [13].

2. Convertir la imagen de ITK a Qt:

A partir del puntero que tiene acceso a la imagen, se realiza una conversión de ITK a un objeto de la clase *QImage*. Esta clase, es la utilizada en Qt para dibujar imágenes.

3. Dibujar y visualizar la imagen en la interfaz gráfica de usuario:

Para dibujar los objetos de tipo *QImage* se ha asociado un evento llamado *paintEvent* a la clase Imagen que automáticamente dibuja una imagen dentro de una especie de marco que las contiene. Cada vez que se ejecuta la instrucción *uptade()* se llama al método *paintEvent()*.

En la clase Imagen, también se implementan funcionalidades como la transición de los diferentes *frames* que componen una serie DICOM mediante el movimiento de la bola central del ratón.

- **Ventana principal**

La clase acgdicom es el cerebro de la aplicación desarrollada. En "acgdicom.ui" se encuentra la interfaz de usuario de la ventana principal provista de todos los *widgets* necesarios, mientras que en el archivo "acgdicom.cpp" se encuentran desarrolladas las principales funcionalidades de las que está provisto el visor de imagen médica implementado. Dichas funcionalidades, son detalladas a nivel de usuario y programador en el apartado posterior.

- **Cuadros de dialogo**

Una forma rápida y cómoda de comunicación entre la aplicación y el usuario es trabajar con cuadros de dialogo. Son diversas las acciones y herramientas que precisan de información que debe ser proporcionada por el usuario. En el capítulo posterior, en el apartado en el que se detallan las funcionalidades del software, también se hará lo propio con todos los cuadros de dialogo implementados explicando tanto la interfaz de usuario como la utilidad y datos requeridos por cada uno de ellos.

Capítulo 4

Desarrollo e implementación de GHQdicom

4.1 Crear un nuevo proyecto en Visual Studio 2008

Una vez se lanza Visual Studio 2008, para la creación de un nuevo proyecto, se debe escoger la opción "Archivo" desplegar la pestaña "Nuevo" y hacer *click* en "Proyecto…". Instantáneamente emergerá un cuadro de dialogo, en el cual se debe seleccionar como tipo de proyecto "Qt4 Projects" y como plantilla "Qt Application". Se establecerá un nombre para el nuevo proyecto y la ubicación donde se desea guardar el mismo (Fig.4.1). Al pulsar el botón "Aceptar" se iniciará un asistente Qt que nos guiará hasta finalizar el proceso.

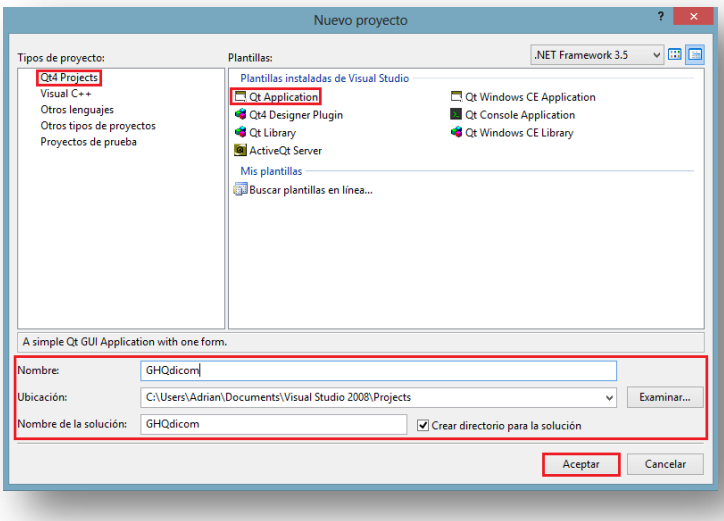


Figura 4.1: Nuevo proyecto en Visual Studio 2008.

El asistente se compone de tres pasos (Fig. 4.2), un primer paso de introducción, un segundo paso donde es posible incluir diversos tipos de librerías (aunque es recomendable dejar las opciones por defecto) y un tercer y último paso en el cual podemos cambiar el nombre por defecto con el que se generan los archivos y especificar la clase base a partir de la cual se crea el proyecto. Cabe destacar que escogiendo como clase base *QMainWindow* (opción por defecto) se obtiene una interfaz inicial que posee una barra de menú, una barra de herramientas y una barra de estado que son de gran utilidad para el desarrollo de la herramienta.

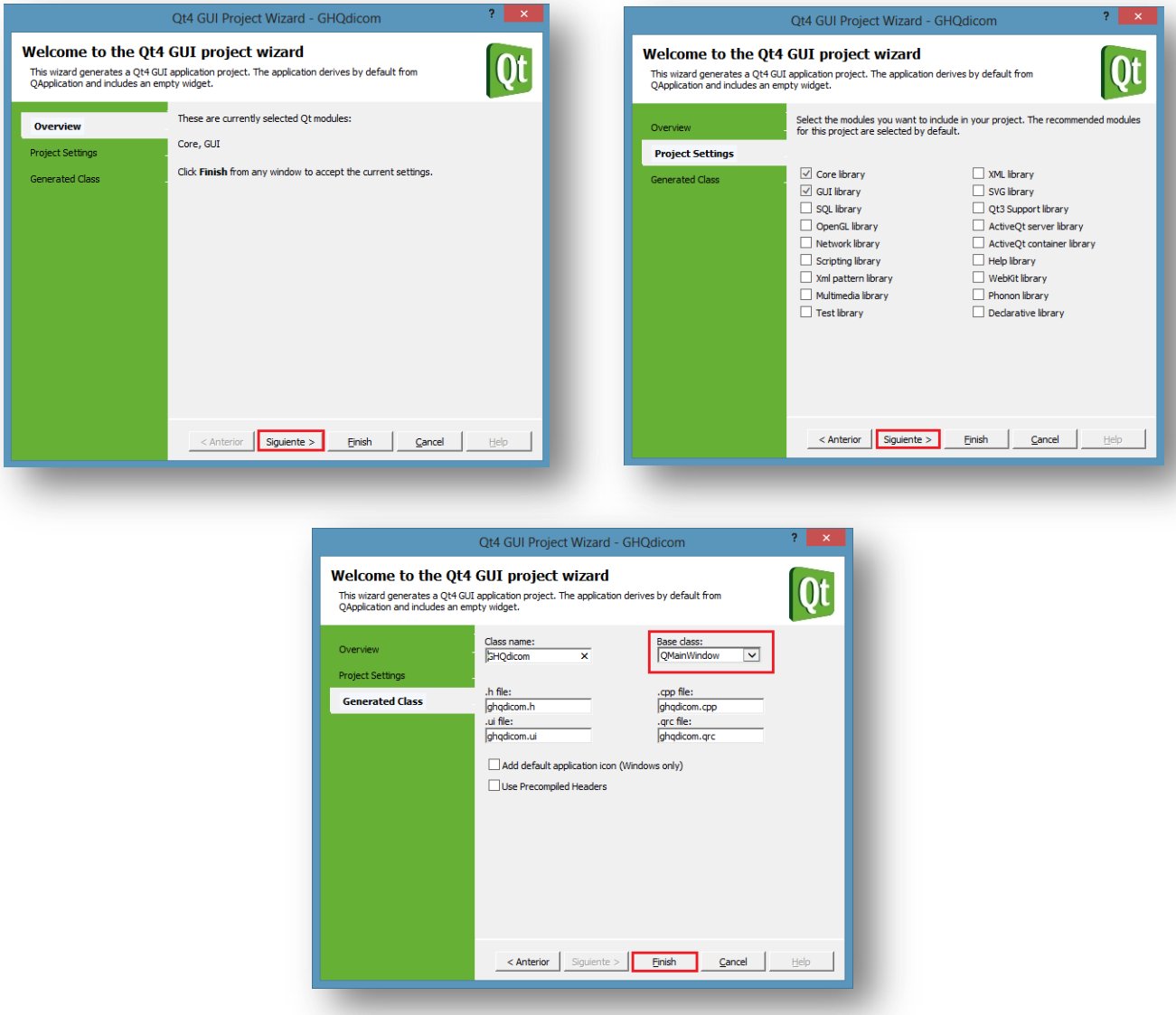


Figura 4.2: Asistente de creación de un proyecto Qt en Visual Studio 2008.

Al pulsar el botón “Finish” en el último paso del asistente, se habrá creado el proyecto exitosamente (esto se puede comprobar en la barra de estado de Visual Studio 2008). Además en el “explorador de soluciones” (ventana izquierda del entorno de trabajo), se pueden observar los ficheros que se han generado (Fig. 4.3).

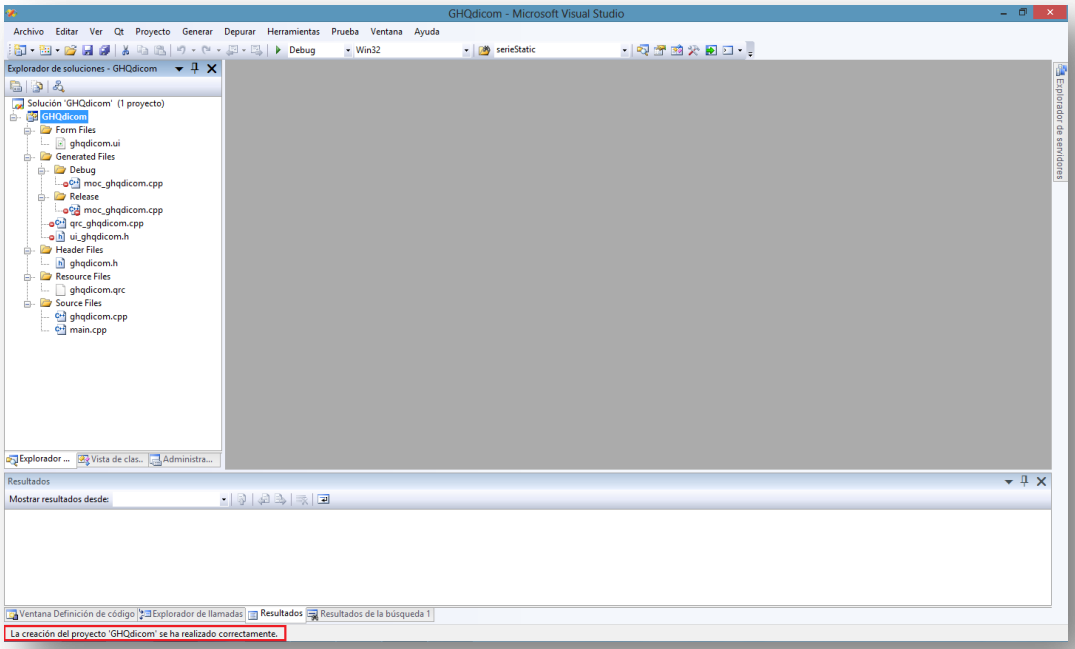


Figura 4.3: Proyecto creado correctamente en Visual Studio.

A continuación se detalla el contenido de los ficheros que se han generado automáticamente (Fig. 4.4):

- **ghqdicom.h**: Los archivos con extensión “.h” es el lugar donde se declaran las clases. Estos ficheros, contienen la declaración de variables globales y la definición de cabecera de todas las funciones que componen la clase.
- **ghqdicom.cpp**: Los archivos con extensión “.cpp” es el lugar donde se desarrollan las funciones de cierta clase.
- **ghdicom.ui**: Este tipo de archivo es utilizado mediante la aplicación *Qt Designer* para desarrollar la interfaz de usuario. Dicho software se detalla en el

apartado 4.3. Es importante destacar que los cambios que se efectúen mediante *Qt Designer*, se verán reflejados automáticamente en el fichero "ui_ghqdicom.h" mediante el código fuente pertinente.

- **main.cpp**: Por defecto, se genera también el fichero principal de la aplicación, si se abre, se puede observar como en él se crea una instancia del objeto de la clase principal y se lanza mediante la instrucción *show()*.
- El resto de ficheros que se generan son únicamente necesarios para la compilación del proyecto, por tanto, ajenos al programador.

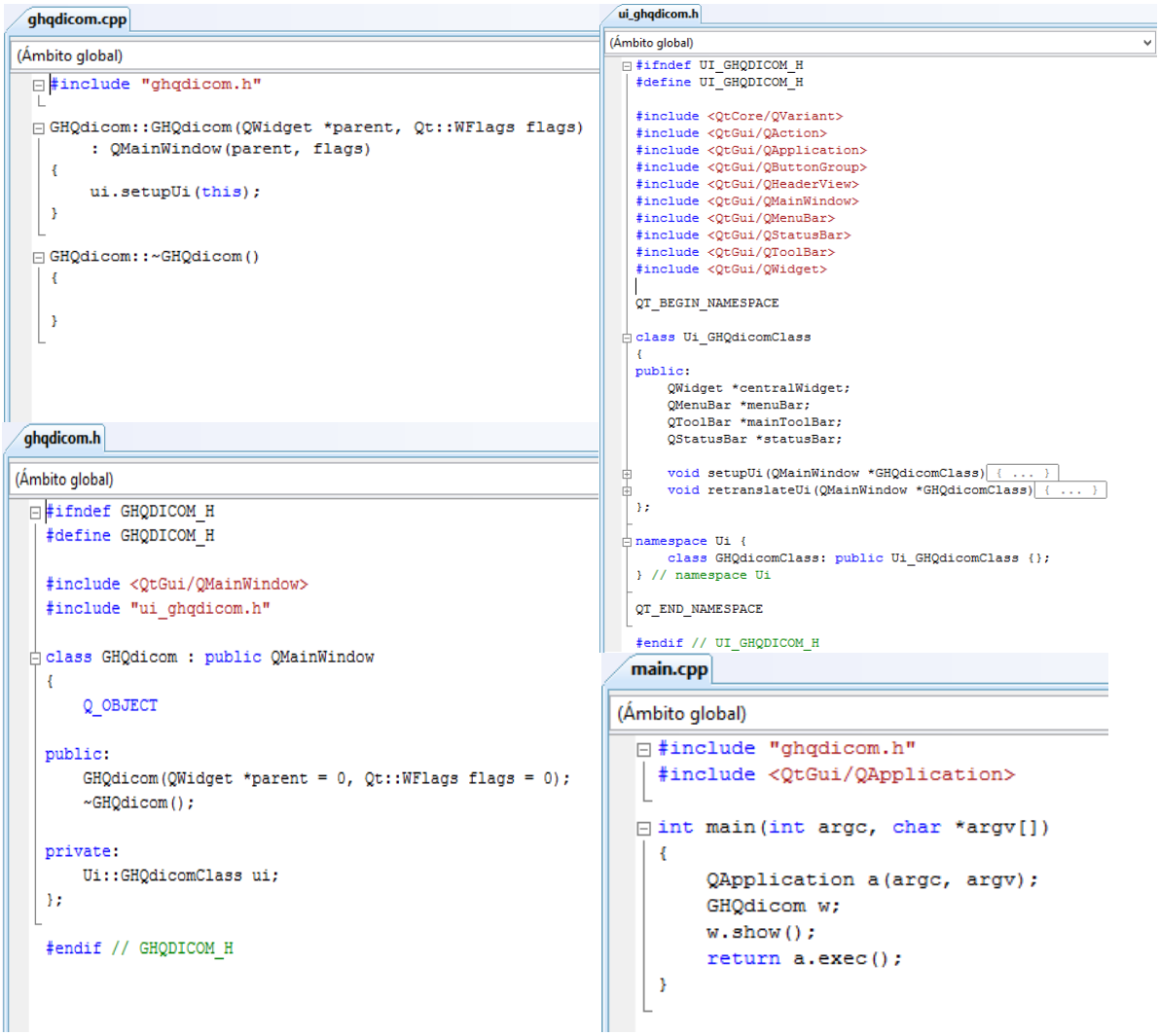


Figura 4.4: Contenido de los ficheros generados al crear un nuevo proyecto.

4.2 Crear una nueva clase en el proyecto

Para crear una nueva clase en el proyecto de Visual Studio 2008 se deben seguir los siguientes pasos: En primer lugar, se presiona botón derecho sobre el nombre del proyecto, se despliega la pestaña “Agregar” y se escoge la opción “Clase…” (Fig. 4.5).

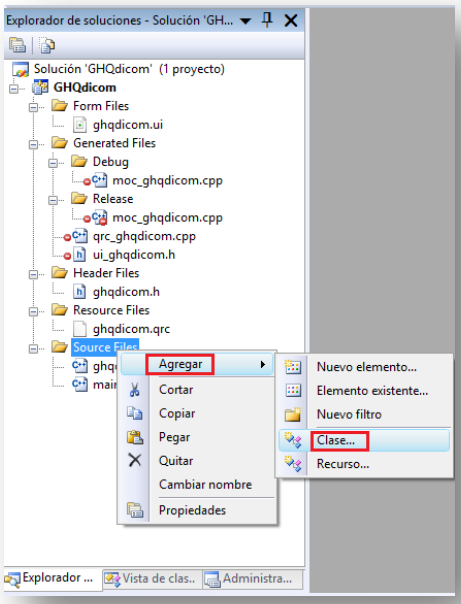


Figura 4.5: Primer paso para crear una nueva clase en el proyecto.

Una vez se abra el cuadro de dialogo se debe seleccionar como categoría *Qt4 Classes* y como plantilla *Qt4GuiClass* pulsando seguidamente el botón “Agregar” (Fig. 4.6).

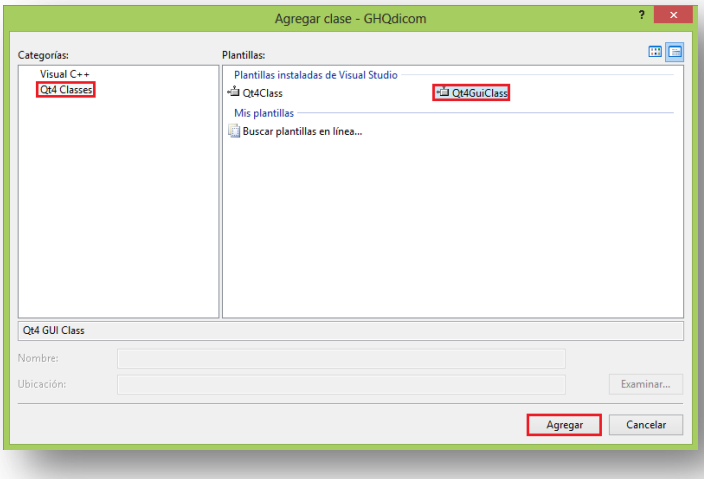


Figura 4.6: Cuadro de dialogo para crear un nueva clase en el proyecto.

Finalmente, solo queda rellenar el campo *Class Name* (automáticamente asigna dicho nombre a todos los ficheros) y *Base Class* en la nueva pantalla que aparece (Fig. 4.7). Las clases usuales que se asignan como clase base son: *QDialog* (si se quiere crear un cuadro de dialogo) o *QWidget* (si se quiere crear un elemento para la interfaz de usuario). No tendría sentido asignar como clase base a esta nueva clase *QMainWindow* puesto que es la que se debe seleccionar al crear el proyecto como se ha visto en el apartado anterior.

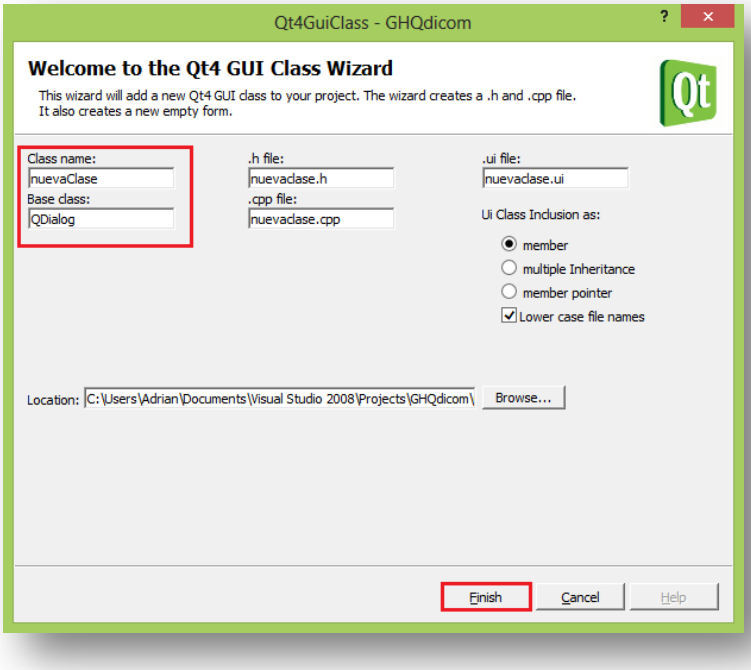


Figura 4.7: Asistente Qt para la creación de una nueva clase.

4.3 Implementación de la interfaz de usuario

Como se ha comentado en el apartado 4.1, la interfaz de usuario se puede desarrollar por el método convencional, escribiendo las instrucciones necesarias para implementar los elementos que la conforman. En este proyecto, se ha optado por una

opción más intuitiva y manejable, se desarrolla la interfaz de usuario mediante una herramienta que proporciona Qt para este propósito: *Qt Designer* (Fig.4.8). Gracias a esta herramienta es posible cambiar las dimensiones y propiedades de los elementos que conforman la IU (interfaz de usuario) mediante un solo *click*.

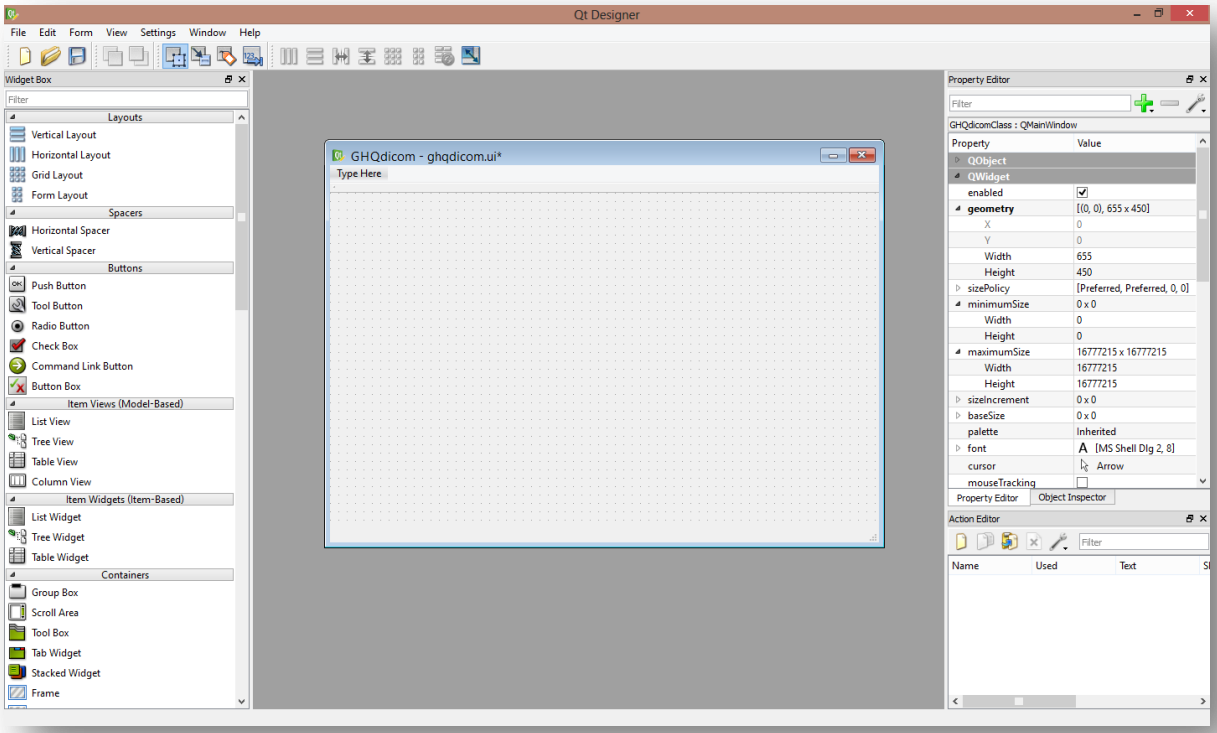


Figura 4.8: Pantalla principal de Qt Designer.

4.3.1 *Widgets* con Qt Designer

Para comenzar a utilizar esta herramienta, cabe destacar, que los elementos que se pueden introducir en la IU se denominan *widgets* y se encuentran en la *Widget Box* (ventana izquierda en *Qt Designer*). Se dispone de *widgets* espaciadores, *layouts*, botones, contenedores y *widgets* de entrada y salida de datos entre otros.

La forma de ir implementando la IU, es haciendo *click* sobre un determinado *widget* y arrastrarlo hasta el lugar deseado de la ventana de nuestra aplicación (*QMainWindow*). Después de realizar esta acción, en la ventana *Property Editor*

(ventana derecha superior de *Qt Designer*), aparecerá el nombre del *widget* insertado con una serie de propiedades desplegables que se modificarán según las necesidades y el gusto del programador (Fig. 4.9). Para todos los ejemplos sobre *widgets* que aparecen en este capítulo, conviene remarcar que existen múltiples propiedades que se podrían aplicar y para ello, se recomienda consultar el asistente de Qt que proporciona el software, junto con la referencia [11].

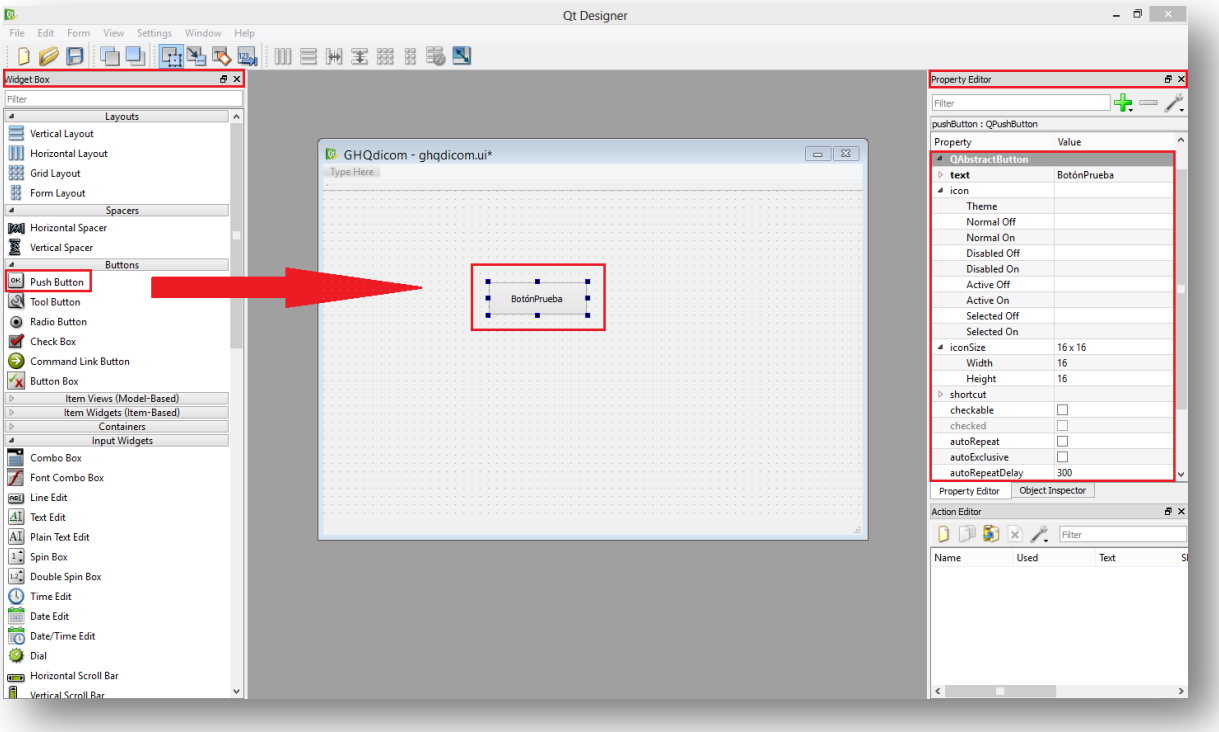


Figura 4.9: Inserción de un widget y visualización de sus propiedades.

Como se ha comentado en este capítulo las creaciones y modificaciones que se hagan en *Qt Designer* se verán reflejadas en el archivo "ui_nombreClase.h" en forma de código fuente (modificable por el programador). En la figura 4.9, se puede observar la inserción de un botón en la ventana principal de la aplicación que se refleja en "ui_ghqdicom.h" de la siguiente forma:

```
pushButton = new QPushButton(centralWidget);                    (1)
```

```
pushButton->setObjectName(QString::fromUtf8("pushButton"));        (2)
pushButton->setGeometry(QRect(200, 110, 121, 41));                 (3)
GHQdicomClass->setCentralWidget(centralWidget);                    (4)
```

La instrucción (1) crea una instancia del botón, posteriormente con *setObjectName()* y mediante una cadena de tipo *QString* se asigna el nombre del componente. La sentencia (3) establece tanto la situación como las dimensiones del botón. Por último con *setCentralWidget()*, introducimos el elemento en el *widget* central de nuestra aplicación.

Existe también la posibilidad de crear *widgets* propios, es decir, crear un elemento que reúna todas las características de cierto objeto de una clase desarrollada por el programador, esta opción será esencial para la inserción de módulos creados en un futuro. Para configurar *widgets* propios, en primer lugar, en *Qt Designer* se arrastra a la ventana principal de nuestra aplicación, un objeto de la clase base de la que hereda nuestra clase propia. Una vez dispongamos de dicho elemento, se hace *click* derecho sobre él y se escoge la opción "Promote to…" (Fig. 4.10).

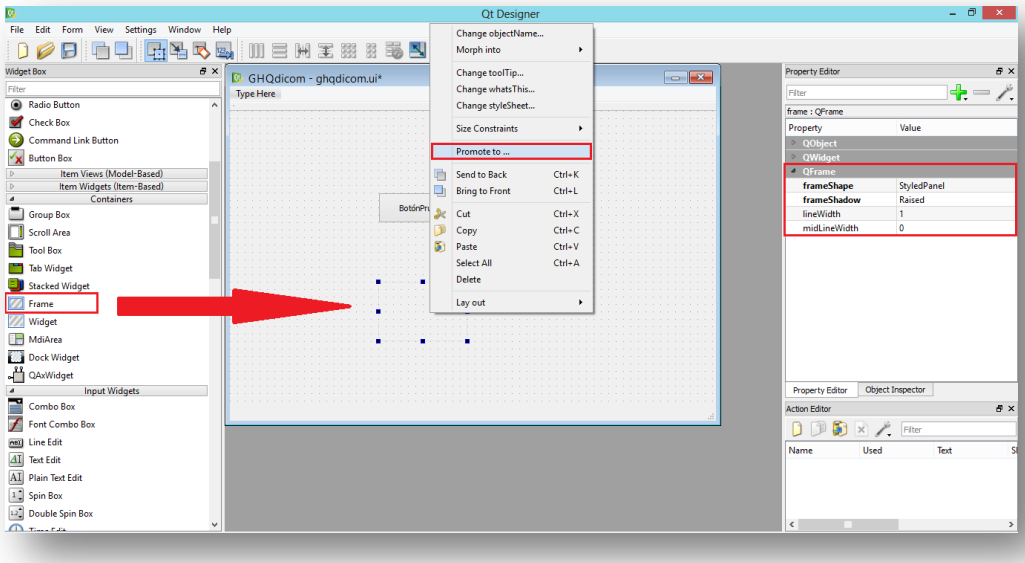


Figura 4.10: Pasos para la creación de un widget propio.

Posteriormente, en el cuadro de diálogo emergente (Fig. 4.11), se pueden observar tres campos. El primero de ellos aparece automáticamente con la información del *widget* que hemos arrastrado (de la misma clase que la clase de la que hereda nuestra clase propia), en el ejemplo se ha arrastrado un objeto *QFrame*, esto se puede observar en el campo *Base class name*. En el segundo campo, *Promoted class name*, se debe indicar el nombre de la clase propia a partir la cual se quiere crear el *widget*. El último campo, *Header File*, se completará automáticamente con el fichero ".h" de la clase indicada arriba. A continuación se pulsa el botón "Add" y para finalizar el botón "Promote".

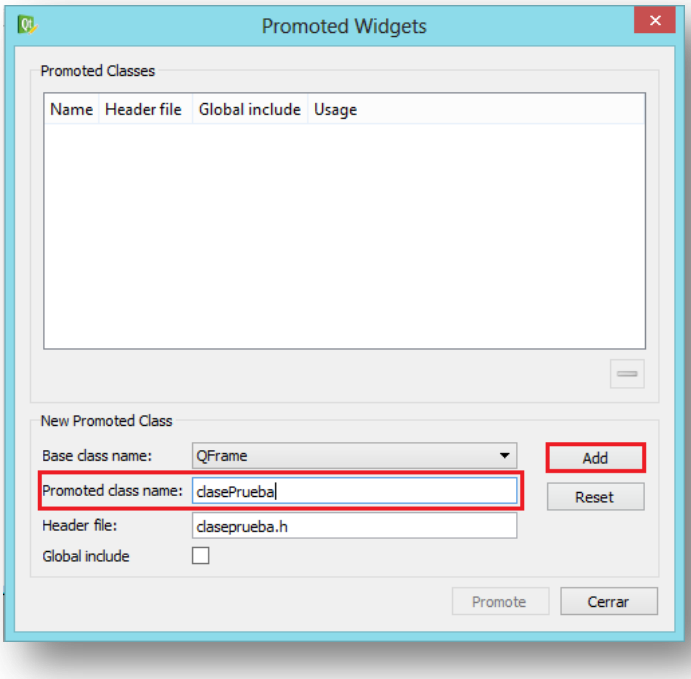


Figura 4.11: Cuadro de diálogo *Promoted Widgets*.

4.3.2 Barra de menú y barra de herramientas con Qt Designer

Como se ha comentado en un apartado anterior, cuando se crea un proyecto Qt en Visual Studio 2008 y se escoge como clase base *QMainWindow*, se incluye por defecto una barra de menú, una barra de herramientas y una barra de estado. En este

apartado se va a explicar la forma de crear menús y submenús en la barra de menú y acciones en la barra de herramientas.

Mediante *Qt Designer*, se observa en la barra de menú, un botón denominado "Type Here", pulsando doble *click* sobre él, se habilita la escritura para establecer el nombre deseado, creando de esta forma un primer menú. Como se puede observar en la figura 4.12, se puede ir formando un nivel jerárquico, es decir, dentro de un menú se puede crear una pestaña y establecerla como submenú mediante el icono . Si en el interior de un menú se crea una pestaña que no se establece como submenú, esta pestaña será una acción, es decir, en la aplicación que estamos desarrollando, cuando se pulse dicha pestaña, se producirá una respuesta. Las acciones que se van creando se agrupan por nombre en la ventana *Action Editor* de *Qt Designer*. Desde dicha ventana, también es posible crear una nueva acción, asignarle un nombre y un icono, y arrastrarla tanto a la barra de menú como a la barra de herramientas, conformando de esta forma nuestras dos barras, según las necesidades de la aplicación a desarrollar. Cabe destacar a su vez que las propiedades de determinada acción se pueden modificar en la pestaña *QAction* de la ventana *Property Editor*.

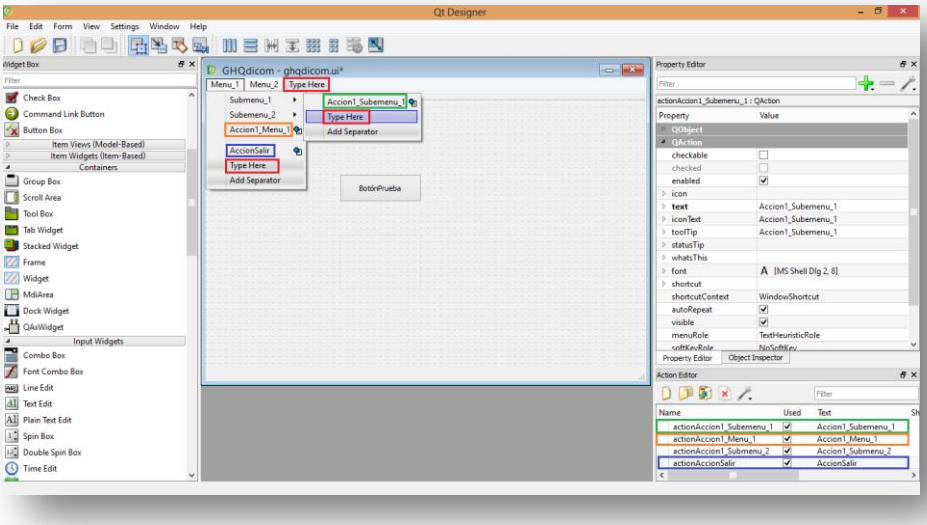


Figura 4.12: Creación de menús, submenús y acciones mediante Qt Designer.

4.3.3 *Signals* y *Slots*

Las acciones del usuario sobre los *widgets* generan eventos que el programa puede responder normalmente ejecutando una o más funciones. Por ejemplo, cuando un usuario aprieta o hace *click* en un *widget*, un evento *MousePress* y un *MouseRelease* se generan.

Los *widgets* emiten señales llamadas *signals*, para indicar que se ha producido una acción del usuario o ha ocurrido un cambio de estado. Por ejemplo, cuando el usuario aprieta un botón perteneciente a la clase *QPushButton*, éste emite una señal llamada *clicked()*. Serían también ejemplos de *signals*, elegir una opción de la barra de menú o de la barra de herramientas, abrir o cerrar una ventana, etc.

Una señal se puede conectar a una función llamada *slot*. Así pues, cuando la señal es emitida, el *slot* es ejecutado automáticamente. Normalmente, la manera de hacerlo sería:

connect(widgetGeneradorEvento, SIGNAL( ), widgetAfectaEvento, SLOT( ))

donde *signal* y *slot* corresponden a la señal que emite el *widget* y a la función encargada de tratar dicha señal respectivamente.

Un ejemplo adaptado al entorno de trabajo disponible en la figura 4.12 es el siguiente: connect(ui.AccionSalir, SIGNAL(triggered()), this, SLOT(close()); con esta línea de código cuando el usuario escoja la opción AccionSalir la aplicación se cerrará.

Cabe destacar la posibilidad de poder conectar una o varias señales a un único *slot* y también una o varias señales a varios *slots*.

4.3.4 Resultado de la interfaz de usuario de GHQdicom

La interfaz de usuario de GHQdicom (Fig. 4.13) está compuesta por diferentes menús (Archivo, Ver, Imagen, Filtros, Ayuda). En la barra de herramientas existen

ocho botones de acción, cuando el usuario pulsa uno de ellos, recibe cierta respuesta. El *widget* central está compuesto por tres *labels* (que contendrán información acerca del paciente, de la adquisición y de la imagen que se está visualizando) y un marco del tipo Imagen (clase propia) que hereda de *QImage*, en el que se visualizarán las imágenes. Estos cuatro elementos se encuentran en un *layout* que los organiza convenientemente. A la derecha del este conjunto, se encuentran dos *scrollBars* verticales, abajo un *scrollBar* horizontal y arriba tres botones, un *label* y un *spinBox* que conforman el modo cine de la aplicación.

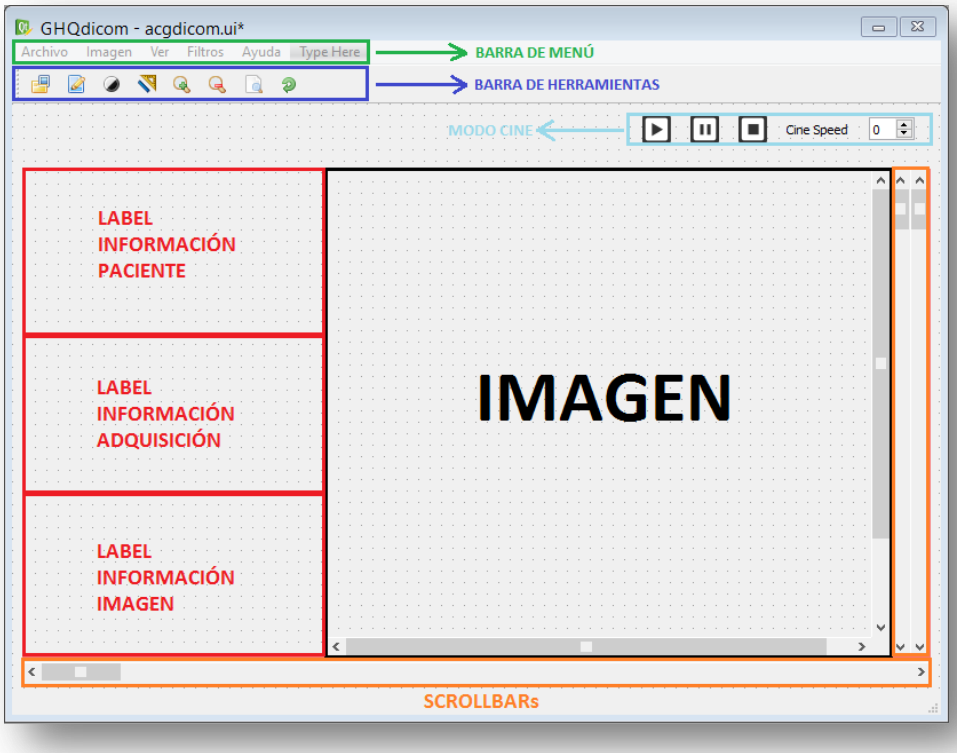


Figura 4.13: Interfaz de usuario de GHQdicom.

4.4 Funcionalidades de la herramienta

Después de haber visto la arquitectura de la aplicación desarrollada, vamos a centrarnos en describir las utilidades de la herramienta, desarrolladas en la ventana principal (clase acgdicom).

- **Visualización correcta y funcional de las diferentes secuencias de imágenes**

Las series estáticas, están compuestas por diferentes cortes de cierto volumen del paciente. Se toma el primer corte como referencia y los siguientes se van adquiriendo con cierto incremento longitudinal constante.

Las series denominadas dinámicas, como su nombre indica, están compuestas por diversos *frames* temporales de un determinado corte de cierto volumen de un paciente. Nuevamente se toma el primer corte como referencia y los posteriores se van adquiriendo con un incremento longitudinal constante. Se adquiere el mismo número de *frames* temporales para todos los cortes. Dicho tipo de series es común en estudios encargados de visualizar procesos fisiológicos que varían a lo largo del tiempo (difusión y perfusión).

Uno de los objetivos de este proyecto es el de visualizar funcionalmente las distintas secuencias de imágenes. Las series estáticas se visualizan por medio de un único slider vertical, mientras que para las dinámicas se utiliza el slider vertical para barrer los distintos cortes adquiridos y mediante el horizontal se visualizan los *frames*, de diferentes instantes temporales, del corte seleccionado. Para dotar a nuestro visor de esta utilidad, se han desarrollado dos *slots* (*moverSerieTiempo()* y *moverSerieCortes()*) que responden con diferentes acciones (dependiendo el tipo de serie) a los eventos que se producen al variar el valor de los sliders.

Para discriminar entre serie estática y dinámica se lee el campo *Slice Location* de la cabecera de cada imagen DICOM. Dicho campo, contiene el corte del volumen al que pertenece la imagen. Si en una determinada serie, ninguna de las imágenes tienen el mismo valor en dicho campo, significará que cada imagen pertenece a un corte, o lo que es lo mismo, se tratará de una serie estática. Si por el contrario, en una serie

tenemos imágenes con el mismo valor en el campo *Slice Location* sabemos que en cada corte tenemos más de un *frame* y por tanto se dispone de una serie dinámica. El algoritmo implementado se encuentra en *serieStatic().*

Cabe destacar que cuando se carga una serie *dinámica* necesitamos acceder a otra etiqueta de la cabecera DICOM para poder ordenar los *frames* de cada corte en el tiempo, puesto que estos no se leen en orden. La etiqueta necesaria se denomina *Instance Number* y nos informa sobre el instante temporal que ocupa la imagen en la serie. Se ha desarrollado un algoritmo de ordenación para poder visualizar las imágenes en el orden correcto (Dicho algoritmo lo componen los métodos *ordenaInstNum_SliceLoc()* y *posMinimo()*).

- **Mostrar datos del paciente y de la adquisición de imágenes**

Para el facultativo que analiza un caso también es importante conocer de un simple vistazo ciertos datos sobre el paciente, la técnica de adquisición y la imagen que está visualizando. Por este motivo, en la ventana principal, al lado izquierdo de la imagen, existen tres cuadros de texto que muestran toda esta información. Para ello, lo que se hace es acceder a los metadatos de la cabecera de la imagen DICOM, mediante las librerías ITK (se detalla en el siguiente apartado), almacenarlos y mostrarlos por pantalla. Las rutinas donde se van almacenando estos valores son *vectorSliceLocation()* y *tagsGeneralesDicom()*. Hay que destacar también la capacidad del software para ocultar toda esta información mediante un click en caso de que se requiera mantener en anonimato ciertos datos. A continuación, quedan reflejados los campos que se muestran en cada cuadro de texto, así como su posición en hexadecimal dentro de la cabecera DICOM.

DATOS DEL PACIENTE		
Nombre del tag DICOM	**Descripción**	**Posición dentro de la cabecera**
Patient Name	Nombre del paciente	0010 | 0010
Patient Birth Data	Fecha de nacimiento del paciente	0010 | 0030
Patient ID	Número ID del paciente	0010 | 0020
Patient Sex	Sexo del paciente	0010 | 0040
Patient Weigth	Peso del paciente	0010 | 1030

Tabla 4.1: Datos sobre el paciente que se muestran en la aplicación.

DATOS DE LA TÉCNICA DE ADQUISICIÓN		
Nombre del tag DICOM	**Descripción**	**Posición dentro de la cabecera**
Acquisition Date	Fecha de adquisición de las imágenes	0008 | 0022
Acquisition Time	Hora de adquisición de las imágenes	0008 | 0032
Institution Name	Nombre del hospital	0008 | 0080
Study ID	ID del estudio	0020 | 0010
Study Description	Descripción del estudio	0008 | 1030
Manufacturer	Fabricante del equipo de adquisición	0008 | 0070
Series Description	Descripción de la serie	0008 | 103e
Series Number	Número identificador de la serie	0020 | 0011
Slice Thickness	Espesor del corte (mm)	0018 | 0050
Pixel Spacing	Distancia (en x e y) a la que corresponde un pixel	0028 | 0030
Echo Time	Tiempo de eco	0018 | 0081
Flip Angle	Ángulo de excitación	0018 | 1314
Repetition Time	Repetición del pulso	0018 | 0080

Tabla 4.2: Datos referentes a la técnica de adquisición que se muestran en la aplicación.

DATOS DE LA IMÁGEN		
Nombre del tag DICOM	**Descripción**	**Posición dentro de la cabecera**
Slice Location	Corte	0020 | 1041
Instance Number	Establece orden temporal	0020 | 0013
Window Center	Centro de la ventana	0028 | 1050
Window Width	Ancho de la ventana	0028 | 1051

Tabla 4.3: Datos de la imagen que se utilizan y se muestran en la aplicación.

- **Visualización modo cine con ajuste de velocidad del *frame***

Para una perfecta visualización de la secuencia que se esté analizando, es de vital importancia la transición automática de *frames*, creando de esta forma un video que ayude a determinar ciertas características. En la aplicación desarrollada esta utilidad ha sido implementada con ayuda de las Qt.

Mediante la clase *QTimer* de las librerías Qt, en el momento que el usuario presiona el botón *play*, se crea un contador descendente en milisegundos. Cuando dicha cuenta atrás finaliza, el slider avanza automáticamente una posición, creando de esta forma la transición del *frame*. Variando el número de milisegundos de la cuenta atrás varía la velocidad de transición de los *frames*. Si el usuario decide parar el pase automático de imágenes presionando el botón *pause*, se detiene el contador mediante la instrucción *timer->stop()*. Si se desea volver al comienzo de la secuencia basta con presionar el botón *stop* para que se ejecute la orden *ui.verticalSlider->setValue(0)*. Los métodos de la clase acgdicom que posibilitan esta utilidad son *botonPlay()*, *siguienteFrame()*, *botonPause()*, *botonStop()* y *valorSpinBox()*.

- **Variación del brillo y contraste de la imagen**

Otra de las utilidades indispensables que debe poseer un visor de imagen médica es la posibilidad de variar el brillo y contraste ya que en algunas ocasiones facilita la detección de una determinada lesión. El modo de implementarlo en esta herramienta es muy manejable para el usuario, puesto que al activar el botón de brillo/contraste, situando el cursor en la imagen, haciendo *click* y variándolo de arriba abajo y de izquierda a derecha varían brillo y contraste simultáneamente. Para ello, lo que se hace es ir variando la intensidad de los pixels modificando los parámetros ventana y centro por medio del movimiento del ratón, según la expresión:

$$\boldsymbol{valorPixels'} = \frac{\boldsymbol{valorPixels - centro + ventana/2}}{\boldsymbol{ventana}}$$

En la función *movRaton()* se implementa el cálculo de centro y ventana según las coordenadas del puntero. Con este valor de centro y ventana se llama al método *loadSeriesImage()* que se encarga de calcular el nuevo valor de intensidad de los *pixels* para dibujar el *frame* con la correspondiente variación, llamando a la subrutina *paintEvent()* por medio de la instrucción *update()*.

- **Zoom a la imagen y vuelta a las dimensiones originales**

Se incluye también en la herramienta desarrollada la capacidad para hacer zoom sobre la imagen que se está visualizando. Cuando el usuario presiona los botones de zoom, se aplica una escala a la imagen. En el caso de aumentar el zoom (función *ZoomIn()*), las dimensiones de la imagen se multiplican por un factor de escala de 1.25,

en el caso de disminuir el zoom (*zoomOut()*) dicho factor es de 0.8. Con estos factores aseguramos siempre los mismos rangos de visualización ya que 0.8*1.25 = 1. La subrutina encargada de realizar la conversión de dimensiones es *scaleImage()*. Se ha implementado además el método *normalSize()*, que permite que pulsando un botón, el usuario restablezca el tamaño original de la imagen. Otra propiedad destacada es la aparición de dos sliders cuando se hace mucho zoom sobre la imagen, para poder recorrerla visualmente sin ningún problema.

- **Medidas sobre la imagen**

En muchas ocasiones, el usuario de la herramienta precisará de una herramienta de medida para conocer distancias reales o dimensiones de ciertas lesiones. Es por ello que el software implementado incluye una regla. El usuario hace *click* en un punto y traza una recta hasta un segundo punto donde suelta el botón del ratón. Para desarrollar esta funcionalidad, se ha desarrollado en primer lugar el *slot iniRegla()* que responde a la señal *MousePressed()*. En dicho slot, se inicia un contador para saber en qué momento comienza la medida. Mientras el usuario está moviendo el puntero buscando el segundo punto de la línea, se va refrescando el trazo gracias a la función *movRegla()*, que responde al evento *MouseMoveEvent()* y se comunica con la clase imagen para representar la línea sobre el dibujo. Cuando se suelta el botón del ratón, se produce el evento *MouseReleaseEvent()* que se capta en *finRegla()*, donde se para el contador, se dibuja la línea final y se calcula la distancia gracias a que conocemos el desplazamiento en la dirección x e y desde el punto inicial hasta el punto final. Por último, se convierte la distancia de la imagen a distancia real gracias al campo *PixelSpacing* de la cabecera DICOM.

- **Transición de *frames* mediante la bola de ratón**

Aparte de navegar por los distintos *frames* mediante los *sliders* o el modo cine, también existe la posibilidad de hacerlo mediante la bola del ratón. Se ha desarrollado el método *movBola()* que capta un evento llamado *wheelEvent()* y va cambiando el valor del *slider* para ir recorriendo la secuencia de imágenes. Cabe destacar que la captura del evento *wheelEvent()* se ha realizado en la clase Imagen, con el objetivo de que el movimiento de la bola se tenga que realizar situando el puntero en la zona de visualización de la imagen.

- **Informe del paciente en formato PDF**

Otra posibilidad que ofrece la herramienta desarrollada, es la de generar un informe de cierto paciente efectuando de esta forma un juicio clínico de lo que el radiólogo va observando. Para ello, se ha decidido crear una nueva clase que haga emerger un cuadro de diálogo donde es posible escribir todo lo que sea necesario. Esta nueva ventana, contiene un cuadro de texto que por defecto aparece con ciertos datos (nombre, fecha de nacimiento, etc.) del paciente bajo estudio. De esta forma se agiliza el proceso de redactar el informe y se evitan posibles errores. En la parte inferior del cuadro de texto aparece el botón de guardar, al presionarlo se ejecuta el explorador de archivos permitiéndonos llegar al directorio correspondiente y salvar el informe clínico en formato PDF. Se han implementado la clase informe y ReportEditor para ofrecer esta utilidad.

- **Filtro de media y filtro binario**

Gracias a las librerias ITK es posible implementar fácilmente módulos de tratamineto de imagen con el objetivo de facilitar el diagnostico clínico. En esta herramienta, se han implementado a modo de ejemplo para futuros desarrollos, un filtro

de media y un filtro binario, para ello se utilizan ciertas clases de las ITK (se detallan en el siguiente apartado) en los slots correspondientes a la acción que escoge el usuario.

Al activar el filtro de media, con el objetivo de suavizar la imagen y reducir el ruido, se genera una señal que se capta mediante la función *pidoDatosFiltroMedia()* que hace emerger un cuadro de diálogo (implementado en la clase datosfiltromedia) donde se debe indicar el número de pixels en la dirección x e y con los que se va a promediar para obtener la nueva intensidad de cada uno de ellos. Al rellenar dichos campos y presionar el botón "Filtrar", se genera un evento que es captado por el método *filtroMedia()*. En esta subrutina, es el lugar donde se utilizan las librerias ITK para aplicar el filtro y llamar a la función de repintado de la clase Imagen. En el momento que se vuelve a presionar el botón "Filtro Media" en la aplicación, éste queda desactivado (*canceloFiltroMedia()*) volviendo a visualizar la secuencia de imágenes original.

El filtro binario se comporta de la misma forma. Mediante un cuadro de diálogo se le pide al usuario que introduzca dos umbrales de intensidad para binarizar la imagen (en el siguiente apartado se explica la clase de ITK utilizada y los parámetros requeridos), se aplica el filtro y cuando se desactiva, la secuencia de imágenes vuelve a su estado original. Las funciones que posibilitan esta utilidad son *filtroBinario()*, *pidoDatosFiltroBin()* y *canceloFiltroBinario()*.

- **Filtro de rotación de imagen**

Otra herramienta interesante es la que permite hacer una transformación geométrica de rotación de la imagen. Se ha decidido que dicha rotación sea de 180º sobre el eje vertical, pero este dato se puede establecer como parámetro modificable por el usuario fácilmente. La función *rotandoImagen()* de la clase acgdicom contiene el código fuente del filtro ITK correspondiente para realizar esta transformación.

- **Barra de estado y posición en el *frame***

Por último, hay que destacar también la existencia de la barra de estado (barra inferior en la ventana principal), que va guiando al usuario a través de la aplicación en cada momento. Para ello, basta con introducir la siguiente orden, cuando el usuario presione un determinado botón o realice cierta acción:

```
labelEstado->setText(QString("Orden"));
```

Si nos fijamos en el lado derecho de esta barra aparece un mensaje cuando el cursor esta situado en la zona de visualización de imágenes indicando las coordenadas (en pixels), en la que nos encontramos dentro de la imagen. Es importante mencionar, que a diferencia del mensaje que guía al usuario, éste es permanente. En las siguientes cuatro lineas de código queda reflejada la diferencia a la hora de implementar dichas variantes:

```
labelEstado= new QLabel; //Creo el Label de info. para el usuario
statusBar()->addWidget(labelEstado); //Lo agrego dentro de la barra
labelEstadoxy= new QLabel; //Creo el Label de info. para el usuario
statusBar()->addPermanentWidget(labelEstadoxy); //Lo agrego en la barra
```

Como se puede observar la diferencia estriba en el método de inserción en la barra de estado. Si queremos hacer que el mensaje sea permanente debemos utilizar *addPermanentWidget()*, mientras que si se desea poder variar el mensaje se usará *addWidget()*. Estas y otras opciones estan más detalladas en la ayuda de Qt, buscando la clase *QStatusBar* [12].

4.5 Filtros ITK

4.5.1 Modo de aplicar un filtro de las librerías ITK

Para aplicar un filtro de las librerías ITK, el primer paso es definir el tipo de imágenes que se va a utilizar. Para ello hay que especificar el tipo de píxel y el número de dimensiones. Para el formato DICOM se emplean 3 dimensiones y *píxels* de tipo *signed short* (enteros con signo de 16 bits), como puede verse en las dos siguientes líneas de código.

```
typedef itk::Image<signed short,3>ImageType;                    (1)
ImageType::Pointer nuestroVolumen;                              (2)
```

Con esta declaración de formato se obtendrán volúmenes tridimensionales que contendrán los cortes del estudio clínico que se esté tratando.

Los filtros también deben ser declarados, posteriormente requieren que se les especifique el volumen de datos de entrada sobre el que tendrán que trabajar, y además una serie de parámetros de funcionamiento. Seguidamente, su salida podrá ser conectada a la entrada de otro filtro o, en caso de no necesitar más filtros, se podrá llamar a la instrucción *Update()*, que pondrá en marcha la ejecución de estos. A continuación se muestra un ejemplo en el que se conectan dos filtros.

```
-------------------- Descripción -----------------------
typedef itk::Image<unsigned char,3>ImageType_UCHAR;             (3)
ImageType_UCHAR::Pointer volumenSalida;                         (4)
-------------------- Filtro 1 --------------------------
typedef itk::RescaleIntensityImageFilter<ImageType,             (5)
ImageType_UCHAR>RescaleFilterType;
RescaleFilterType::Pointer rescaleFilter=RescaleFilterType::New(); (6)
```

```
rescaleFilter->SetInput(volumenEntrada->GetOutput());                (7)
rescaleFilter->SetOutputMinimum(0);                                  (8)
rescaleFilter->SetOutputMaximum(255);                                (9)
-------------------- Filtro 2 ---------------------------
typedef itk::CurvatureFlowImageFilter<ImageType_UCHAR,
ImageType_UCHAR>FilterType;                                          (10)
FilterType::Pointer filter=FilterType::New();                        (11)
filter->SetInput(rescaleFilter->GetOutput());                        (12)
filter->SetNumberOfIterations(5);                                    (13)
filter->Update();                                                    (14)
volumenSalida=filter->GetOutput();                                   (15)
```

En este ejemplo, el primer filtro es empleado para escalar los valores de un volumen de datos tridimensional con píxeles de tipo *signed short* declarado en (1), a píxeles de tipo *unsigned char* (8 bits por píxel) declarado en (3), los cuales son aptos para su representación en pantalla. Este filtro definido en (5) y declarado en (6) necesita que se le especifique el volumen de datos de entrada (7) y el rango de valores entre los que se encuentra la salida (0 y 255 en este ejemplo, (8) y (9) respectivamente). A continuación se conecta la salida de este filtro a otro que realiza un suavizado de imagen (12), cuyo único parámetro es el número de iteraciones a realizar. Finalmente, se pone en marcha la ejecución de ambos filtros (14), guardando el resultado en "volumenSalida" (15).

4.5.2 Clases de ITK utilizadas

Son muchas y muy útiles las clases que nos proporcionan las librerías ITK, con cientos de métodos de procesamiento de imagen aplicados a la ingeniería biomédica [13].

A continuación, se detallan las clases que se han empleado en el presente proyecto por orden alfabético:

1. **itkBinaryThresholdImageFilter**

Este filtro se utiliza para binarizar una imagen cambiando los valores de intensidad de los píxeles de acuerdo con dos umbrales definidos por el usuario mediante los métodos *SetUpperThreshold()* y *SetLowerThreshold()* y dos intensidades que se definen mediante *SetInsideValue()* y *SetOutsideValue()*. El valor de cada píxel de la imagen de entrada es comparado con el umbral alto y bajo. Si el valor de píxel está en el interior del rango definido por [*Lower*, *Upper*] la salida de píxel se le asigna a *InsideValue*. En caso contrario, la salida del píxel se le asigna a *OutsideValue*. Este filtro espera que tanto las imágenes de entrada como las de salida tengan el mismo número de dimensiones.

2. **itkFlipImageFilter**

Este filtro se encarga de realizar la transformación geométrica de giro de imagen sobre cualquiera de los ejes de coordenadas. Como todos los filtros de las librerías ITK, se aplica sobre todo el volumen de imágenes de la secuencia que se carga de un determinado paciente.

3. **itkGDCMImageIO**

Esta clase utiliza la clase ImageIO para la lectura y escritura de imágenes DICOM. Mediante esta clase es posible crear un puntero para poder obtener el contenido de los distintos campos de la cabecera.

4. **itkGDCMSeriesFileNames**

Esta clase genera una secuencia de ficheros cuyos nombres provienen de un fichero de imágenes DICOM. Lee todas las imágenes contenidas en un directorio

(asumiendo que sólo hay una serie) y extrae la orientación y la posición, a partir de las cuales obtiene el valor de la coordenada 3D del corte.

5. **itkImage**

Esta clase representa una imagen con una dimensión y un tipo de píxel. Las imágenes son modeladas como arrays y por tanto, se definen con un índice inicial y un tamaño. Conviene destacar que en ITK, una región de la imagen puede ser procesada por otras clases en el sistema.

6. **itkImageFileReader**

Esta clase lee los datos de una imagen desde un único fichero, que puede tener distintos formatos.

7. **itkImageSeriesReader**

Esta clase lee los datos de una imagen y construye imágenes n-dimensionales a partir de series de ficheros. Estos ficheros almacenados en un vector de caracteres, se leen usando la clase itkImageFileReader. El formato de los ficheros puede variar entre ellos, pero los datos de las imágenes deben tener el mismo tamaño en todas sus dimensiones.

8. **iktMeanFilter**

Se trata de un filtro conocido para la reducción de ruido, a cambio de un ligero emborronamiento de la imagen. El filtro calcula el valor de cada pixel de salida mediante la búsqueda de la media estadística del vecindario del pixel de entrada correspondiente. Dicho vecindario, puede ser ajustado por el usuario mediante el cuadro de dialogo pertinente.

4.6 Inserción de un *plugin*

Después de todo lo explicado en este capítulo ya estamos en disposición de detallar los pasos que se deben seguir si se quiere implementar un nuevo módulo para esta herramienta.

- Antes de comenzar se debe estudiar a fondo la necesidad que va a cubrir el *plugin* y de qué forma se va a implementar.
- En primer lugar se desarrollará la interfaz de usuario mediante *Qt Designer* editando el fichero "acgdicom.ui" (ventana principal) tal y como se explica en el apartado 4.3, modificando si es necesario la IU actual. Recordar que se disponen de *widgets* por defecto de todo tipo (Fig. 4.14) pero también es posible implementar propios a partir de clases desarrolladas por el programador.

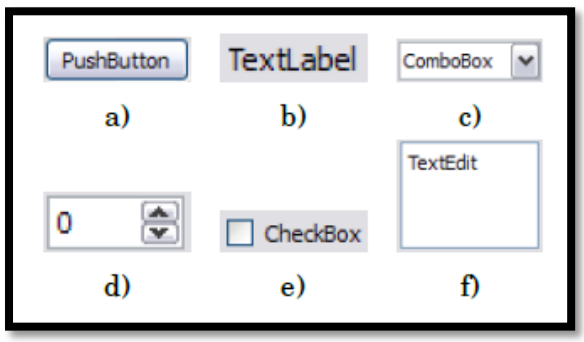


Figura 4.14: Widgets por defecto más comunes.

- Una vez se dispone de los elementos gráficos necesarios para que el usuario pueda poner en práctica las funcionalidades que aporta el *plugin* se deberán programar las acciones correspondientes (en funciones denominadas slots y detalladas en el apartado 4.3.3) que respondan al evento que se capta, es decir, se implementará el *plugin* propiamente dicho. Lógicamente al tratarse de un módulo independiente, se pueden programar las clases individualmente y

llamarlas mediante los *slots* de la ventana principal (que responden a eventos producidos por el usuario desde la UI) incluyendo los archivos ".h" en "acgdicom.h" mediante la siguiente instrucción:

```
#include "nombreClasePlugin.h"
```

Si el código fuente del *plugin* se encuentra en librerías externas es totalmente necesario incluirlas en el proyecto tal y como se hace en el capítulo 2 con las ITK e incluir el fichero de cabecera como se ha acaba de comentar.

- Como los módulos que se diseñen estarán basados en biomarcadores de imagen, para tratar de conseguir un mejor diagnóstico, en muchas ocasiones será necesario dibujar sobre la imagen o seleccionar una cierta región de interés, por lo que se tendrá que ampliar la clase Imagen con los métodos necesarios, teniendo en cuenta que el método donde se realizan trazos sobre la imagen es *PaintEvent()*.

Nota: Si se desean incluir *plugins* tanto de las librerías ITK como VTK (visualización y reconstrucción de volúmenes 3D), se deben seguir al pie de la letra los capítulos 2 y 4 del presente trabajo, donde se ha explicado la forma de añadirlas a nuestro proyecto y utilizarlas en nuestras clases.

Capítulo 5

Resultados

5.1 Un paseo por la aplicación

En el presente capítulo, vamos a mostrar la aplicación introduciéndonos en sus menús y submenús y ejecutando las funcionalidades de las que se ha provisto el visor.

Al abrir la aplicación se presenta la ventana principal de la misma con las opciones, que se pueden llevar a cabo, activadas y una primera instrucción para orientar al usuario en la barra de estado (Fig. 5.1).

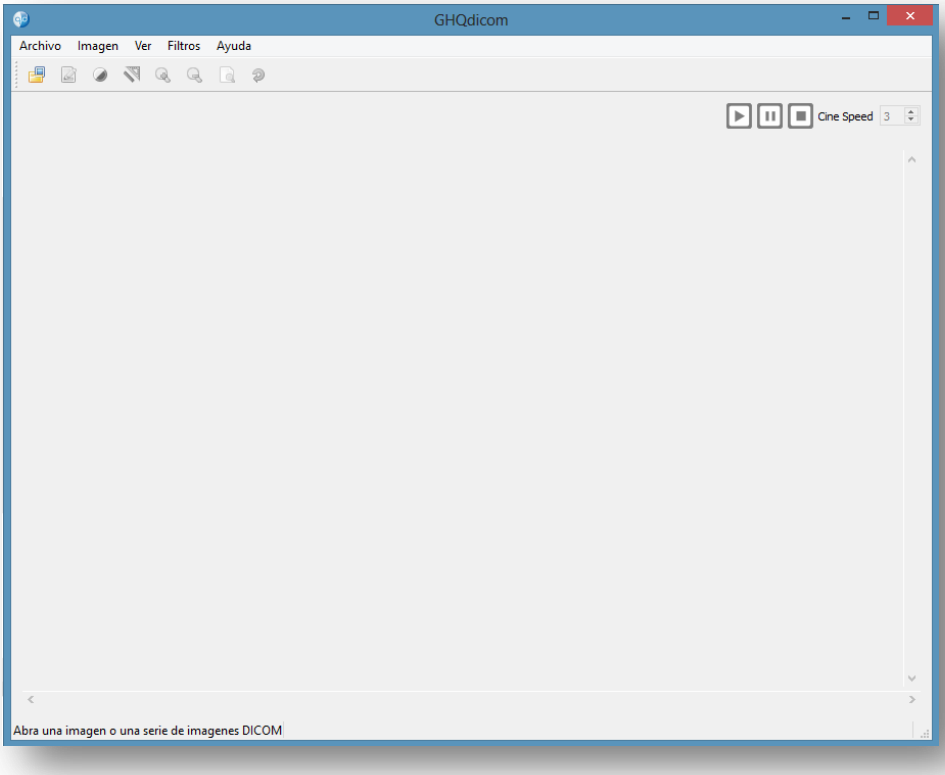


Figura 5.1: Aspecto de la aplicación en el estado inicial.

Lo primero que se debe hacer, tal y como indica la barra de estado, es abrir una imagen o una serie de imágenes DICOM. Para ello, se hace *click* sobre el menú "Archivo" y se selecciona una de las dos opciones (Fig. 5.2). Lo más habitual será abrir una secuencia de imágenes, por lo que se ha creado un acceso directo en la barra de herramientas para realizar dicha acción (Fig. 5.3). Comentar también que en el menú archivo, se dispone de la opción "Salir" que si se selecciona dará por finalizada la instancia de la aplicación.

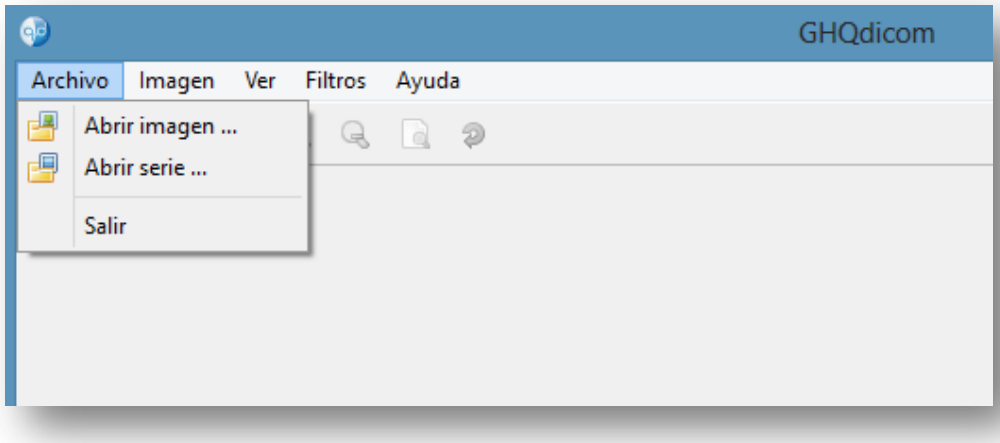


Figura 5.2: Menú "Archivo" de la aplicación.

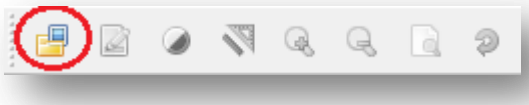


Figura 5.3: Acceso directo al explorador de Windows para abrir una serie DICOM.

Una vez se escoja la opción "Abrir serie…", se lanzará el explorador de Windows para buscar y seleccionar la serie de imágenes sobre la que se desea trabajar. Se selecciona el directorio pertinente y se pulsa la opción "Seleccionar carpeta" (Fig. 5.4).

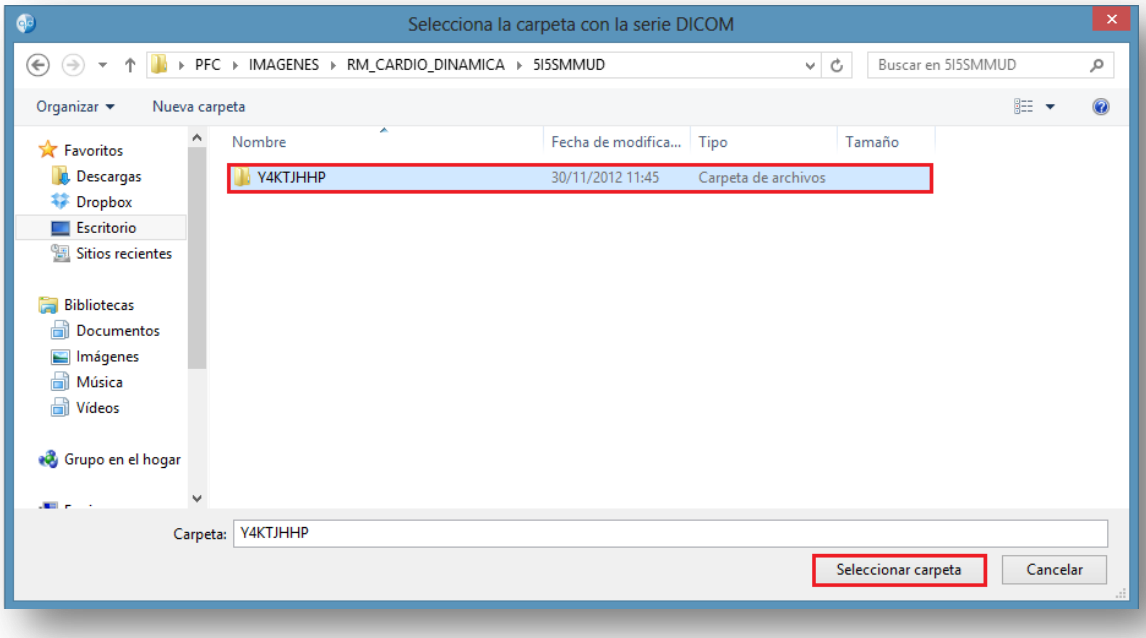


Figura 5.4: Explorador de archivos de Windows.

Transcurridos unos instantes (el tiempo de espera dependerá del "peso" de la serie DICOM), se podrá visualizar el primer *frame* que compone la serie y se iluminarán ciertas opciones, que al iniciar la aplicación, aparecían desactivadas (Fig. 5.5).

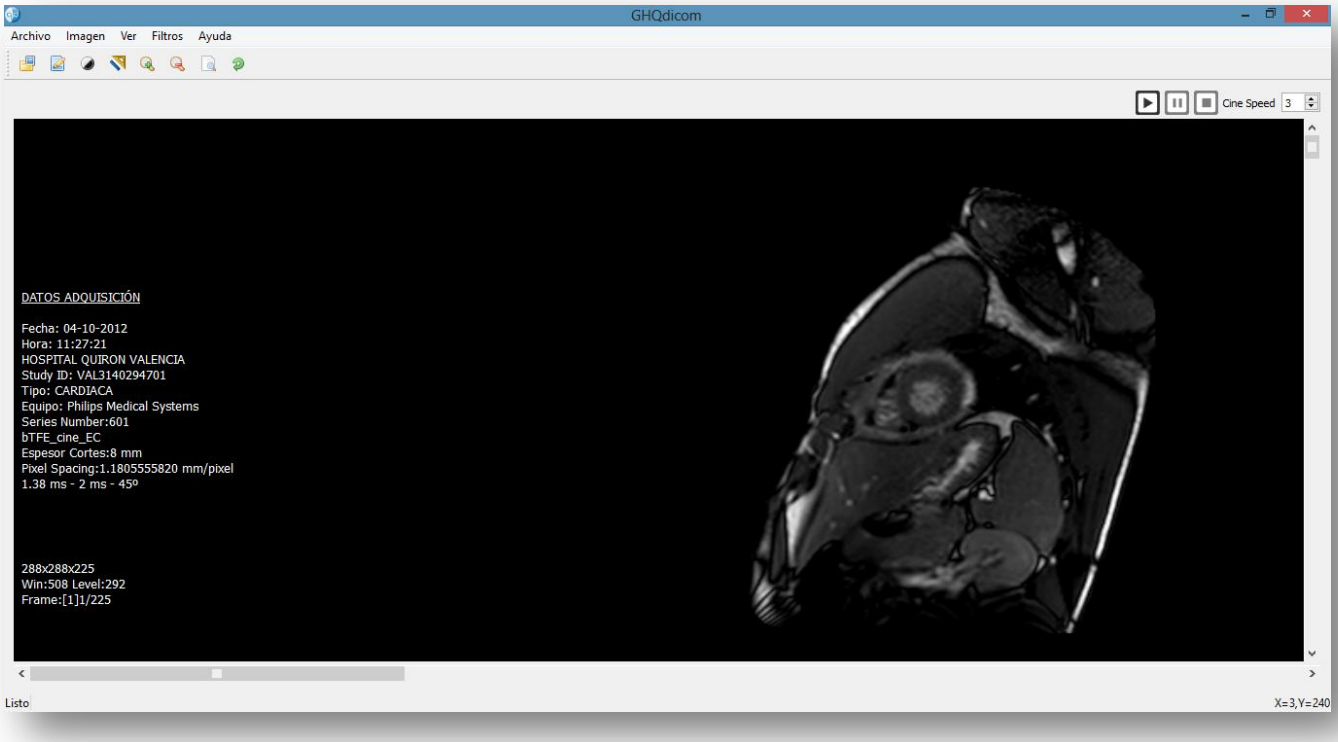


Figura 5.5: Serie cargada en nuestra aplicación.

Como se puede observar en la figura 5.5, a la izquierda de la imagen aparecen automáticamente los datos del paciente, los datos de adquisición y los del propio *frame* que se está visualizando. **Cabe destacar, que en todas las imágenes que aparecen en esta memoria, se han ocultado los datos del paciente con el fin de preservar su identidad mediante una opción que se ha implementado a conciencia y que se explica posteriormente.**

A partir de este momento, vamos a ir viendo el efecto que tiene cada acción que el usuario selecciona:

Como se ha comentado en el capítulo anterior, podemos avanzar o retrasar los *frames* que componen la secuencia de varias formas: En primer lugar, mediante los sliders vertical y horizontal. Se recuerda que si se carga una serie estática solo se activará el vertical, mientras que si la serie es dinámica, se activaran ambos (el ejemplo que se expone trata de una serie dinámica). El segundo método para navegar por la secuencia de imágenes es mediante el movimiento de la bola del ratón, situando el cursor encima de la zona de visualización de la imagen. Por último, gracias a la barra del modo cine, podemos obtener una transición automática con solo un *click* pudiendo variar la velocidad de transición gracias a un *widget* spinBox (Fig. 5.6).

Figura 5.6: Barra modo cine.

Al abrir el menú “Imagen” y seleccionar la primera opción “Brillo/Contraste” en la barra de estado se muestra el mensaje “Ahora puede variar el brillo y contraste de la imagen”. Por lo que mediante movimientos verticales sobre la zona de visualización se variará el parámetro centro con la consiguiente variación del brillo de la imagen (Fig. 5.7) y

mediante movimientos horizontales se varía el valor de la ventana con la consiguiente variación del contraste de la imagen (Fig. 5.8)

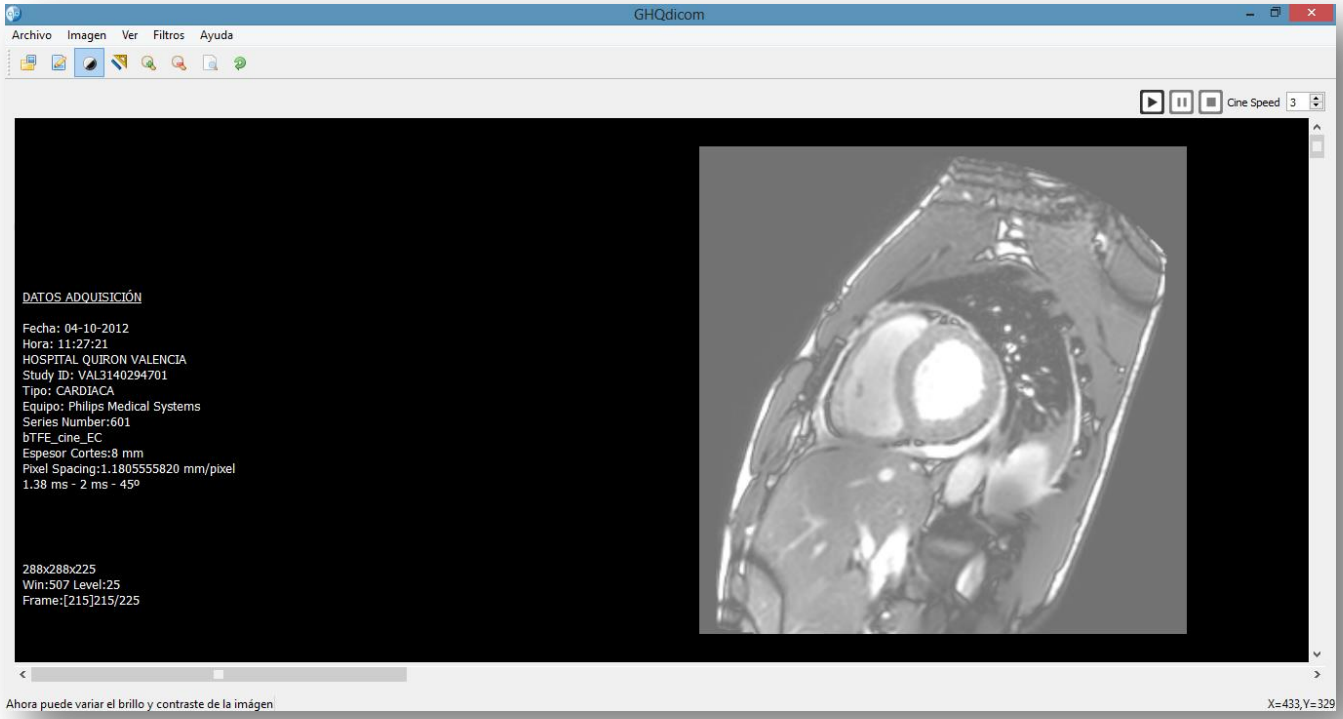


Figura 5.7: Variación del brillo de la imagen.

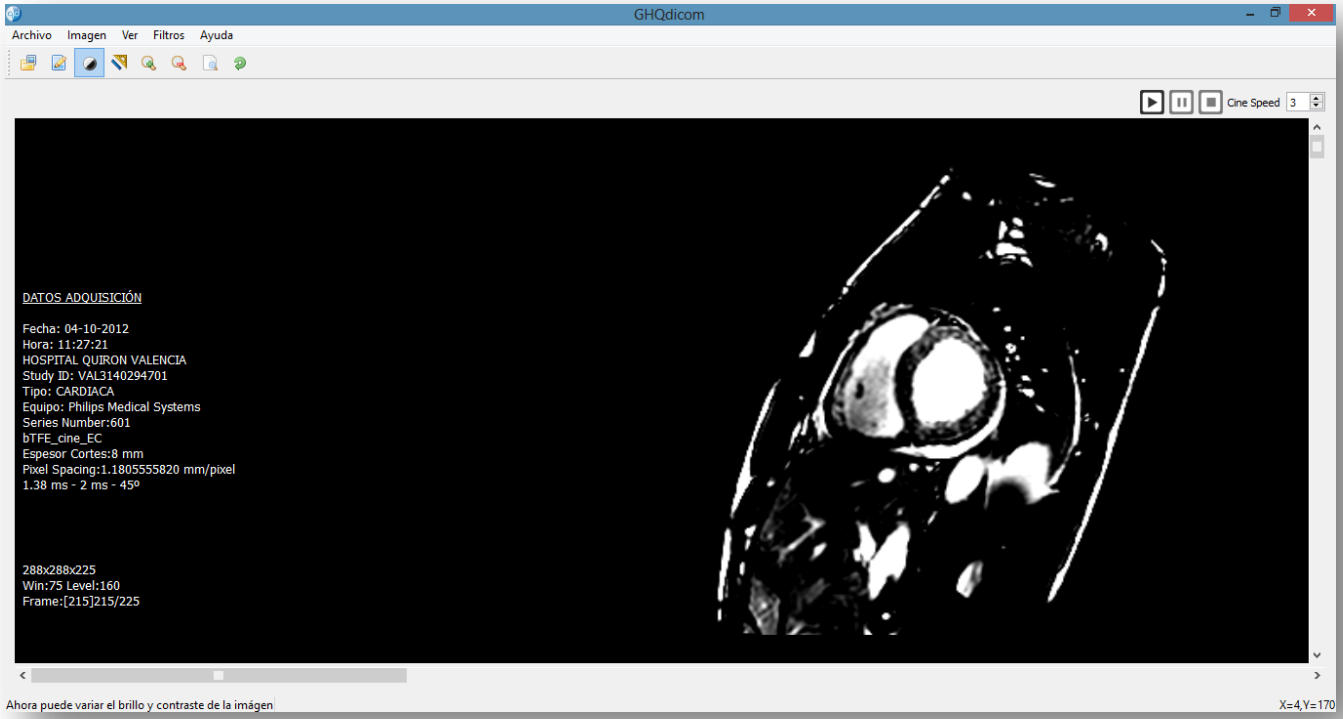


Figura 5.8: Variación del contraste de la imagen.

Escogiendo la segunda opción del menú "Imagen", la opción "Regla" en la barra de estado se reflejará "Ahora puede medir", por lo que haciendo *click* en un primer punto y arrastrando el cursor hasta un segundo punto donde soltamos el botón, se traza una recta. La distancia real que comprende la recta que se ha dibujado se muestra en la parte superior izquierda de la imagen (Fig. 5.9).

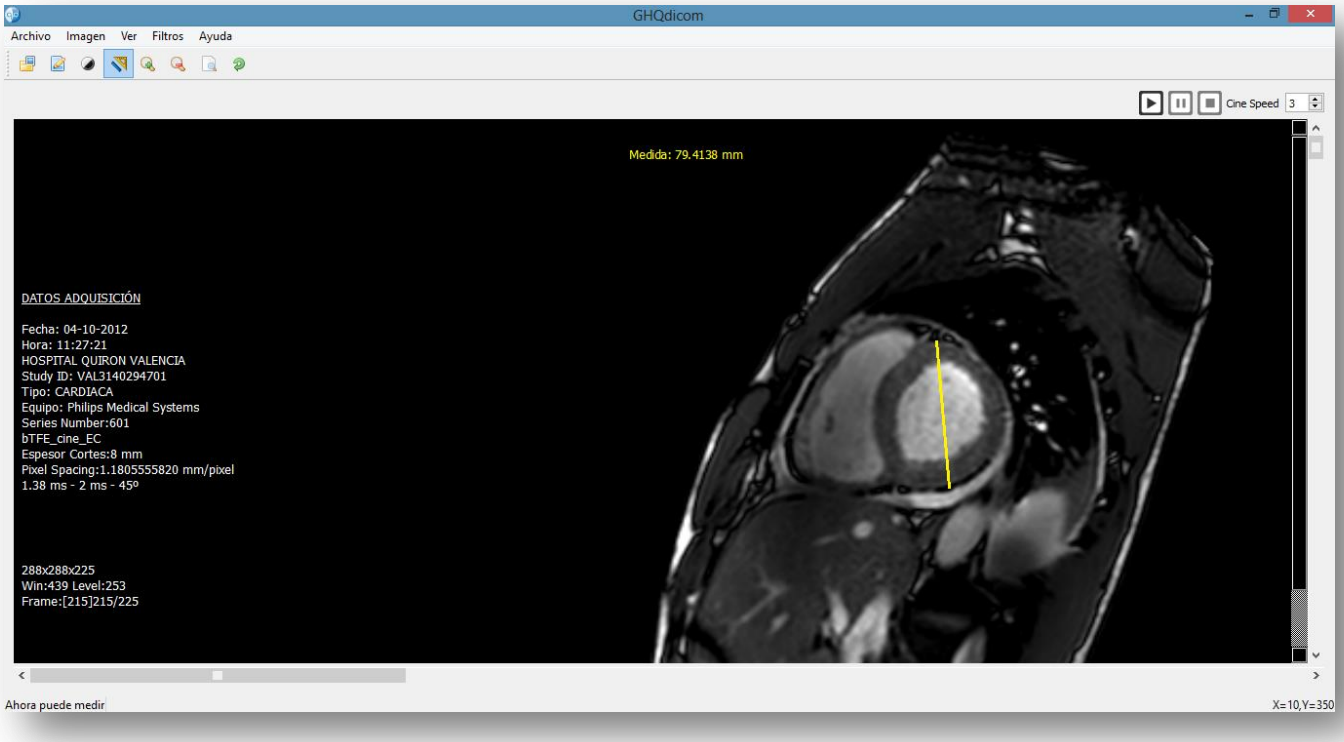


Figura 5.9: Ejemplo de medida en una imagen.

Por último, escogiendo la opción "Giro" del menú "Imagen" o de la barra de herramientas, se efectúa una transformación geométrica de rotación 180º sobre el eje vertical (Fig. 5.10). Hay que recordar que se trata de un filtro de las ITK por lo que se aplica a todos los *frames* de la secuencia de imágenes. Si se desea desactivar y volver a la secuencia de imágenes original basta con deseleccionar la opción.

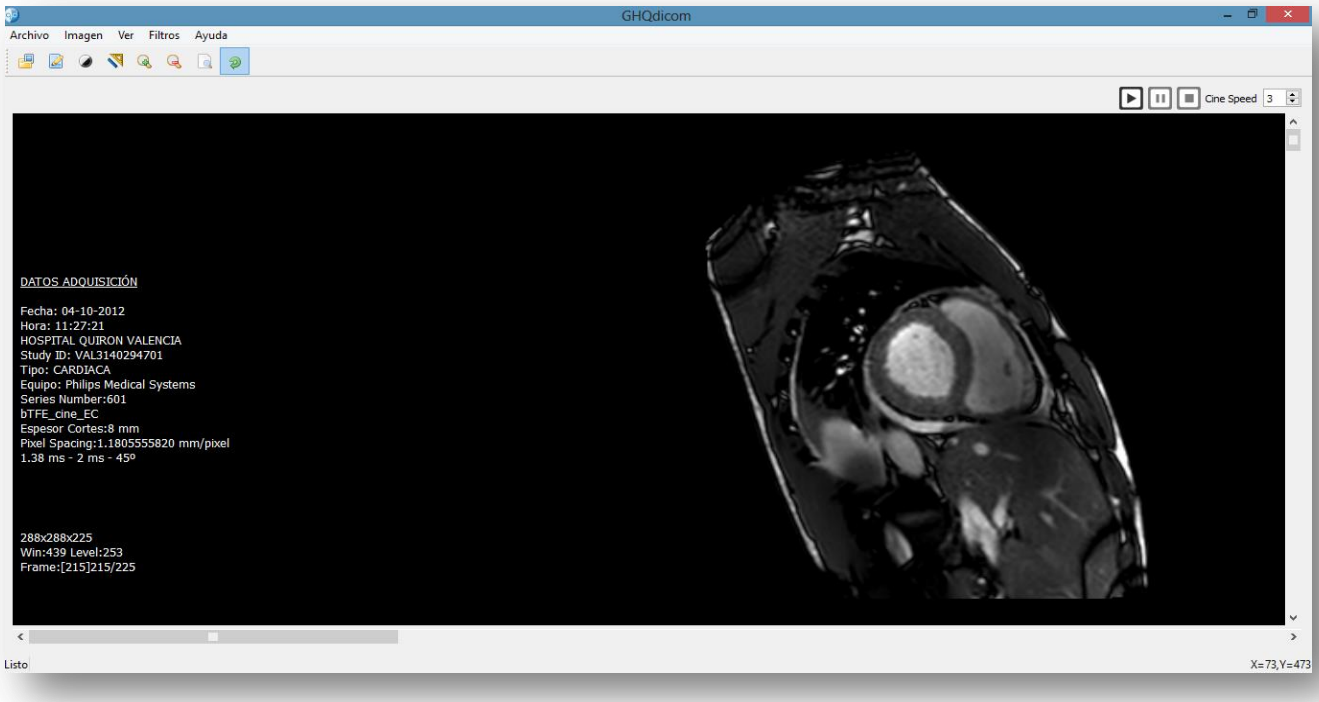


Figura 5.10: Rotación 180º de la imagen sobre el eje vertical.

Desde el menú "Ver" es posible aumentar y disminuir las dimensiones de la imagen. Lo habitual será aplicar zoom para observar algún detalle de la imagen. La manera de hacerlo es seleccionando la opción "Aumentar Zoom" desde el menú o desde la barra de herramientas (Fig. 5.11).

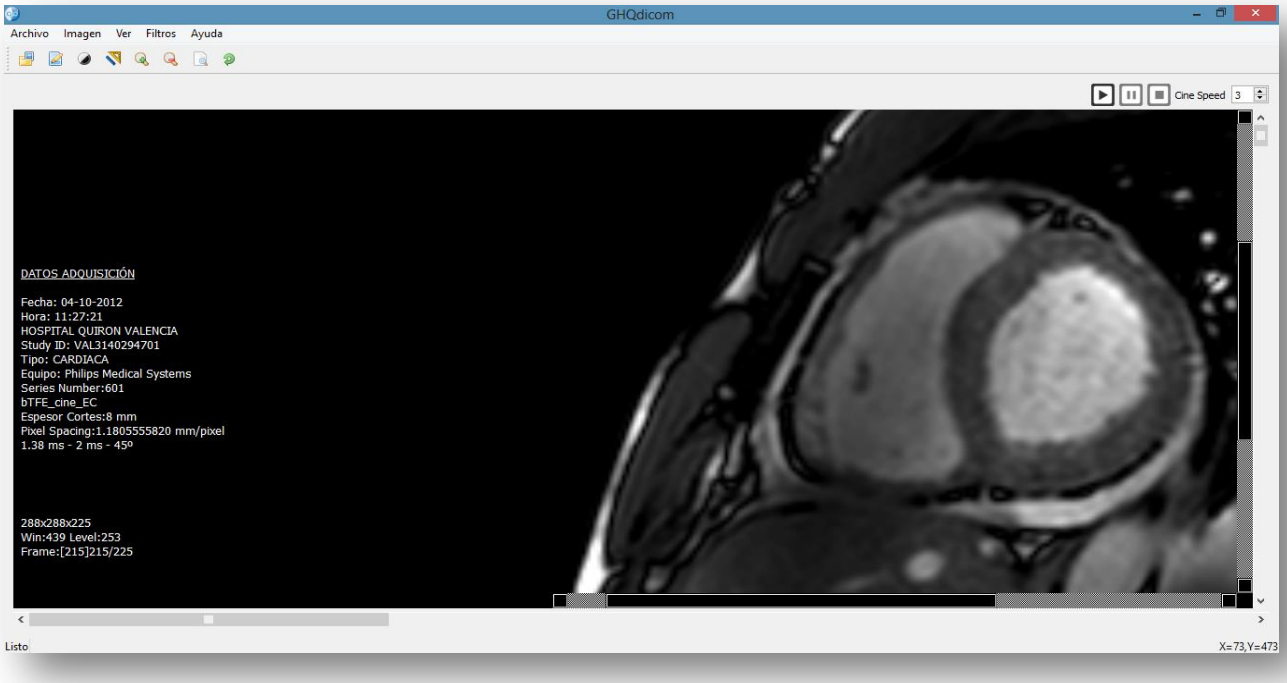


Figura 5.11: Imagen con "Aumentar Zoom" aplicado.

Como ya se ha comentado anteriormente, para poder recorrer visualmente toda la imagen aparecen dos sliders interiores de color negro. Si por el contrario lo que queremos es alejarnos de la imagen, se deberá presionar el botón "Disminuir Zoom" del menú "Ver" o presionando el botón de acceso directo de la barra de herramientas (Fig. 5.12).

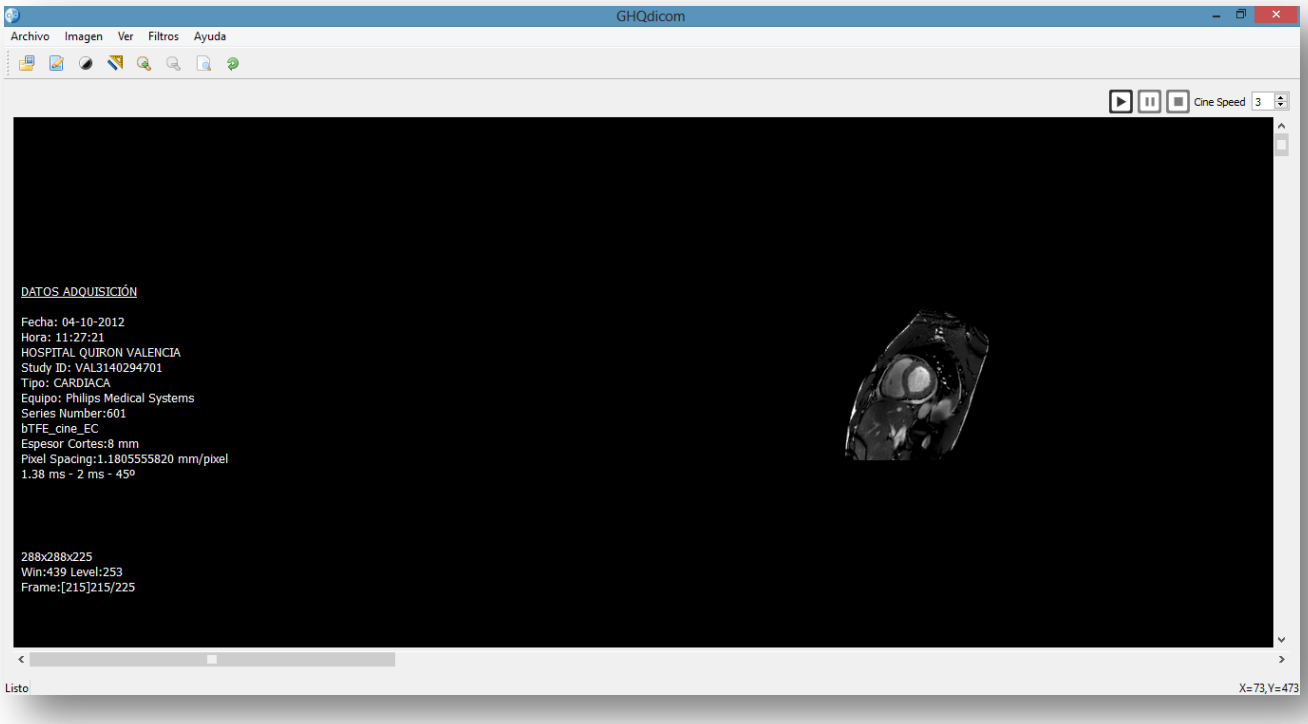


Figura 5.12: Imagen con "Disminuir Zoom" aplicado.

Se ha implementado también la opción de "Zoom original" la cual nos devuelve a las dimensiones originales de la imagen, independientemente del nivel de zoom que esta tuviera aplicado.

Las opciones restantes que componen el menú "Ver" tienen como objetivo ocultar o mostrar una serie de elementos. Cuando se lanza la aplicación por defecto se muestra la barra de modo cine, pero esta se puede desactivar pulsando la opción "Botones modo cine" en el menú que se comenta. Las dos restantes opciones "Datos paciente" y "Datos

adquisición" tienen como objetivo poder ocultar dichos datos y así no revelar tanto la identidad del paciente como la institución donde se ha realizado el estudio (Fig. 5.13).

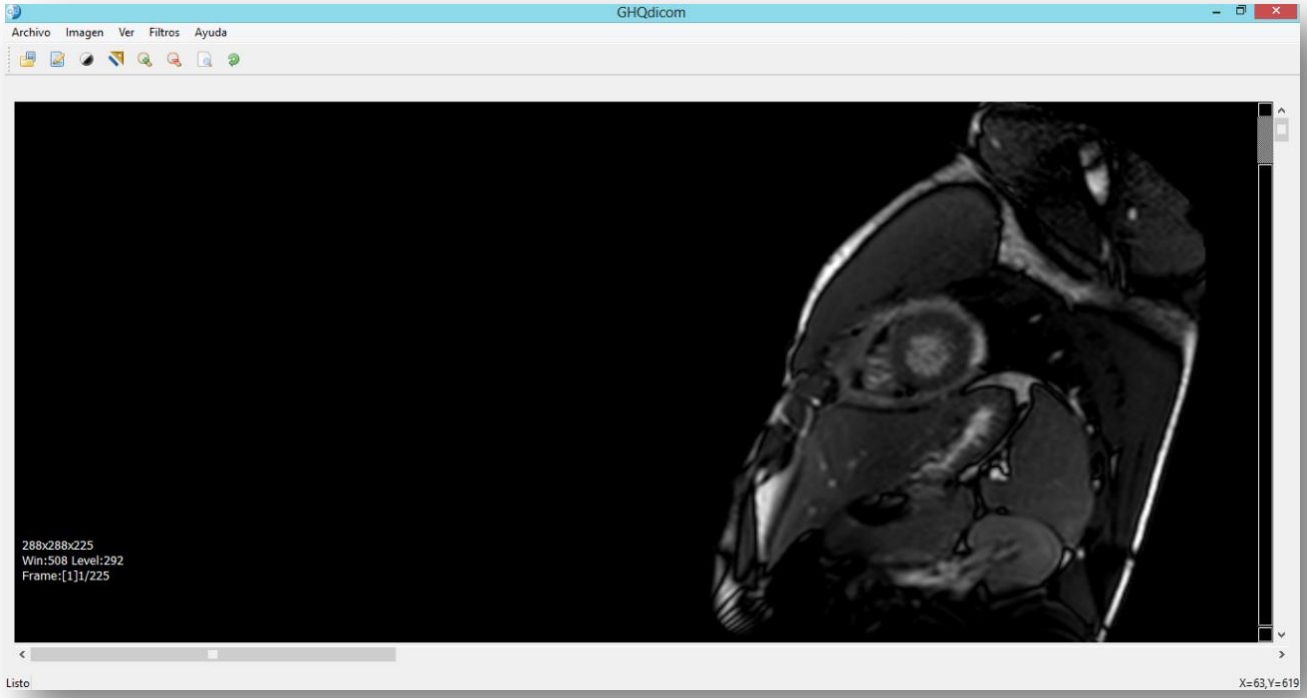


Figura 5.13: Datos paciente y adquisición ocultos al igual que barra modo cine.

En el menú "Filtros" se pueden encontrar los dos filtros que se han implementado por medio de las librerías ITK. En primer lugar, cuando el usuario activa el filtro de media (opción "Media"), aparece un cuadro de diálogo para que indique el número de vecinos tanto en dirección x como en dirección y para con los que se efectuará el promedio (Fig. 5.14).

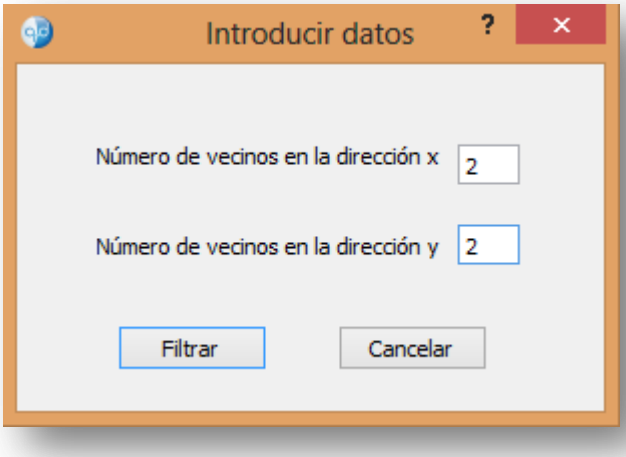


Figura 5.14: Campos a rellenar para aplicar el filtro de media.

Pulsando el botón “Filtrar” se aplicará el filtro reduciendo el ruido pero emborronando ligeramente la imagen (Fig. 5.15).

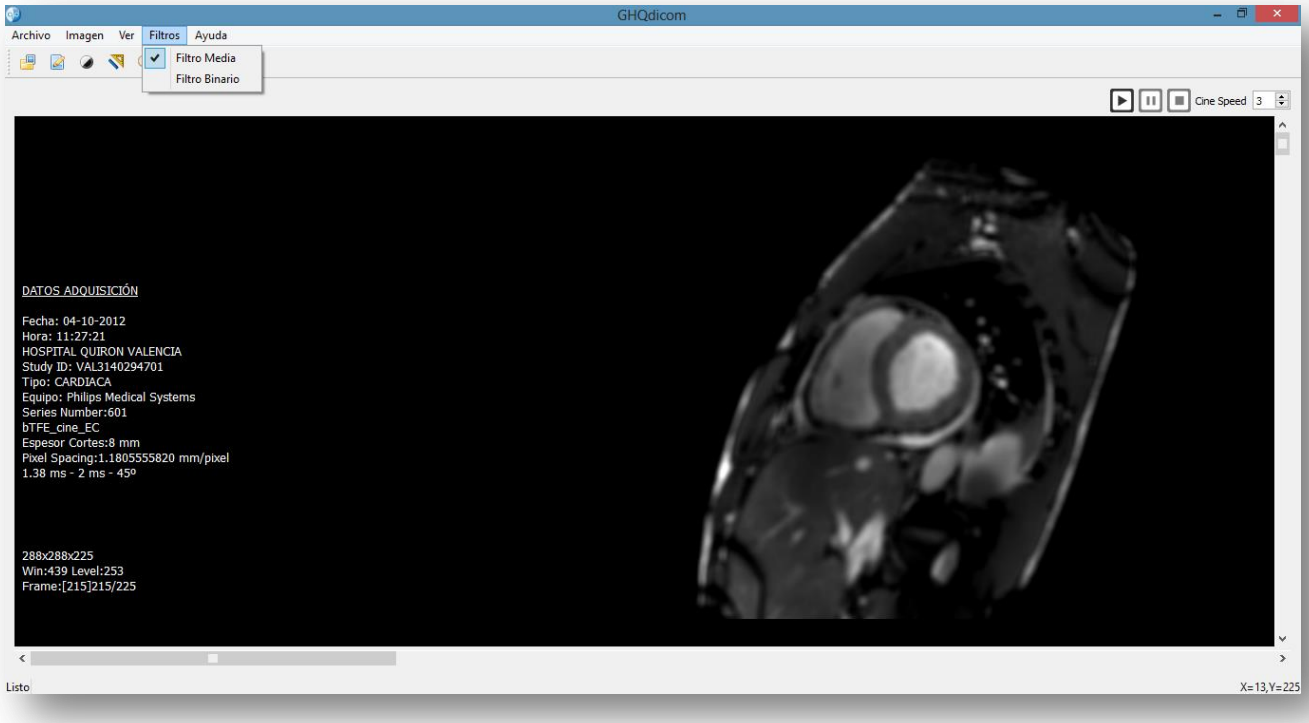


Figura 5.15: Secuencia con el filtro de media aplicado.

Como podemos observar cuando activamos el filtro aparece la opción seleccionada. Si se desea desactivar y volver a la secuencia de imágenes original basta con deseleccionar la opción.

El proceso es similar cuando se activa el filtro binario (opción “Filtro Binario”). En primer lugar, emerge un cuadro de diálogo en el que el usuario debe indicar dos umbrales (Fig. 5.16). El valor de cada píxel de la imagen de entrada es comparado con los valores introducidos por el usuario (umbral alto y bajo). Si el valor de píxel está en el interior del rango definido por [Bajo, Alto] el pixel de salida toma el valor 255 (blanco). En caso contrario el pixel toma valor 0 (negro).

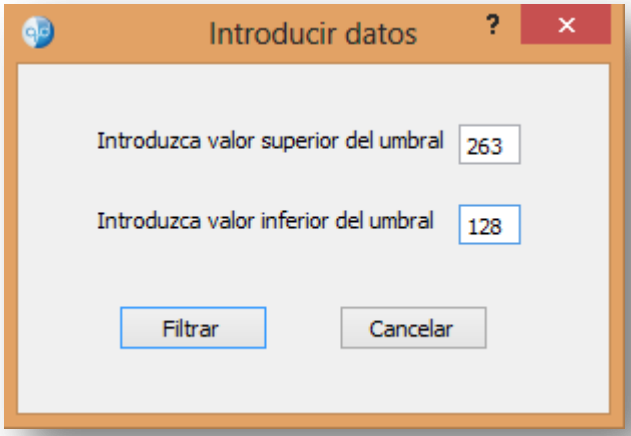


Figura 5.16: Campos a rellenar para aplicar el filtro binario.

Pulsando el botón "Filtrar" se aplicará el filtro binarizando la imagen (Fig. 5.17).

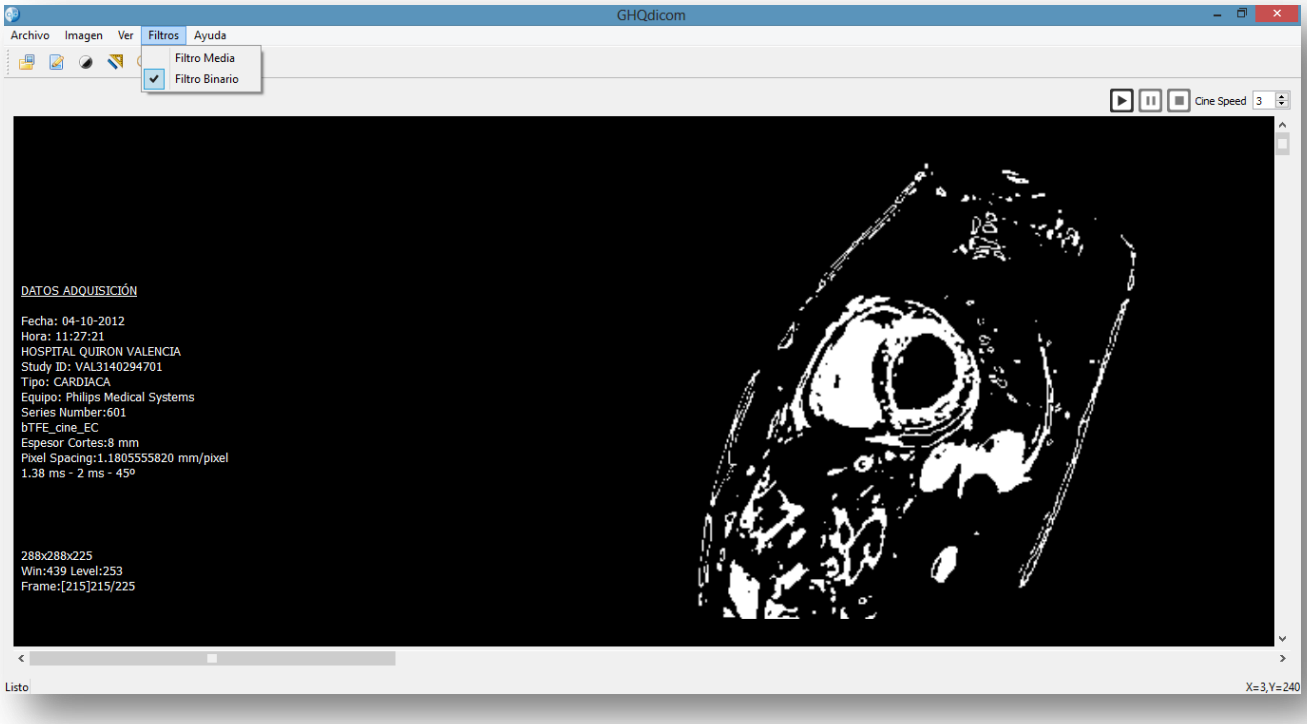


Figura 5.17: Secuencia con el filtro binario aplicado.

Los botones de la barra de herramientas son accesos directos de los menús que conforman la aplicación a excepción del botón "Generar informe" que utilizará el facultativo para generar un juicio clínico sobre el caso que está estudiando. Presionando

dicho botón aparecerá un cuadro de diálogo donde se escribirá todo lo necesario (Fig. 5.18).

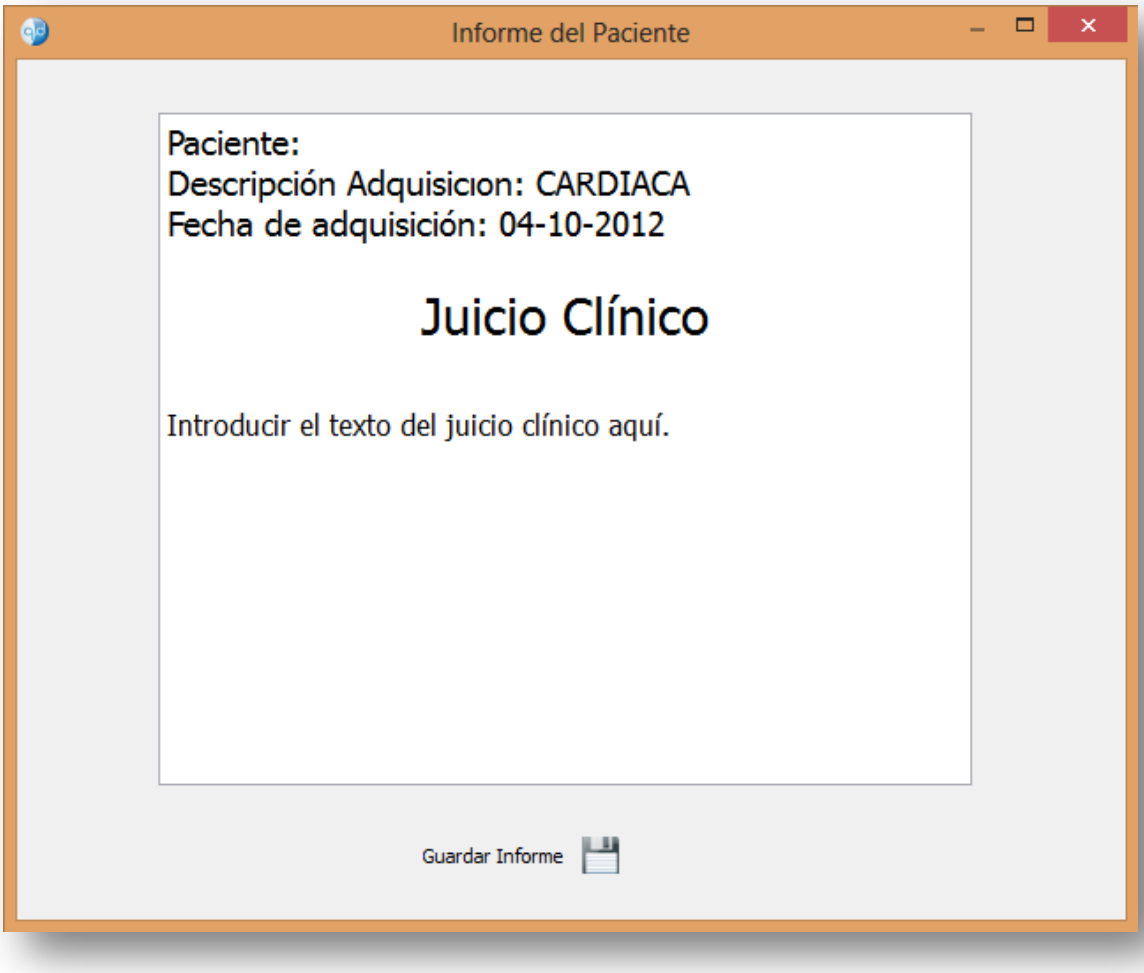


Figura 5.18: Cuadro de dialogo para elaborar el juicio clínico del paciente.

Cuando se finaliza el informe, al presionar el botón "Guardar Informe" se ejecuta el explorador de archivos permitiéndonos llegar al directorio correspondiente y salvar el informe clínico en formato PDF.

Por último, en el menú "Ayuda" si se selecciona la opción "Acerca GHQdicom" se muestra un cuadro de diálogo con los detalles del software, desarrollador y directores del proyecto objeto de la aplicación (Fig. 5.19). En cuanto a la información necesaria para la instalación del software, se debe consultar el Anexo II.

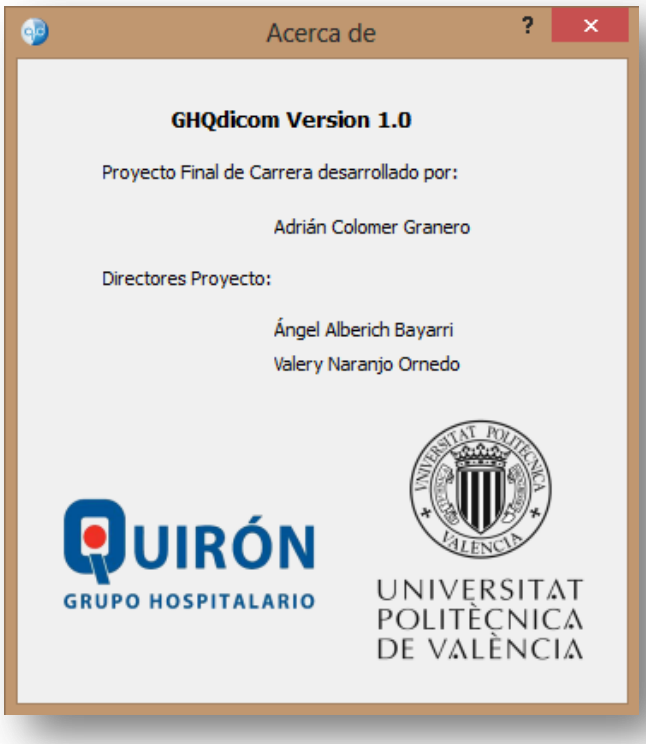


Figura 5.19: Cuadro de diálogo acerca de los detalles de la aplicación.

Para finalizar el presente capítulo, que tiene como objetivo guiar al usuario a través de la aplicación, se muestran unas capturas de pantalla con diferentes casos cargados en el visor DICOM.

Figura 5.20: Caso cabeza pediátrica cargado en el visor DICOM.

Figura 5.21: Caso mama cargado en el visor DICOM.

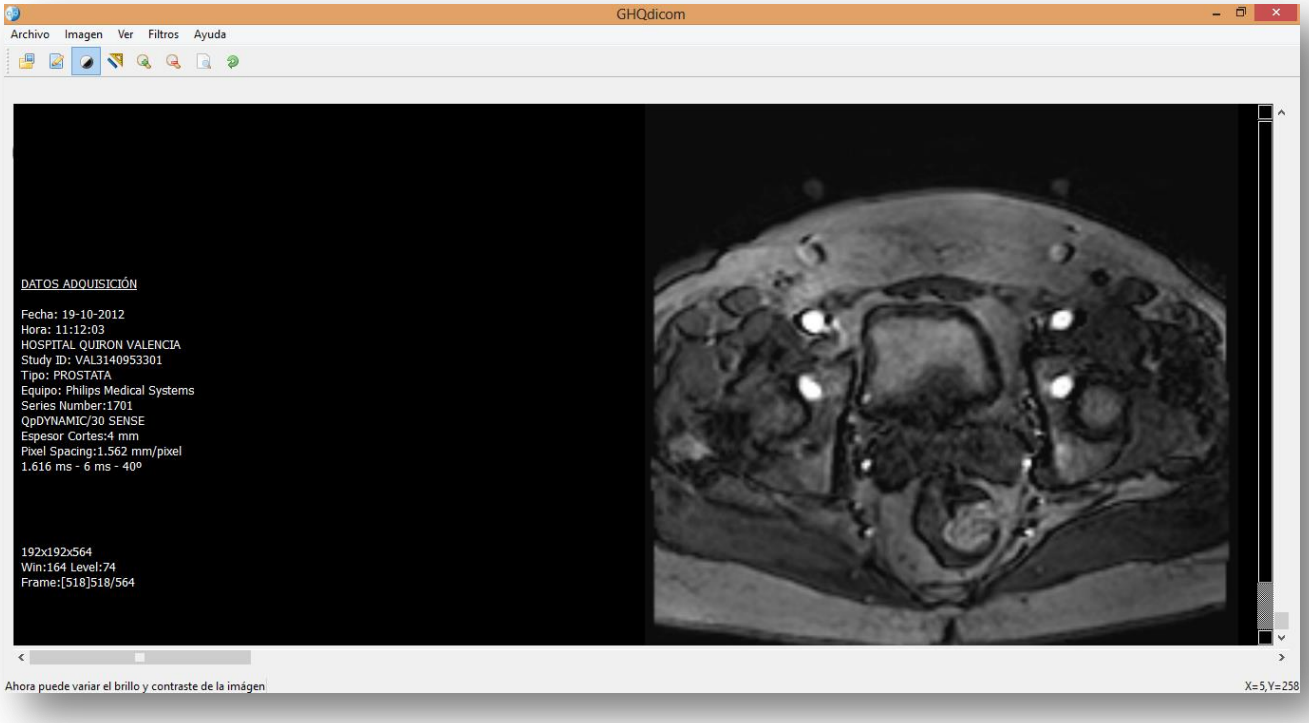


Figura 5.22: Caso próstata cargado en el visor DICOM.

Figura 5.23: Caso pie cargado en el visor DICOM.

Figura 5.24: Caso rodilla cargado en el visor DICOM.

Capítulo 6

Conclusiones y líneas futuras

6.1 Conclusiones

En este proyecto se ha diseñado y desarrollado un visor original de imágenes médicas que permite incorporar módulos o *plugins* para la mejora del diagnóstico clínico utilizando técnicas de postproceso de imagen. Se trata de un software de código abierto e independiente del fabricante de adquisición de imágenes médicas. La arquitectura modular que posee, potencia enormemente la aplicación, puesto que ésta es ampliable con gran facilidad mediante algoritmos independientes que realicen cierta función, como por ejemplo, métodos de segmentación que posibiliten la separación e identificación de ciertas regiones de la anatomía humana, módulos de detección automática de lesiones en diversos tejidos mediante técnicas de procesado de imagen, métodos de clasificación de las lesiones según el nivel de desarrollo de cada una, y en general cualquier nuevo desarrollo basado en Biomarcadores de Imagen.

Otra de las prioridades de la herramienta, ha sido la de diseñar e implementar una interfaz de usuario muy intuitiva, ya que está pensada para su uso en un entorno clínico multidisciplinar con especialistas clínicos y profesionales de la ingeniería biomédica.

En cuanto a la ventana principal de la aplicación, cabe señalar la importancia de la imagen que se visualiza, para ello, se ha provisto al marco que contiene las imágenes de unas amplias dimensiones. Además se han implementado diversos métodos para la transición de los diferentes *frames* que componen la secuencia bajo estudio, diferenciando ésta, según su tipo (estática o dinámica) y activando las barras de desplazamiento necesarias posibilitando así una visualización funcional, que no ofrecen la mayoría de visores en la actualidad.

Por todas estas características, se puede concluir afirmando que se han cubierto los objetivos que perseguía este proyecto final de carrera, abriendo un amplio abanico de posibilidades que permiten el crecimiento futuro de la plataforma desarrollada.

6.2 Líneas futuras

Debido a la gran escalabilidad que ofrece la herramienta son diversas las líneas futuras que admite este proyecto. Es posible crear múltiples desarrollos que mejoren y potencien la aplicación. Incluso es posible que pueda extenderse su uso entre la comunidad de especialistas en la imagen médica y se utilice como plataforma de trabajo con Biomarcadores de Imagen. Se proponen como líneas futuras principales:

- Implementar más funcionalidades a modo de *plugin* basadas en biomarcadores de imagen que hagan crecer a la aplicación.
- Obtener y mostrar valores de difusión y perfusión optimizando así el método de visualización. Para ello, será necesaria la lectura de *tags* privados de la cabecera DICOM, mediante una versión de las librerías ITK más avanzada, concretamente ITK 4.0 o superior.

- Desarrollo de un método de captura de imágenes (series) modificadas por el usuario, es decir, exportar en formato DICOM, imágenes procesadas mediante las técnicas de las que dispone el visor.
- Realización de un módulo de renderizado de volúmenes 3D a partir de las imágenes adquiridas en 2D mediante el uso de librerías VTK.
- Trabajar con bases de datos que alberguen los informes que se realicen de todos los pacientes bajo estudio.
- Optimizar el tiempo de espera cuando se efectúa la carga de una serie de imágenes utilizando diversos hilos (clase *QThread* del software Qt).

Anexo I

Construcción y compilación de Weasis mediante terminal bash

Los pre-requisitos necesarios para llevar a cabo la construcción y compilación de Weasis a través de un terminal bash son los siguientes:

- Descargar e instalar JDK6 o superior (Sun o OpenJDK).
- Maven 2.2 o superior (http://maven.apache.org/guides/index.html)
- Se precisa del software GitBash en el caso de trabajar en Windows o de un cliente con instalación del paquiete git-core en caso de trabajar en Linux o MAC.

Estando a disposición de un ordenador que contenga los pre-requisitos mencionados anteriormente se procederá a seguir los siguientes pasos:

1. Establecer la variable de entorno JAVA_HOME.
   - Acceder a Panel de Control y escribir "variables entorno" en la barra de búsqueda.
   - Pulsar "Editar las variables de entorno de esta cuenta"
   - En el cuadro de diálogo que se abre, hacer click sobre "Nueva…"
   - Establecer "Nombre de la variable: JAVA_HOME"
   - Establecer "Valor de la variable: C:\Program Files\Java\jdk1.7.0_09" (Este valor será la ruta absoluta donde se instale JDK).

   Nota: Otra forma de establecer dicha variable de entorno es accediendo a GitBash y en la línea de comandos ejecutar:

export –p JAVA_HOME=C:/Program Files/Java/jdk1.7.0_09

2. Obtención del código fuente de Weasis.
   - Desde la línea de comandos de GitBash ejecutamos:

     git clone git://github.com/nroduit/Weasis.git

   Nota: Este comando lo que hará será clonar el código fuente existente en el repositorio a un directorio local llamado Weasis.

3. Compilación de Weasis.
   - Desde el terminal GitBash nos dirigimos al directorio donde se ha creado Weasis y debemos ejecutar la orden externa mvn (Como orden externa debemos ir al directorio donde se encuentra la orden para ejecutarla o en su defecto crear una variable de entorno cuyo valor será la ruta absoluta donde se encuentre dicha orden). La orden mvn (maven) se encontrará en el lugar donde hayamos descomprimido la carpeta Maven 2.2 (o superior).
   - Concretamente la instrucción que ejecutaremos será: **mvn clean install**
   - Si Weasis compila correctamente se podrá observar: BUILD SUCCES.

4. Construcción de distribuciones Weasis.
   - Desde el terminal GitBash nos dirigimos al directorio *weasis-distributions*, para ello basta con ejecutar en la línea de comandos (estando en el directorio *Weasis*): **cd weasis-distributions**
   - Una vez en dicho directorio ejecutar: **mvn clean package**
   - Se puede utilizar el pack200 compresión que reduce el tamaño del .jar considerablemente: **mvn clean paquete –P pack200**

- Para utilizar Weasis Webstar a través de Java es necesario firmar con nuestras credenciales a la hora de la creación del .jar. Dichas credenciales puden ser generadas mediante la keytool : (http://docs.oracle.com/javase/1.3/docs/tooldocs/win32/keytool.html)
- Añadir la opción "-Dportable=true", para la construcción de la distribución portable: **mvn clean paquete -Dportable=true**
- Los archivos de distribución los podremos localizar en *target/web_dist* y *target/portable-dist.*

Construcción y compilación de Weasis mediante Eclipse

Los pre-requisitos necesarios para llevar a cabo la construcción y compilación de Weasis a través de un terminal bash son los siguientes:

- Descargar e instalar JDK 6 o superior (Sun u OpenJDK).

  Nota: JRE no es adecuado para m2eclipse.
- Descargar e instalar Eclipse IDE para desarrolladores de Java 3.7 (ya incluye la integración de Maven y diseñador swing) o Eclipse Classic (recomendada 3.5.2, 3.6.2, 3.7 o e4).
- Git, Eclipse plugin Egit.
- m2eclipse plugin (integración Maven para Eclipse), incluido en Eclipse IDE para desarrolladores de Java 3.7 (Indigo).

Estando a disposición de un ordenador que contenga los pre-requisitos mencionados anteriormente se procederá a seguir los siguientes pasos:

1. Configuración Eclipse.

- Iniciar Eclipse y seleccionar la ruta del espacio de trabajo Weasis.

Nota: *Se recomienda crear un nuevo espacio de trabajo para mantener una configuración Weasis específica.*

- *Ir a Window > Preferences > Java > Installed JREs > Execution Enviroments* y seleccionar el JDK instalado para JavaSE-1.6.
- En *Ventana > Preferences > General > Workspace:*

  · Seleccionar UTF-8 para la codificación de archivo de texto.

  · Seleccionar Unix en Nuevo archivo de texto delimitador de línea.

2. Código, estilo y convención.

- Descargar el *Weasis code style format:*

  En *Window > Preferences > Java > Code Style > Formatter*, click en *Import* y selecciona el archivo *weasis-eclipse-formatting.xml*

  Pulse aceptar después de importar.
- Descargar el *Weasis clean up code style:*

  En *Window > Preferences > Java > Code Style > Clean up*, click en *Import* y selecciona el archivo *weasis-eclipse-formatting.xml*

  Pulse aceptar después de importar.
- Descargar el *Weasis code templates:*

  En *Window > Preferences > Java > Code Style > Code Templates*, click en *Import* y selecciona el archivo *weasis-eclipse-formatting.xml*

  Pulse aceptar después de importar.

3. Instalar Egit y m2eclipse en Eclipse.

- Ir a *Help > Install New Software...*
- Seleccionar *work with* (menu *All Available Sites*).

- Escribir en la barra *type filter text* m2eclipse packages (Maven Integration for Eclipse) o m2e package from Eclipse 3.7.
- Pulse *Finalizar* para instalar y validar la advertencia de seguridad cuando se le solicite.
- Escribir en la barra "type filter text" egit y repetir el paso anterior.

4. Obtención del código fuente de Weasis.

- Desde la perspectiva de GitEclipse: *Window* > *Open Perspective* > *GIT Repository Exploring*. Click en el botón "Clone a GIT repository".
- En el nuevo cuadro de diálogo introducir URLs:

  git://github.com/nroduit/Weasis.git (repositorio público)

  https://weasis.repositoryhosting.com/git/weasis/weasis-dev.git (login y pass.)
- También se puede acceder mediante *File* > *Import…* > *CVS* > *Project from CVS* (doble click) > introducir la URL del punto anterior.
- Conectar proyectos maven importados a GIT: Selecciona todos los proyectos, click con el botón derecho > *Team* > "Share Project...", selecciona GIT. En el cuadro de dialógo, selecciona "Use or create repository on in parent folder of project".

5. Compilación de los plugins Weasis.

- Seleccionar el proyecto *weasis-framework* (o el pom.xml en el proyecto)
- Botón derecho > *Run us…* > *Maven install.*
- Si Weasis compila correctamente se podrá observar: BUILD SUCCES (Fig. A1.1).

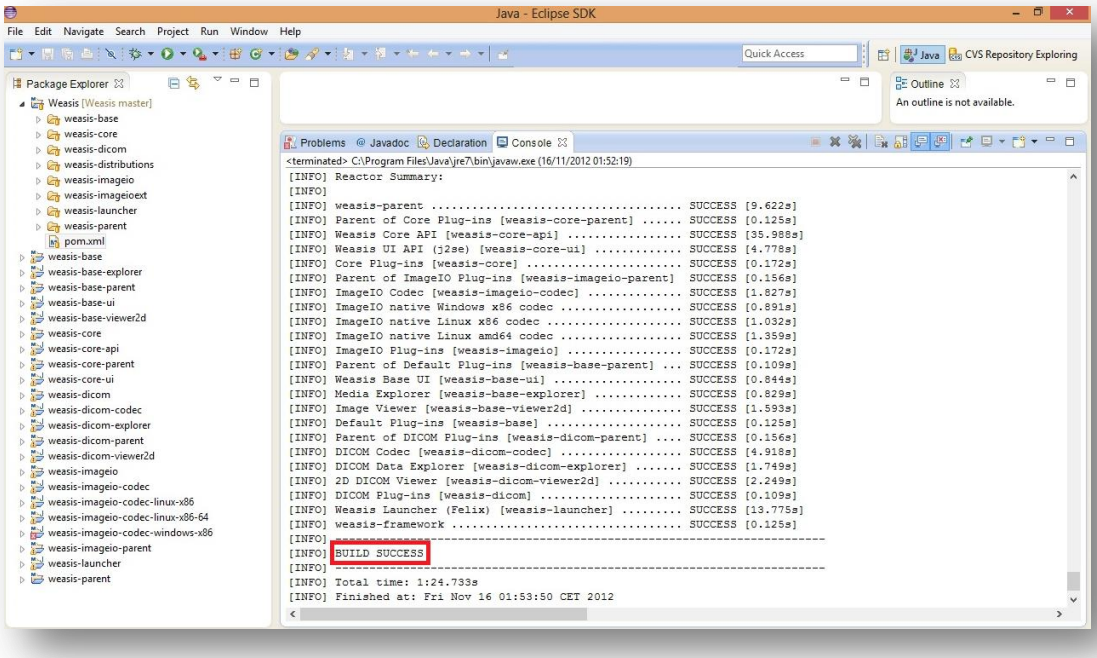


Figura A1.1: Compilación con éxito de Weasis en el entorno Eclipse.

- Una vez realizado el paso anterior, debemos compilar las Weasis-Distributions, para ello, desplegamos el proyecto Weasis-Distribution > click con el botón derecho sobre el pom.xml > *Run...* > *Maven Install.*
- Los archivos que nos genera la instrucción anterior se almacenarán en *Weasis-Distributions/target/web-dist/weasis.zip*.

6. Depuración y ejecución de Weasis.

- Abrir *Run* > *Debug Configurations...*
- Crear una nueva Java Application
- Hacemos click sobre la pestaña *tab*:
  - Project: **weasis-launcher**
  - Main Class: **org.weasis.launcher.WeasisLauncher**
- Hacemos click sobre la pestaña *Arguments* y rellenamos los campos:
  - Program arguments (Ejemplo pasarle un archivo de imágenes DICOM):

$dicom:get -l "C:/Users/Adrian/Desktop/PROYECTO/IMÁGENES

/RM_CARDIO/5I5SMMUD/Y4KTJHHP"

- VM arguments:

-Xms32m -Xmx512m -Dmaven.localRepository=
"C:/Users/Adrian/.m2/repository"-Dgosh.args="-sc telnetd -p 17179 start"

- Other VM arguments:

-Dfelix.config.properties=

file:conf/config.properties-Djava.ext.dirs=""

Nota: Cuando efectuamos alguna modificación en el código fuente, debemos hacer lo siguiente para visualizarla:

- Compilar el fichero pom.xml del módulo del cual hemos cambiado o añadido nuevo código.
- Posteriormente ir a Debug as... > Escoger la configuración (creada en el punto anterior) que deseemos.
- Weasis se lanzará automáticamente con los cambios oportunos.

Estructura de Weasis

En cuanto a la estructura que presenta el código fuente de Weasis, lo más destacable es que se organiza en diferentes módulos (Fig. A1.2).

- El módulo *Viewer or Editor* donde encontramos los ficheros fuente relacionados con la visualización de la imagen y videos.
- En *Main User Interface*, se encuentra el núcleo de la interfaz de usuario, todo lo relacionado con mediciones, variación brillo/contraste, filtros y LUTS, etc.

- Posteriormente encontramos todo lo relacionado con el explorador de imágenes (donde cargando un estudio de cierto paciente, tendremos a nuestra disposición todas las series que lo componen listas para visualizar) en *Data Explorer*.
- En el módulo *Codecs* se encuentra toda lo necesario para el correcto funcionamiento de la entrada y salida de imágenes así como el manejo de todos los datos contenidos en la imagen DICOM.
- Por último, el nexo de unión de todos los módulos anteriores, se puede encontrar en el apartado *Core*.

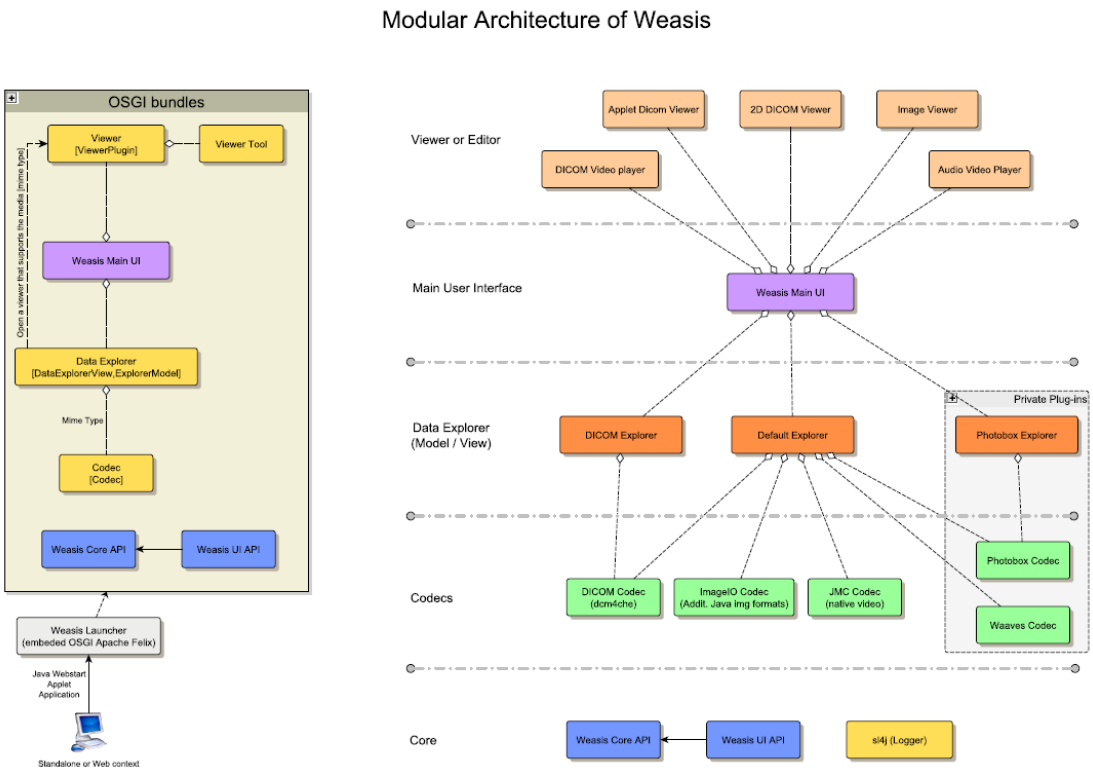


Figura A1.2: Esquema modular de la arquitectura de Weasis.

Anexo II

Instalación de la aplicación GHQdicom

La carpeta que se proporciona contiene dos versiones del software, una versión destinada al usuario y la versión para el programador.

Versión destinada a programadores

A continuación se describe la versión que se ha proporcionado para toda persona que desee consultar/modificar el código fuente de la aplicación o ampliarla mediante la inserción de *plugins.*

Al abrir el directorio "GHQdicom Programador" se mostrará la carpeta que contiene el proyecto realizado, entrando en ella, se pueden observar dos directorios y dos archivos. El proyecto recibe el nombre de "acgdicom" (nombre de la clase principal) y aparece con el icono propio de Visual Studio 2008. El otro archivo que encontramos posee extensión ".ncb" y lo utiliza el sistema para la ejecución del programa por lo que no es recomendable modificarlo ni eliminarlo. El directorio "Win32", contiene la última compilación de la aplicación en modo Debug (para programadores) y modo Release (para usuarios). Por último, el directorio que se denomina como el proyecto posee todo el contenido del mismo (ficheros de las diferentes clases e imágenes que se utilizan para la interfaz de usuario).

Cabe destacar que el software ha sido desarrollado para ser compilado y ejecutado en Windows, pero existe la posibilidad de hacerlo en otra plataforma modificando las partes correspondientes del código fuente. Esta propiedad es posible

puesto que las librerías utilizadas (ITK y Qt), poseen versión disponible para Linux y MAC.

Versión destinada a usuarios

Si se desea instalar el software, se debe abrir la carpeta “GHQdicom Usuario” y hacer doble *click* sobre “setupGHQdicom” lanzando de esta forma el asistente de instalación (Fig. A2.1).

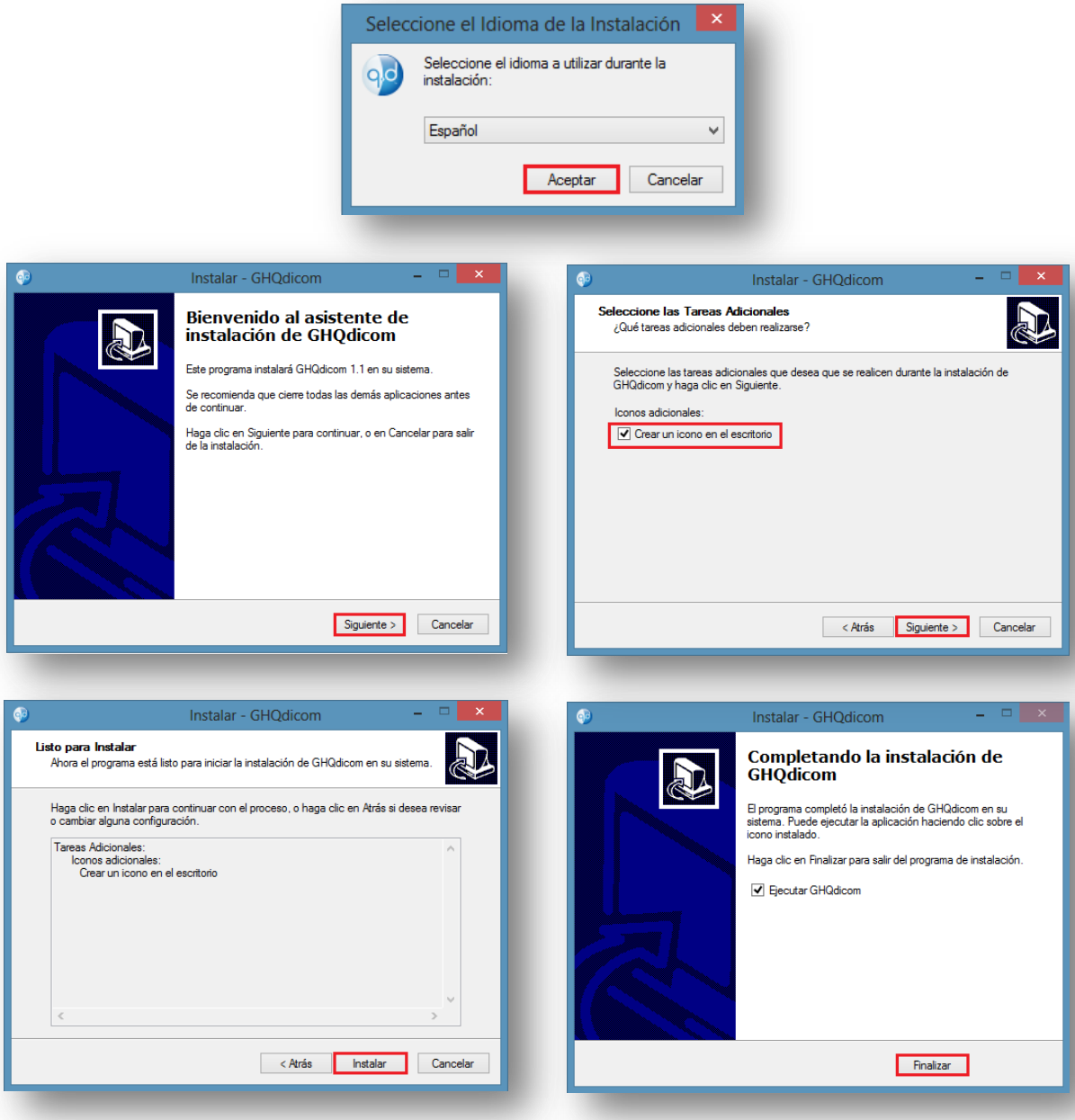


Figura A2.1: Pasos a seguir para la instalación de GHQdicom.

Una vez instalada la aplicación, se crea un acceso directo en el escritorio cuyo icono es el de cualquier aplicación Windows por defecto. Por esta razón, se adjunta el logotipo de GHQdicom en el mismo directorio que se encuentra el *setup*. Si se desea cambiar el icono una vez instalada la aplicación, se deben seguir los siguientes pasos: Se presiona botón derecho sobre el acceso directo y se escoge la opción "Propiedades". A continuación, se pulsa el botón "Cambiar icono" y en el cuadro emergente se selecciona la opción "Examinar…". Por último, una vez se abre el explorador de Windows, buscamos el logotipo que se proporciona pulsando posteriormente "Aceptar". El icono habrá cambiado y ya estamos a disposición de utilizar la herramienta.

Bibliografía

[1] Alberich-Bayarri A. The Engineer's Role in Medical Imaging and Research: a Real Need. IEEE GoldRush.2012; Sep-Dec:p.18.

[2] Martí Bonmatí L, Alberich-Bayarri A, García-Martí G, Sanz Requena R, et al. Biomarcadores de imagen, imagen cuantitativa y bioingeniería. Radiología. 2011. Doi:10.1016/j.rx.2010.12.013.

[3] Millet Roig J, Guillem Sánchez M. Tema 7 de la asignatura Señales e Imágenes Biomédicas: Imagen por Resonancia Magnética. Universidad Politécnica de Valencia, ETSIT.

[4] Concha G. Conceptos básicos sobre imágenes por Resonancia Magnética.

[5] Martínez Latorre M.T. Memoria Proyecto Final de Carrera 'Herramienta software para la segmentación y visualización 3D de hígados reales a partir de imágenes de escaneos 3D no invasivos'. Universidad Politécnica de Valencia, Escuela Politécnica Superior de Gandía, 2010.

[6] Sanz Requena R, Ródenas García J, Martí Bonmatí L, Alberich-Bayarri A, García-Martí G, Flórez Ordoñez N. Cuantificación de estudios dinámicos y compartimentales.

[7] Martí Bonmatí L, Sanz Requena R, Alberich-Bayarri A. Nuevos horizontes en la imagen en oncología.

[8] Sección de descripción del software ImageJ. Disponible en http://rsbweb.nih.gov/ij/

[9] Sección de descripción del software Gimias. Disponible en http://www.gimias.net/

[10] Sección de descripción del software Weasis. Disponible en http://www.dcm4-che.org/confluence/display/WEA/Home.

[11] J. Blanchette, M. Summer_eld C++ GUI Programming with Qt 4 (2nd Edition)
Prentice Hall Open Source Software Development Series, 2006.

[12] Asistente de ayuda del software Qt (Qt Assistant) disponible en http://qt-project.org/search/tag/qt~assistant.

[13] Ibáñez L, et al. The ITK Software Guide Second Edition Updated for ITK version 2.4, 2005.

“Si el camino es difícil, normalmente es porque vas en la dirección correcta”

Anónimo