EPS
Escola Politècnica Superior
UdG

Projecte/Treball Fi de Carrera

Estudi: Eng. Tècn. Informàtica de Gestió. Pla 2001

Títol: Entorn de suport a la inserció de pròtesis

Document: Memòria

Alumne: Marc Costa Garriga

Director/Tutor: Imma Boada
Departament: Informàtica i Matemàtica Aplicada
Àrea: LSI

Convocatòria (mes/any): 09/09

Índex

1. Introducció

1. Motivació

El diagnòstic per la imatge s'ha convertit en una eina bàsica de treball de qualsevol centre hospitalari. Les imatges que s'obtenen a partir de radiografies, tomografies computades o ressonàncies magnètiques, entre d'altres, permeten obtenir informació de l'interior de l'organisme del pacient. Els radiòlegs, analitzen aquestes imatges usant programes especialitzats de processament i visualització i realitzen un diagnòstic, el qual serà enviat a l'especialista que ha demanat l'exploració. En aquest context els radiòlegs són els usuaris principals dels programes de visualització d'imatges.

A l'Hospital Universitari de Girona Josep Trueta aquesta situació però ha canviat en aquests darrers mesos ja que s'ha posat en marxa el Pla de Digitalització de la Imatge endegat pel Departament de Salut. Aquest pla elimina la possibilitat d'imprimir imatges, tot es farà de forma digital. Això tindrà efectes sobre tots els especialistes que normalment treballen amb imatges. D'entre tots els especialistes afectats en aquest projecte ens centrarem en els traumatòlegs.

Fins ara el traumatòleg diagnosticava i planificava intervencions a partir de l'estudi de plaques radiogràfiques. Sobre aquestes plaques es realitzen operacions de càlcul de distancies, angles, etc. per determinar la gravetat de fractures i decidir si cal o no realitzar intervencions. En el cas de decidir intervenir es realitzen simulacions sobre la placa per determinar la mida de la pròtesi. Per poder satisfer les noves necessitats del traumatòlegs cal desenvolupar eines de software especialitzades capaces de reproduir totes les operacions que fins ara es feien de forma manual.

Figura 1 Càlcul de distàncies en un canell dislocat

Figura 2 Càlcul d'angles en un canell dislocat

2. Antecedents

El Laboratori de Gràfics i Imatge (GILab) de la Universitat de Girona ha estat col·laborant durant anys amb l'Institut de Diagnòstic per la Imatge (IDI) de l'hospital Dr. Josep Trueta de Girona en el desenvolupament d'una plataforma informàtica de processament i visualització de imatges mèdiques per donar suport al diagnòstic. Aquesta plataforma es coneix amb el nom d'Starviewer.

La plataforma Starviewer integra las funcionalitats bàsiques que cobreixen el 99% de les operacions que ha de realitzar un radiòleg. Té un disseny modular que permet la integració de noves funcionalitats. A més a més la plataforma permet la integració de mòduls especialitzats dissenyats per diagnosticar patologies més concretes. Alguns d'aquests mòduls són l'StarStroke o l'StarColon que donen suport a diagnòstic d'infarts i càncer de colon respectivament. La modularitat de la plataforma permet incorporar nous mòduls de forma fàcil en funció de les demandes dels especialistes del diagnòstic.

Tot i les funcionalitats que ofereix la plataforma s'ha fet palès que no satisfà completament les necessitats dels traumatòlegs. La principal limitació és la poca versatilitat de les eines de mesura. D'altra banda tampoc es suporten les diferents operacions que normalment es realitza sobre una placa per poder simular la inserció d'una pròtesi. Per posar fi a aquestes limitacions ens proposem els objectius que presentem a continuació.

3. Objectius

L'objectiu principal d'aquest projecte és integrar a la plataforma Starviewer un entorn de suport a la inserció de pròtesis, que permeti automatitzar al màxim les operacions que actualment es realitzen de forma manual. Hem de tenir en compte que, tot i que, la imatge més usada pel radiòleg es la radiografia (Rx), també treballa amb tomografia computada (TAC). El TAC dona una visió 3D de l'organisme, mentre que la Rx és 2D.

Per assolir aquest objectiu s'han analitzat les diferents operacions que realitza el traumatòleg.

Aquestes es mostren en les imatges de les Figures següents. Primer (veure Figura 3) es localitza la zona en la que hi ha la lesió, en aquest cas en la part superior del fèmur. Com es pot veure, a més de trencament hi ha un desplaçament de l'os, per tant cal inserir una pròtesi per tornar-lo a la posició correcta i que es consolidi de nou. Les següents operacions que realitza el traumatòleg tenen com a objectiu determinar com col·locar la pròtesi. El primer que fa és identificar la zona en la que hi ha la fractura tal com es veu en les Figures Figura 4 i Figura 5. En una identifica la part inferior del fèmur i en l'altre la part superior. A continuació desplaça les zones seleccionades fins col·locar-les en la posició correcte (veure Figura 6 i Figura 7). Després de fer aquesta operació interessa controlar que la posició en la qual han quedat és la correcta i, per això, el que es fa es sobreposar-la sobre l'altra extremitat. Finalment, es col·loca la pròtesi (veure Figura 8).

Figura 3 Identificació de la lesió

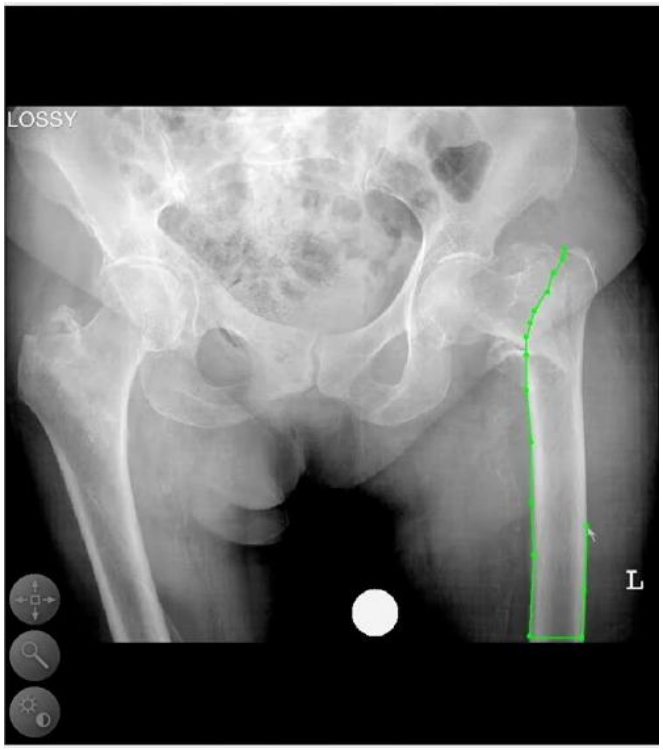


Figura 4 Retallar primer os a desplaçar

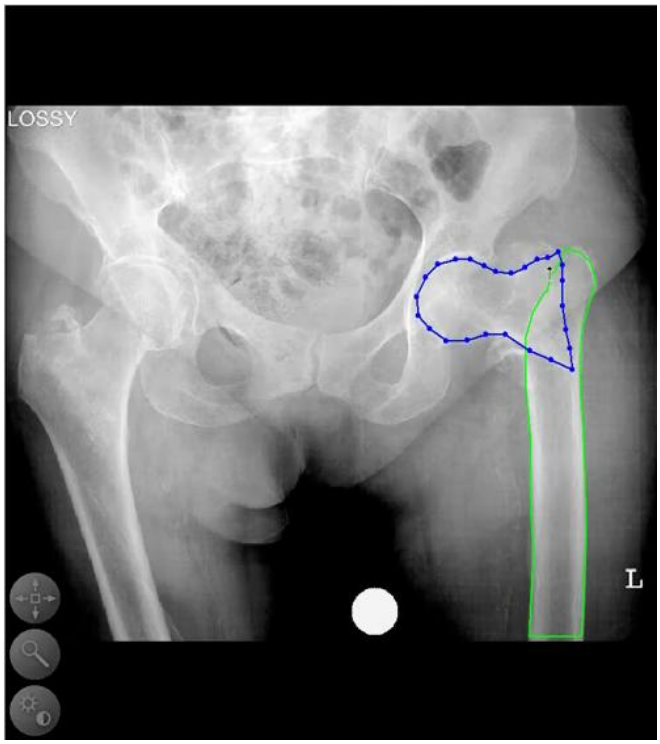


Figura 5 Retallar segon os a desplaçar

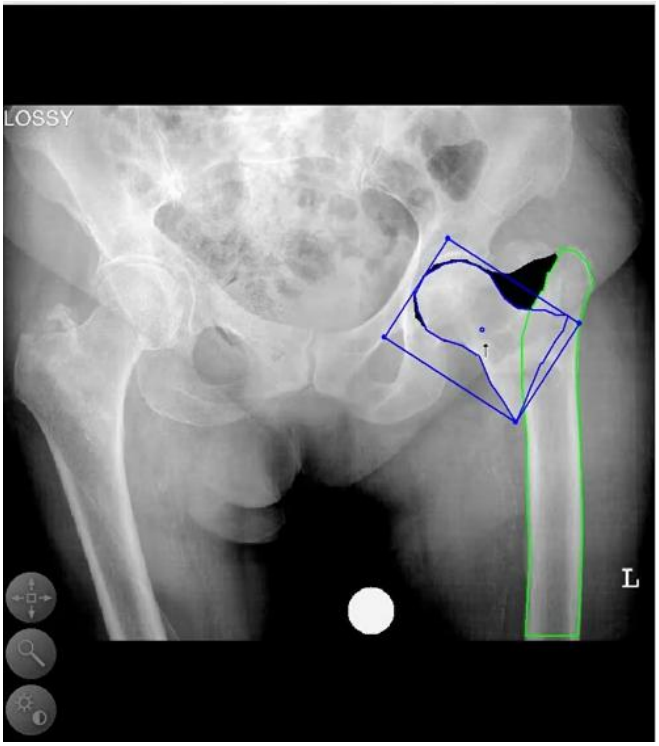


Figura 6 Desplaçar os retallat

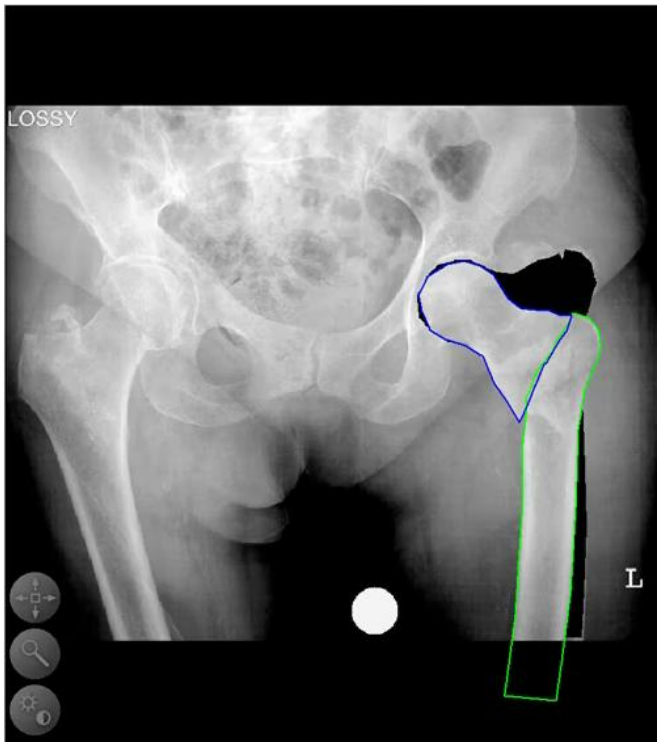


Figura 7 Desplaçar l'altre os desplaçat

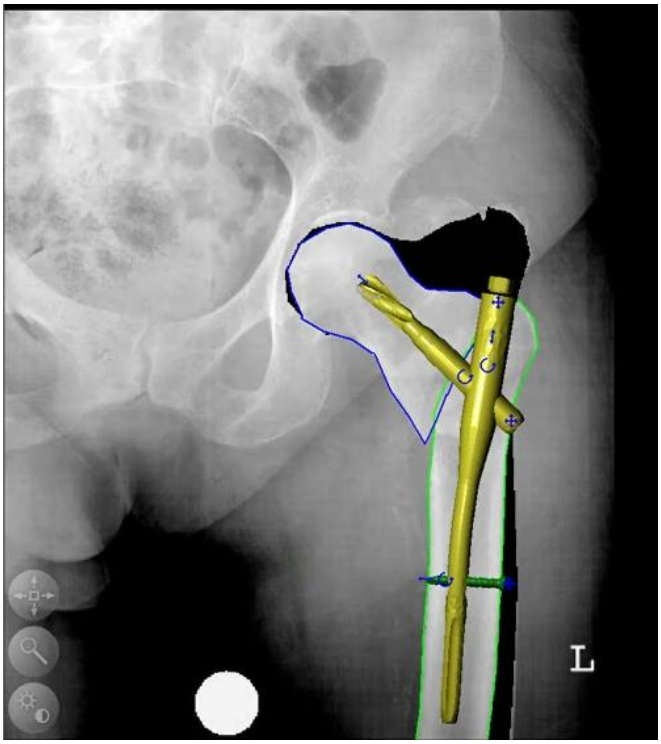


Figura 8 Inserir pròtesis

Tenint en compte aquestes operacions s'han definit el següents objectius específics.

- Estudiar, dissenyar i implementar les operacions bàsiques de mesures, definició de regions d'interès, càlcul d'angles, etc. de forma que puguin aplicar-se sobre imatges de Rx i TAC. Aquest objectiu es correspondria al pas 1 del procés de la Figura….
- Eines de retallat que siguin aplicables sobre Rx. Aquestes tècniques s'usen per calcular el grau de solapament entre dues parts d'un os fracturat. Aquest objectiu es correspondria al pas 1 del procés de la Figura….
- Una tècnica de representació de la informació relacionada amb les pròtesis (plantilles de forma, la pròpia pròtesis, etc) de forma que el traumatòleg en funció de les operacions realitzades prèviament sàpiga quina és la pròtesi més adequada. Aquest objectiu es correspondria al pas 1 del procés de la Figura….
- Una tècnica que permeti visualitzar com quedarà col·locada la pròtesi, permetent interaccionar amb aquesta visualització. Aquest objectiu es correspondria al pas 1 del procés de la Figura….
- Estudiar, dissenyar i implementar alguna tècnica que permeti optimitzar la manipulació dels volums generats a partir de TACs. En el cas de lesions de pelvis, per exemple, el traumatòleg només està interessat en representar un part de tot el model de volum que es reconstrueix a partir del TAC. Per aconseguir-ho calen tècniques especials de manipulació de volums.

4. Metodologia

La plataforma Starviewer està dissenyada sobre el paradigma de l'Orientació a Objectes i seguint la metodologia de l'*eXtreme Programming* (XP).

L'*eXtreme Programming* és una metodologia àgil basada en quatre principis: simplicitat, comunicació, retroalimentació i valor. A més, orientada per proves i *refactorització*[1], es dissenyen i implementen les proves abans de programar la funcionalitat.

Els objectius de l'XP són grups petits i mitjans de construcció de software on els requisits encara són molt ambigus, canvien molt ràpidament o són d'alt risc. L'XP busca la satisfacció del client intentant mantenir al llarg del temps la seva confiança en el producte. A més, suggereix que el lloc de treball sigui una sala àmplia, a ser possible sense divisions (al centre els programadors, a la perifèria els equips individuals). Un avantatge de l'espai obert és l'augment de la comunicació i proporcionar una agenda dinàmica al voltant de cada projecte.

4.1. Activitats de l'eXtreme Programming

Codificar

És necessari codificar i plasmar les idees a través del codi. En programació, el codi expressa la interpretació del problema, així es pot utilitzar el codi per comunicar, per fer comuns les idees, i, per tant, aprendre i millorar.

Fer proves

Les característiques del software que no poden ser demostrades mitjançant proves simplement no existeixen. Les proves donen l'oportunitat de saber si allò implementat és el que en realitat es tenia en ment. Les proves indiquen que la feina funciona, quan no es pugui pensar en cap prova que pogués provocar un error en el sistema, llavors s'haurà acabat del tot.

Escoltar

Si s'han de fer proves s'ha de preguntar si allò obtingut és allò desitjat, i s'ha de preguntar a qui necessita la informació. S'han d'escoltar els clients, entendre els problemes del negoci, i explicar el que és fàcil o difícil d'obtenir. Aquesta retroalimentació ajuda a tothom a entendre els problemes i així facilitar la seva solució.

Dissenyar

El disseny crea una estructura que organitza la lògica del sistema. Un bon disseny permet que el sistema creixi amb canvis en un sol lloc. Els dissenys han de ser senzills. Si una part del sistema necessita un desenvolupament complex és millor dividir-la en diverses. Si hi ha errors en el disseny o mals dissenys s'han de corregir el més ràpid possible.

[1] Tècnica per reestructurar el codi font, aterant la seva estructura interna sense canviar-ne el comportament.

Resumint les activitats de l'XP:

- S'ha de codificar perquè sense codi no hi ha programes.
- S'han de fer proves perquè sense aquestes no es pot saber si s'ha acabat de codificar.
- S'ha d'escoltar perquè sense això no se sap què s'ha de codificar i provar.
- I s'ha de dissenyar per poder codificar, provar i escoltar indefinidament.

4.2. Pràctiques bàsiques de l'eXtreme Programming

De forma aïllada, qualsevol pràctica individual de l'XP té poc sentit, però en conjunt, unes compensen les carències que la resta puguin tenir.

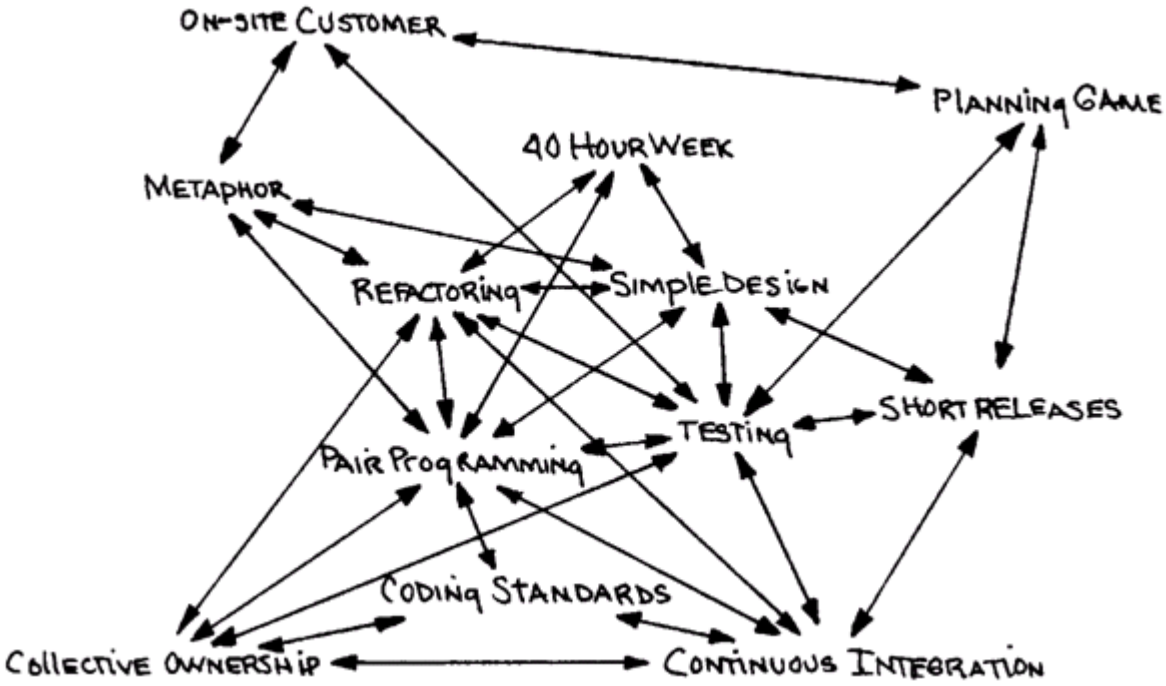


Figura 9 Pràctiques de l'eXtreme Programming

El joc de la Planificació (Planning Game)

L'abast de la següent versió està definit per les consideracions de negocis (prioritat dels mòduls, dates d'entrega) i estimacions tècniques (funcions, conseqüències).

L'objectiu del joc és maximitzar el valor del software produït. L'estratègia és posar en producció les característiques més importants el més aviat possible.

Petites versions (Short Releases)

Un sistema simple es posa ràpidament en producció. Periòdicament, es produeixen noves versinos agregant en cada iteració aquelles funcions considerades valuoses pel client.

Metàfora del Sistema (Metaphor)

Cada projecte està guita per una història simple de com funciona el sistema en general, substitueix l'arquitectura i ha d'estar en llenguatge comú, entès per tots (Desenvolupadors i Client). Aquesta pot canviar permanentment.

Disseny simple

El sistema es dissenya amb la màxima simplicitat possible[2]. Es plasma el disseny en targetes CRC (Classe – Responsabilitat – Col·laboració) i no s'implementen característiques que no són necessàries.

Proves contínues (Testing)

Els casos de proves s'escriuen abans que el codi. Els desenvolupadors escriuen proves unitàries i els clients especifiquen proves funcionals.

Refactorització (Refactoring)

És possible reestructurar el sistema sense canviar-ne el comportament, per exemple eliminant codi duplicat, simplificant funcions, etc. Si el codi s'està tornant complicat, caldria modificar el disseny i tornar a una de més simple.

Programació per parelles (Pair Programming)

El codi s'escriu per dues persones treballant en el mateix lloc. "Una sola màquina amb un teclat i un mouse".

Possessió col·lectiva del codi (Collective Code Ownership)

Ningú és amo d'un mòdul. Qualsevol programador pot canviar qualsevol part del sistema en qualsevol moment, sempre s'utilitzen estàndards i s'exclouen els comentaris. Els testos sempre han de funcionar al 100% per realitzar integracions amb tot el codi permanentment.

Integració contínua (Continuous Integration)

Els canvis s'integren al codi base diverses vegades al dis. Tots els casos de prova s'han de passar abans i després de la integració, es disposa d'una màquina per a la integració i es realitzen testos funcionals on hi participa el client.

Setmana labolar de 40 hores (40-Hour Week)

Cada treballador treballa coma màxim 40 hores per setmana. Si fos necessari fer hores extra, això no s'hauria de fer dues setmanes consecutives. Sense herois, això fa que es redueixi la rotació del personal i millora la qualitat del producte.

[2] YAGNI: You Ain't Gonna Need It (no ho necessitaràs)

Client al lloc (On Site Customer)

L'equip de desenvolupament té accés, sempre, al client, qui està disponible per respondre preguntes, fixar prioritats, etc. Això no sempre és possible, però.

Estàndards de codificació (Coding Standard)

Tot el codi ha d'estar escrit seguint un estàndard de codificació.

4.3. Cicle de vida

El cicle de vida de l'XP s'emfatitza en el caràcter interactiu i incremental del desenvolupament. Una iteració de desenvolupament és un període de temps en el que es realitza un conjunt de funcionalitats determinades que en el cas de l'XP corresponen a un conjunt d'històries d'usuaris.

Les iteracions són relativament curtes ja que es considera que com més ràpid se li entreguin desenvolupaments al client, més retroalimentació s'obtindrà i això representarà una millor qualitat del producte a llarg termini. Hi ha un fase d'anàlisi inicial orientada a programar les iteracions de desenvolupament i cada iteració inclou disseny, codificació i proves. Fases superposades de manera que no se separin en el temps.

La següent figura mostra les fases en les que se subdivideix el cicle de vida XP:

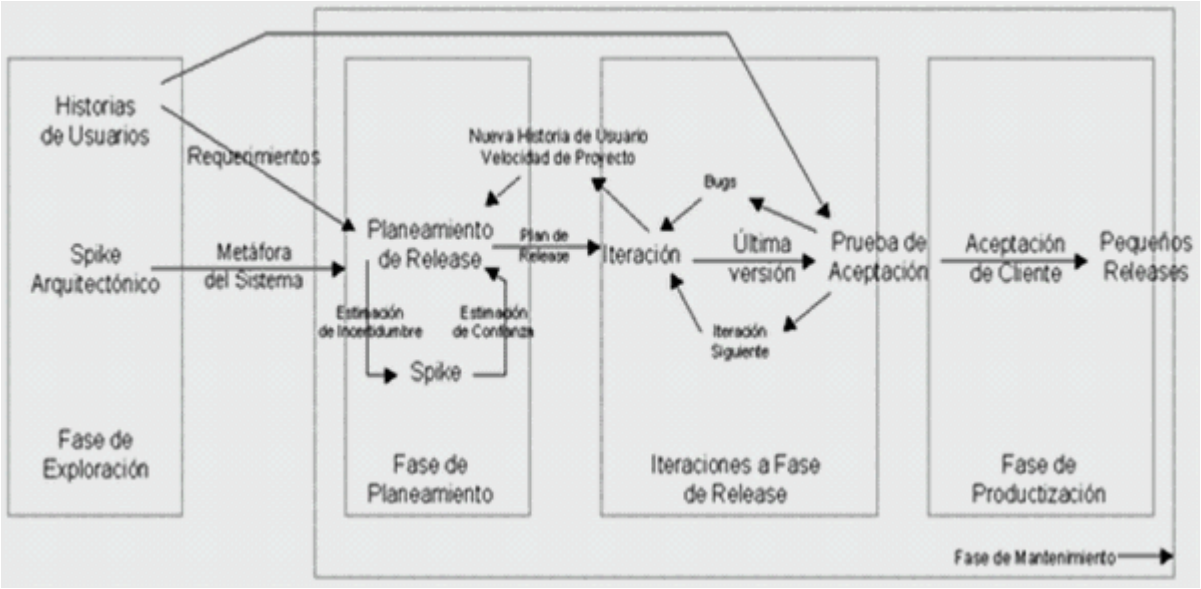


Figura 10 Cicle de Vida de l'eXtreme Programming

Fase d'exploració

En aquesta fase, els clients plantegen a grans trets les històries d'usuari que són d'interès per la primera entrega del producte. Al mateix temps l'equip de desenvolupament es familiaritza amb les eines, tecnologies i pràctiques que s'utilitzaran en el projecte.

Es prova la tecnologia i s'exploren les possibilitats de l'arquitectura del sistema construint un prototip. La fase d'exploració es pot estendre entre poques setmanes i pocs mesos, depenent de la familiaritat que tinguin els programadors amb la tecnologia.

Fase del plantejament

Es prioritzen les històries d'usuari i s'acorda l'abast del *release*. Els programadors estimen quant esforç requereix cada història i a partir d'allà es defineix el cronograma. La primera iteració crea un sistema amb l'arquitectura del sistema complet.

Fase de producció

Requereix prova i comprovació extra del funcionament del sistema abans que aquest pugui ser ofert al client. En aquesta fase, encara es poden trobar nous canvis i s'ha de prendre la decisió de si s'inclouen o no en la *release* actual. Després de realitzar la primera *release* productiva per a ús del client, el projecte ha de mantenir el funcionament del sistema mentre es realitzen noves iteracions.

Fase de manteniment

Requereix més esforç per satisfer les demandes del client. La velocitat del desenvolupament pot descendir durant aquesta fase. A més, pot requerir la incorporació de nova plantilla i canviar l'estructura de l'equip.

Fase de mort

Es dóna quan el client no té més històries per ser incloses en el sistema. Això pot significar satisfer altres necessitats del client en altre aspectes com rendiment i fiabilitat del sistema. Es genera la documentació final i no es realitzen més canvis en l'arquitectura. La mort del projecte també es dóna quan el sistema no genera els beneficis esperats pel client o quan no hi ha pressupost per mantenir-lo.

4.4. Adaptació de la metodologia

En el meu cas no s'ha aplicat la metodologia al 100%.

Per començar, el cicle de vida ja estava avançat: en fase de manteniment. El que he hagut de fer en aquest cas és desenvolupar noves funcions del sistema, però el sistema ja estava en producció de feia anys.

Les activitats sí que s'han mantingut: codificar, dissenyar, escoltar i fer proves.

I, pel que fa al tema de les pràctiques bàsiques, cal destacar que no he hagut de passar per la de planificació. Però tota la resta es mantenen, a excepció d'algunes. Al ser un projecte de caràcter personal i d'avaluació, la programació ha estat individual i no en parelles tal com resa la metodologia de l'XP. Tot i això, tots els moviments s'han comentat amb la resta del grup o algun representant.

M'he integrat a l'equip de desenvolupament de la plataforma Starviewer i, a part de les reunions amb la tutora per decidir l'abast del projecte i avaluar el procés, he mantingut reunions de treball amb l'equip per ajustar el canvis que s'havien de fer i els camins que s'havien de prendre.

5. Pla de Treball

El pla de treball seguit en aquest projecte s'ha dividit en dues parts.

La primera part: teòrica i formació. Adquisició dels coneixements previs a la implementació.

- Instal·lar les eines bàsiques de suport a la plataforma de treball.
- Instal·lar la plataforma de treball i el codi font.
- Familiaritzar-se amb la plataforma i els mòduls a tractar.
- Familiaritzar-se amb les principals llibreries utilitzades en la implementació de la plataforma: ITK (Insight Toolkit), VTK (Visualization Toolkit) i Qt.

Un cop acabada l'etapa de formació, comença el disseny i implementació de la solució. Aquesta és la segona part del pla de treball. El desenvolupament d'aquesta part i seguint la metodologia *eXtreme Programming*, s'ha fet d'aquesta manera:

- Pensar i dissenyar diferents solucions per fer prou versàtil el mòdul de les eines perquè serveixin a les noves necessitats.
- Decidir, conjuntament amb els responsables de la plataforma, la solució més adient. La modificació dels mòduls afectats i la implementació de noves eines han d'anar integrats a la plataforma, per tant s'ha de seguir la filosofia d'aquesta i respectar el disseny original.
- Implementar la solució més adient.
- Els testos de les noves funcionalitats s'han realitzat sobre la mateixa plataforma, donat que els canvis s'han implementat directament sobre aquesta.

6. Estructura del document

A part d'aquest capítol introductori on s'explica tot allò referent a abans de començar el disseny i implementació dels objectius del projecte en sí, en aquest document hi apareixen 6 capítols més.

La llista d'aquests amb una breu descripció ve a continuació:

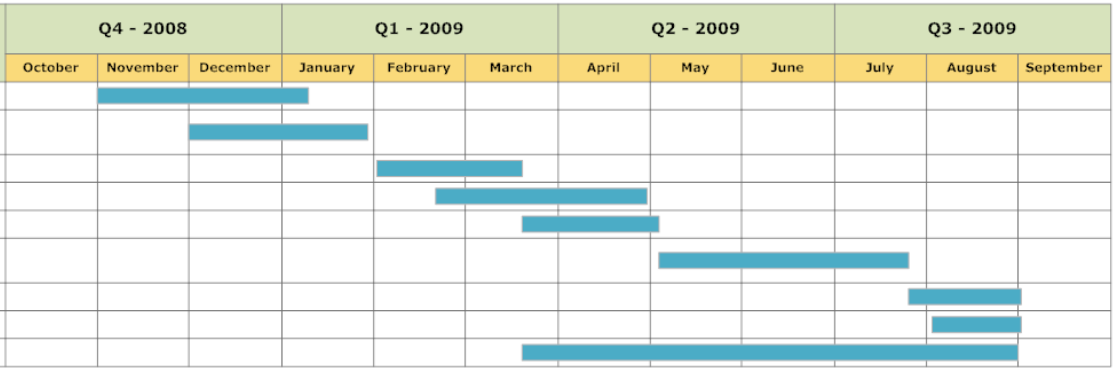
- Marc de Treball: explicació de la plataforma que allotja la implementació duta a terme en aquest projecte, a més de les llibreries utilitzades i estudiades.
- Starviewer Tool's: explicació de la modificació del mòdul de `Tool`'s de l'aplicació.
- ExtractImageTool: explicació de la implementació d'una nova eina. Aquesta s'utilitza per retallar parts de les imatges.
- SuperimposeImageTool: explicació de la implementació d'una nova eina. Aquesta s'utilitza per sobreposar una imatge sobre l'anterior invertida horitzontalment.
- Conclusions: breu repàs de la introducció i els objectius, així com de la valoració general de l'experiència del projecte.
- Bibliografia: llistat de llibres i pàgines web utilitzades durant el desenvolupament d'aquest projecte.

7. Calendari

El següent diagrama mostra la durada de cada etapa del projecte. Cal remarcar que el diagrama ha estat actualitzat a mesura que passava el temps i les etapes s'allargaven.

Project: PFC

Number	Task	Start	End	Duration
1	Estudi de les llibreries ITK, VTK i Qt.	1/11/2008	10/1/2009	50
2	Instal·lació de Linux, Starviewer i compilació de les llibreries.	1/12/2008	30/1/2009	44
3	Estudi de la plataforma Starviewer.	2/2/2009	20/3/2009	34
4	Estudi del mòdul de Tool's.	20/2/2009	30/4/2009	49
5	Estudi de solucions per la modificació del mòdul de Tool's.	20/3/2009	4/5/2009	31
6	Disseny, implementació i proves del mòdul de Tool's modificat.	4/5/2009	26/7/2009	60
7	Disseny, implementació i proves de ExtractImageTool.	26/7/2009	2/9/2009	27
8	Disseny, implementacio i proves de SuperimposeImageTool.	3/8/2009	2/9/2009	22
9	Redacció de la memòria del projecte.	20/3/2009	1/9/2009	117



2. Marc de Treball

Per a la realització d'aquest projecte s'utilitza una aplicació en funcionament: la plataforma Starviewer. A més, s'ha necessitat documentar-se sobre diferents llibreries de suport al desenvolupament, a saber ITK, VTK i Qt, que s'explicaran més endavant.

1. La plataforma Starviewer

STARVIEWER és una aplicació de diagnòstic capaç de combinar, en el mateix entorn, la interfície de diagnòstic habitual amb una aplicació de recerca més específica.

STARVIEWER

- Integra eines bàsiques i avançades
- Disseny modular que permet integrar noves funcionalitats
- Disponible en els sistemes operatius Windows i GNU/Linux
- Suporta l'estàndard DICOM i perfils IHE

La plataforma és el resultat de combinar el coneixement dels tècnics en recerca en Gràfics i Imatge, de la Universitat de Girona, amb l'experiència dels radiòlegs de l' Institut de Diagnòstic per la Imatge, un institut d'imatge prestigiós situat en els principals hospitals públics catalans.

1.1. Descripció de l'aplicació

L'aplicació és pot desglossar en certs blocs funcionals. Cada bloc d'aquests s'encarrega d'una funció principal. Aquests blocs es poden apreciar a la següent figura, on es mostren en un diagrama de classes d'alt nivell que il·lustra l'arquitectura més bàsica de l'aplicació.

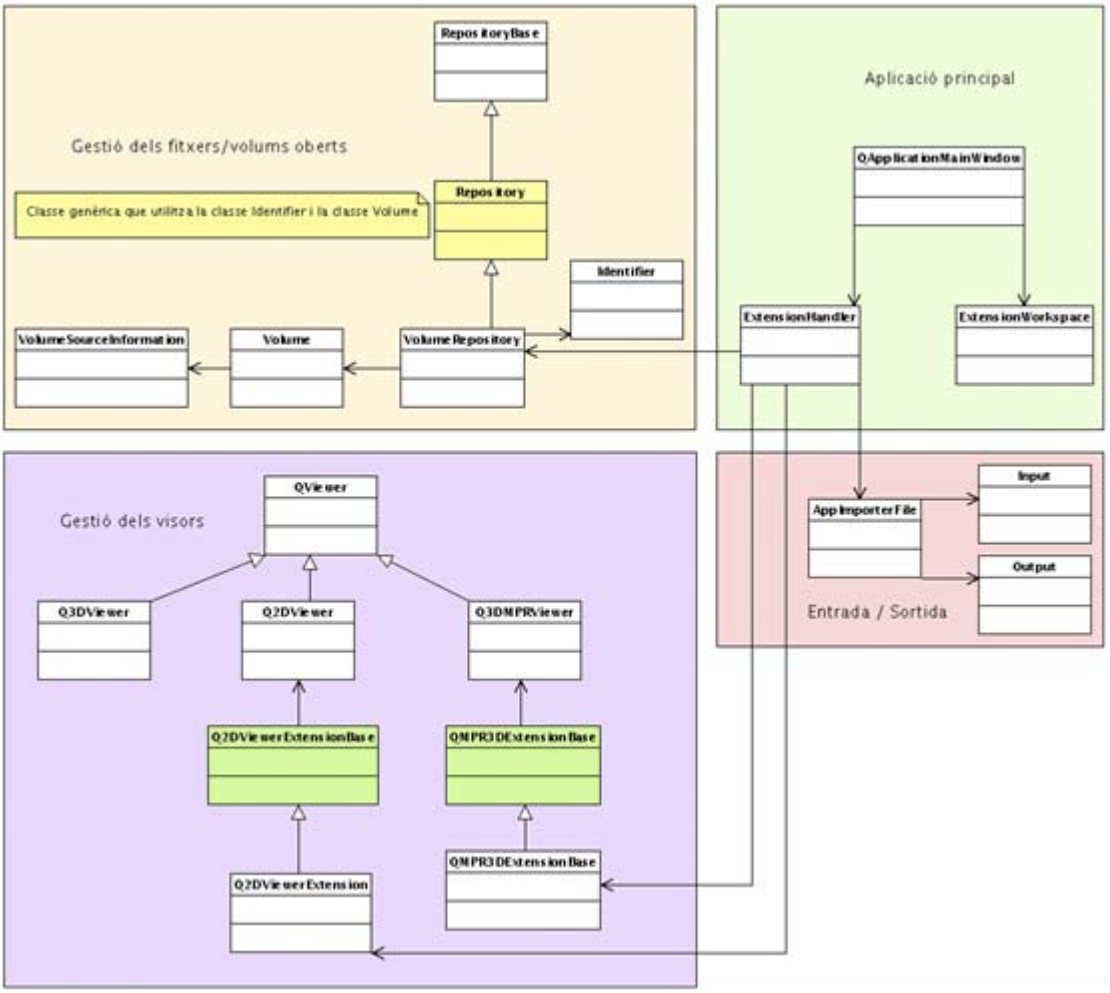


Figura 11 Blocs principals de la plataforma Starviewer

1.1.1. Descripció dels blocs principals

Aplicació principal

Proporciona la interfície bàsica i s'encarrega de gestionar la obertura de diferents visors.

Entrada/Sortida

Aquest mòdul permet la importació i exportació d'exploracions.

Gestió de fitxers i volums oberts

Tota informació gràfica que s'utilitza a l'aplicació es guarda internament com a `Volume`. Aquest bloc s'encarrega de gestionar la informació relacionada a aquests volums i els propis volums.

Gestió de visors

L'aplicació disposa de diferents visors per a poder analitzar els volums oberts: 2D, 3D, etc. Cada visor s'encarrega d'una visualització en concret.

En aquest projecte es treballa sobre una part del bloc **gestió de visors**. Aquesta part és la de **Tool**'s.

Les Tool's són aquells elements que ens permeten interactuar amb els volums oberts i en visualització en algun dels visors. Aquestes s'encarreguen per exemple de canviar d'imatge d'un volum en una visualització en 2D, de fer *zoom*, de moure la imatge pel visor i de fer certes mesures sobre aquests volums: distàncies, angles, mitjanes de gris en certes àrees, etc.

L'estructura d'aquesta part és la següent:

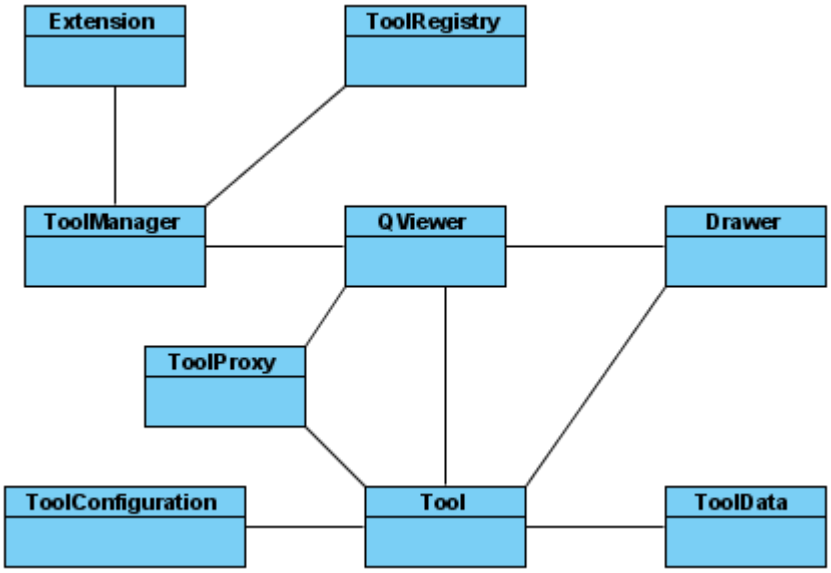


Figura 12 Diagrama de classes de la part de Tool's

En aquest diagrama es pot apreciar la presència de classes del bloc de **gestió de visors** com ara `QViewer` o `Extension`.

2. Les llibreries

La plataforma Starviewer utilitza en gran mesura 3 llibreries, que són les que he hagut d'estudiar i familiaritzar-me abans de començar el disseny i la implementació. En afegir funcionalitats a l'aplicació he hagut d'utilitzar el llenguatge en què està implementada la plataforma: C++ i comptar amb l'ajuda de les següents llibreries:

- Insight Toolkit (ITK), versió 3.6.0: sistema de processat d'imatges, segmentació i registre.
- Visualization Toolkit (VTK), versió 5.0.x: sistema de visualització de gràfics per computador.
- Qt (pronunciat com la paraula anglesa "cute"), versió 4.5.x: sistema de creació d'aplicacions amb interfície gràfica.

Totes aquestes llibreries són de codi lliure i multi-plataforma. A més, utilitzen el llenguatge C++ per a la seva implementació i segueixen el paradigma de l'orientació a objectes. Tot això fa que sigui molt fàcil integrar-les totalment a l'aplicació, la plataforma Starviewer.

El desenvolupament del projecte s'ha fet sobre els sistemes operatius de Linux Ubuntu 8.04, Linux Mandriva 2007 Spring, Windows XP 64 bits i Windows Vista 64 bits. L'entorn de programació utilitzat ha estat el KDevelop en Linux i Microsoft Visual C++ 2008 Express Edition en Windows.

A continuació, s'expliquen més detalladament les llibreries utilitzades.

2.1. Insight ToolKit (ITK)

ITK és una llibreria de codi lliure per a la segmentació i el registre. La segmentació és el procés d'identificar i classificar dades que es troben en una representació digital. Normalment, aquesta representació és una imatge adquirida d'algun instrument mèdic com els escàners de CT (Computated Tomography) o MRI (Magnetic Ressonance Imaging). El registre és la tasca d'alinear o desenvolupar correspondències entre les dades. Per exemple, en el terreny mèdic, un escaneig de CT pot estar alineat amb un MRI per combinar la informació continguda en ambdós.

ITK està implementat en C++. ITK és multi-plataforma, utilitzant l'entorn de construcció CMake per gestionar el procés de configuració. A més, un procés de *wrapping* automàtic genera interfícies entre C++ i llenguatges de programació interpretats com Tcl, Java i Python. Això permet als desenvolupadors crear software amb varietat de llenguatges de programació. L'estil de programació de ITK amb C++ és identificat amb la programació general (per exemple, utilitzant codi de plantilles). Aquestes plantilles de C++ permeten un codi altament eficient. També permet que molts problemes de software siguin descoberts en temps de compilació, molt millor que en temps d'execució, durant l'ús de l'aplicació.

Com que ITK és un projecte de codi lliure, desenvolupadors d'arreu del món poden utilitzar, corregir, mantenir i estendre el software. ITK utilitza un model de desenvolupament de software identificat com a *eXtreme Programming*. Aquest concentra la metodologia de creació de software més usual dins un procés simultani i iteratiu de disseny-implementació-proves-lliurament. Les característiques clau de l'*eXtreme Programming* són la comunicació i les proves. La comunicació entre els membres de la comunitat ITK és el que ajuda a gestionar la ràpida evolució del software. Provar és el que manté el software estable.

2.2. Visualization Toolkit (VTK)

El Visualization Toolkit (VTK) és un sistema de software de codi lliure per a gràfics per computador en 3D, modelatge, processat d'imatges, *rendering* de volums i visualització de dades científiques. VTK, a més, inclou suport complementari per a aparells d'interacció en 3D, anotacions en dos i tres dimensions, i computació en paral·lel. El seu nucli està implementat com a una llibreria de C++, requerint als usuaris a construir aplicacions combinant diferents objectes dins una aplicació. El sistema, a més, suporta *wrapping* automatitzat per utilitzar el nucli de C++ dins de Python, Java i Tcl, de manera que les aplicacions de VTK poden estar escrites, també, utilitzant aquests llenguatges de programació interpretats.

VTK utilitza el Quality Software Process de Kitware (CMake, CTest, CDash i CPack) per construir, provar i empaquetar el sistema. Fent VTK una aplicació multi-plataforma dependent d'un desenvolupament conduit per testos i l'*eXtreme Programming* i habilitatnt l'aplicació produir codi

robust i d'alta qualitat. VTK s'utilitza arreu del món en aplicacions comercials, recerca i desenvolupament, i és la base de moltes aplicacions de visualització avançada com: ParaView, VisIt, VisTrails, Slicer, MayaVi i OsiriX.

2.3. Qt

Qt és el *framework* de C++ estàndard *de facto* per al desenvolupament de software d'alt rendiment. A més d'una llibre de C++ extensible, Qt inclou eines per escriure aplicacions ràpidament i fàcil. Qt és una llibreria multi-plataforma i, a més, el seu suport internacional li assegura arribar al més ample mercat possible.

El *framework* de C++ de Qt ha estat al nucli d'aplicacions comercials des de 1995. Qt és utilitzat per companyies i organitzacions tan diverses com Adobe®, Boeing®, Google®, IBM®, Motorola®, NASA, Skype®, i nombroses companyies i organitzacions més petites. Qt 4 està dissenyat per ser més fàcil d'utilitzar que les versions anteriors de Qt, afegint funcionalitats més potents. Les classes de Qt tenen característiques molt completes i proveeixen interfícies consistents per ajudar a l'aprenentatge, reduir la càrrega de treball del desenvolupador i incrementar la productivitat del programador. Qt és, i sempre ha estat, del tot orientada a objectes.

Qt disposa d'un ric conjunt de *widgets* ("controls" en la terminologia de Windows) que proveeix funcionalitats estàndards de GUI (Graphical User Interface – Interície Gràfica d'Usuari). Qt introdueix una alternativa innovadora per la comunicació entre objectes anomenada "*signals and slots*", que reemplaça la tècnica de *callback* antiga i poc segura:

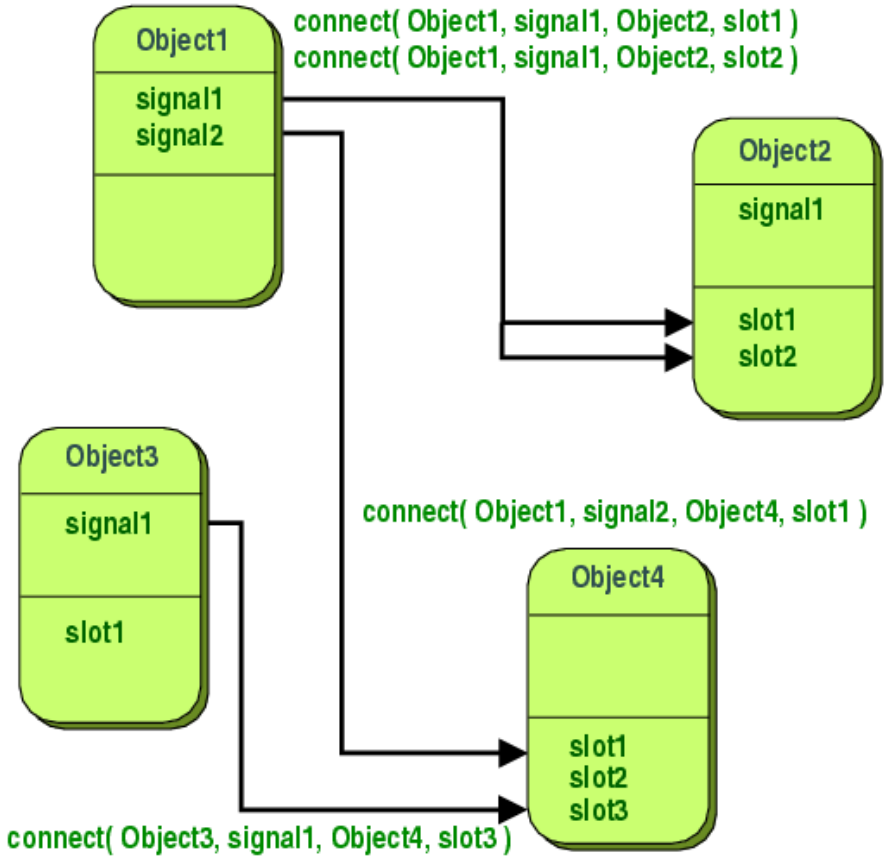


Figura 13 Model de connexions entre Signals i Slots

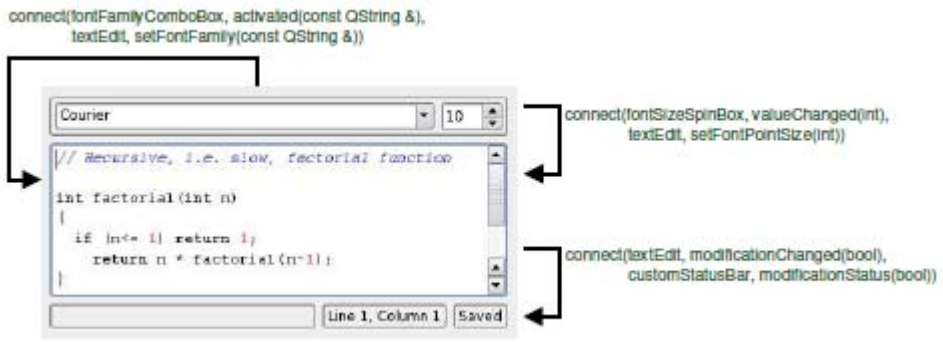


Figura 14 Exemple de connexions entre Signals i Slots

Qt distribueix el programa QtDesigner, una eina per dissenyar interfícies gràficament. QtDesigner suporta les potents característiques de distribució de Qt a més de posicionament absolut. Es pot utilitzar tant per dissenyar GUI's, com per crear aplicacions senceres gràcies al suport per integració amb populars entorns de programacio integrats (IDE's).

Qt disposa d'un suport excel·lent per a multimèdia i gràfics en 3D. Qt és el *framework de facto* per a programació en OpenGL® independent i multi-plataforma. Un sofisticat *framework* per *canvas* permet als desenvolupadors crear applicacions d'interacció gràfica que utilitzen les característiques de dibuix de Qt.

Qt fa possible crear aplicacions amb bases de dades independents de la plataforma utilitzant bases de dades estàndard. Qt inclou *drivers* per Oracle®, Microsoft® SQL Server, Sybase® Adaptive Server, IBM DB2®, PostgreSQL™, MySQL®, Borland® Interbase, SQLite i bases de dades que compleixen l'estàndard ODBC.

Qt és un *framework* de C++ madur que és utilitzat arreu del món. A més dels molts usos comercials de Qt, l'edició de Codi Lliure és la fundació de KDE, l'entorn d'escriptori de Linux.

Qt converteix el desenvolupament d'aplicacions en un plaer, amb el seu sistema multi-plataforma, el disseny visual de GUI's i la seva elegant API.

3. Starviewer Tool’s

1. Introducció

Per a poder diagnosticar amb la plataforma Starviewer, a part de visualitzar els estudis és necessari poder interactuar-hi. La interacció amb les imatges dels estudis comença amb la possibilitat de moure's per la representació en 3 dimensions (3D) de l'estudi, canviant les llesques (imatges) del volum a voluntat i per cada una de les vistes: axial, sagital i coronal. En aquest grup s'hi poden afegir eines com calibrar la lluminositat, el *zoom*, moviment de la imatge per la pantalla, rotació, *screenshot'*s, informació de llum / color en vòxels, etc... Totes aquestes funcionalitats ens permeten explorar el model de volum i ajuden als experts a identificar la lesió o la patologia. Una vegada s'ha identificat la lesió interessa poder realitzar càlculs sobre aquesta zona, ja sigui per calcular-ne el volum, per veure la dimensió d'una fractura, etc.

Tot aquest conjunt d'eines les anomenarem eines de mesura i en aquest conjunt i inclourem entre d'altres les distàncies, els angles, les regions d'interès (*ROI*'s), etc...

2. Objectius

La plataforma Starviewer ja porta integrades un conjunt d'eines de mesura. La principal limitació d'aquestes eines es que no es poden editar. L'ús actual de les eines es limita a la creació i posterior eliminació (explícita o implícita) de la mesura.

El nostre objectiu serà modificar el comportament actual de la interacció amb l'aplicació per permetre "editar" el resultat de les eines un cop creades: línies de distància, angles, àrees, etc...

Per assolir aquest objectiu ens cal estudiar la implementació actual de les eines i modificar-ne el comportament. Volem remarcar que els anàlisis, dissenys i implementacions són per les eines de creació[3], la resta no han de sofrir cap canvi important.

[3] Eines de creació: eines que "creen" un dibuix a la pantalla i, per tant, necessiten la intervenció de l'usuari.

3. Anàlisi de Requeriments

La interacció de l'usuari amb l'aplicació per a l'ús d'eines de diagnòstic es resumeix en:

- Selecció de l'eina desitjada.
- Ús continu de l'eina (l'eina no es "desselecciona").
- Selecció d'una altra eina, que talla l'ús de l'eina anterior.

Per a la implementació de l'edició es resol canviar la interacció de l'usuari, una qüestió delicada, ja que és un canvi important en l'experiència d'usuari.

La nova interacció de l'usuari es resumiria en:

- Selecció de l'eina desitjada (sempre parlant d'eines de creació/mesura).
- Ús únic de l'eina (es "desselecciona" després de l'ús).
- Resta seleccionada l'eina en "mode edició" anterior. Fa possible l'edició de totes les eines creades fins el moment.

D'això es desprèn la necessitat de tenir en l'aplicació dos modes: creació i edició.

El mode **creació** és l'estat de l'aplicació quan s'utilitza una eina de creació i el mode **edició** és l'estat implícit de l'aplicació quan no s'utilitza una eina de creació.

3.1. Diagrames d'estats

Els següents diagrames d'estats il·lustren els estats de les eines durant el seu ús. El primer és l'actual i el que ja està en funcionament. El segon és el proposat per integrar-hi la part d'edició.

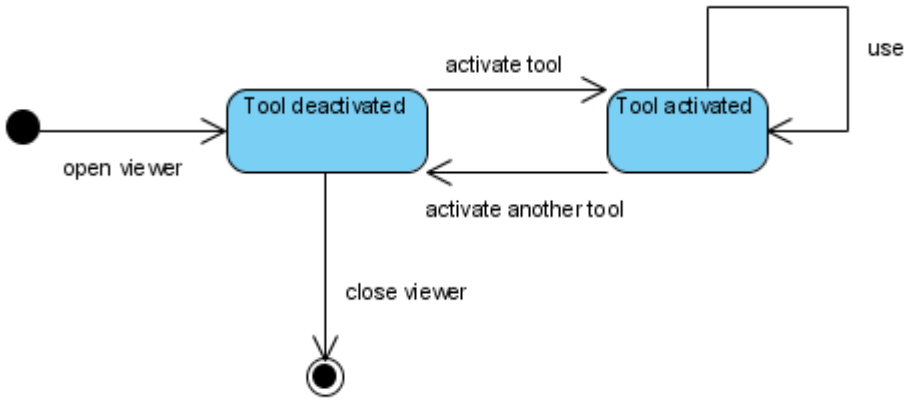


Figura 15 Diagrama d'Estats original de l'aplicació

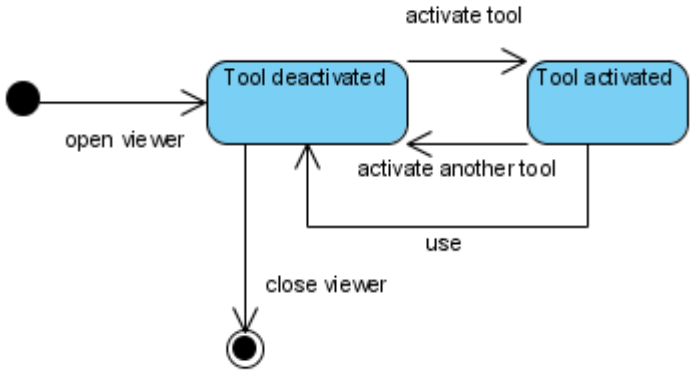


Figura 16 Diagrama d'Estats proposat per a integrar-hi l'edició

En els anteriors diagrames d'estats es pot apreciar que la diferència entre ells és mínima. De fet, a aquest nivell, l'única diferència és el fet que una eina passa d'activada a desactivada després del seu ús en el nou diagrama proposat.

És un petit canvi en un diagrama d'estats, però un canvi important en l'experiència d'usuari en utilitzar l'aplicació.

4. Disseny

Donat que l'aplicació Starviewer està funcionant en real abans de fer qualsevol modificació ens cal tenir clar com és el disseny actual de l'aplicació. En aquest apartat presentarem un descripció de l'estat original de l'aplicació i a continuació proposarem un nou disseny per integrar l'edició.

4.1. Descripció de l'estat original de l'aplicació

Per començar presentarem el diagrama de classes , a continuació veurem una descripció detallada de les classes i finalment els diagrames de seqüència.

4.1.1. Diagrama de classes

El mòdul actual de Tool's o eines de l'aplicació té el següent esquema:

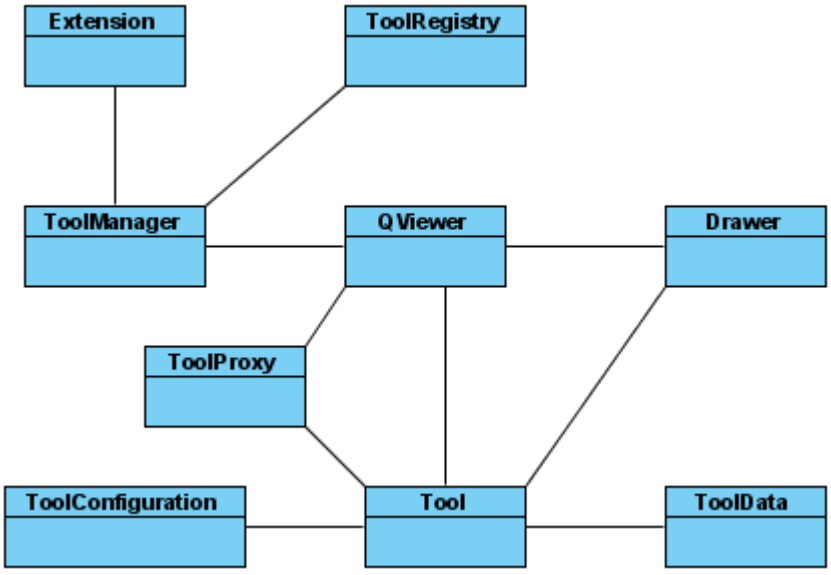


Figura 17 Diagrama de classes del mòdul Tool's de la plataforma Starviewer.

4.1.2. Descripció de les classes

A continuació es descriuen les classes que formen part d'aquest diagrama:

Nom de la Classe	Descripció
`Extension`	les “extension’s” són les diferents modalitats de visualització de la plataforma.
`ToolManager`	gestiona les Tool’s. Defineix per a cada visor quines Tool’s li seran assignades i amb quina configuració, gestiona els grups de Tool’s, les accions que les disparen, etc...
`ToolRegistry`	és l’encarregat de crear les Tool’s i d’obtenir les “action’s” associades a aquestes.
`QViewer`	mostra per pantalla la informació gràfica. Conté un Drawer i un ToolProxy.
Drawer	s’encarrega de dibuixar les primitives (línies, polilínies, polígons, text, etc) a la pantalla. S’encarrega de gestionar si aquestes primitives es veuen o no en funció de la imatge actual del visor.
`ToolProxy`	rep els events del visor. És qui coneix les Tool’s actives per cada visor. Reenvia els events rebuts a les Tool’s actives.
`ToolConfiguration`	guarda de forma genèrica els atributs que pot configurar una Tool.
`ToolData`	defineix les dades d’una Tool.

4.1.3. Diagrames de Seqüència

A continuació es mostra el diagrama de seqüència de l'aplicació en inicialitzar una extensió, en aquest cas l'extensió de visualització en 2D, la més utilitzada.

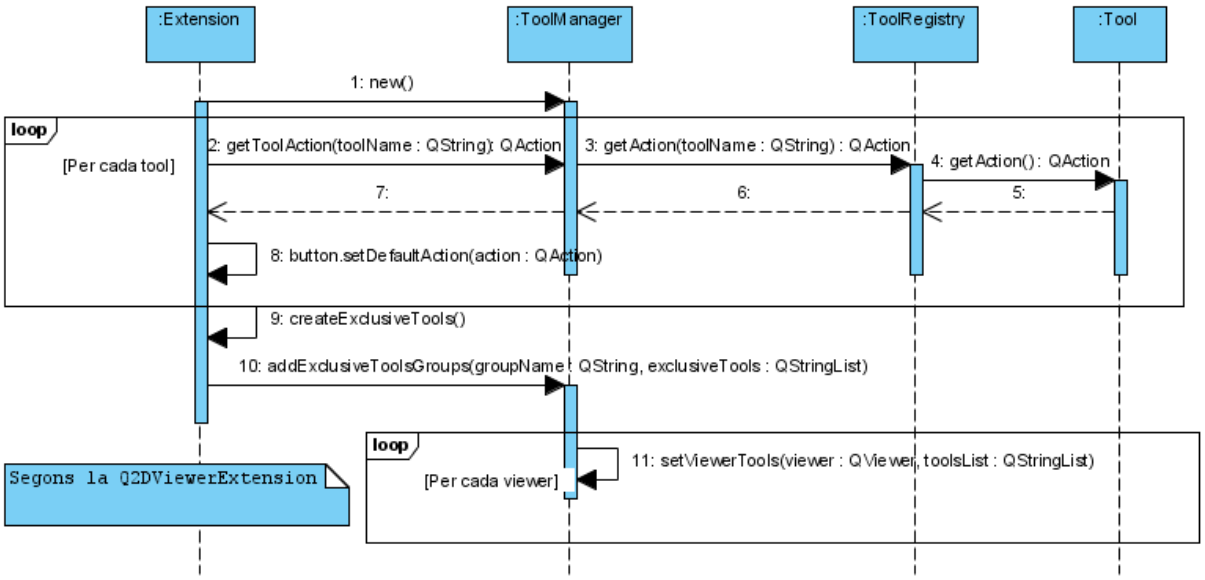


Figura 18 Diagrama de seqüència de l'inicialització de l'extensió de visualització en 2D

En aquest diagrama es mostra com, des de la pròpia extensió i utilitzant el `ToolManager`, es van creant les accions que "disparen" l'execució de les Tool's. Finalment, a cada visor de l'extensió se li passa una llista amb les Tools, que suporta, per a la seva posterior creació.

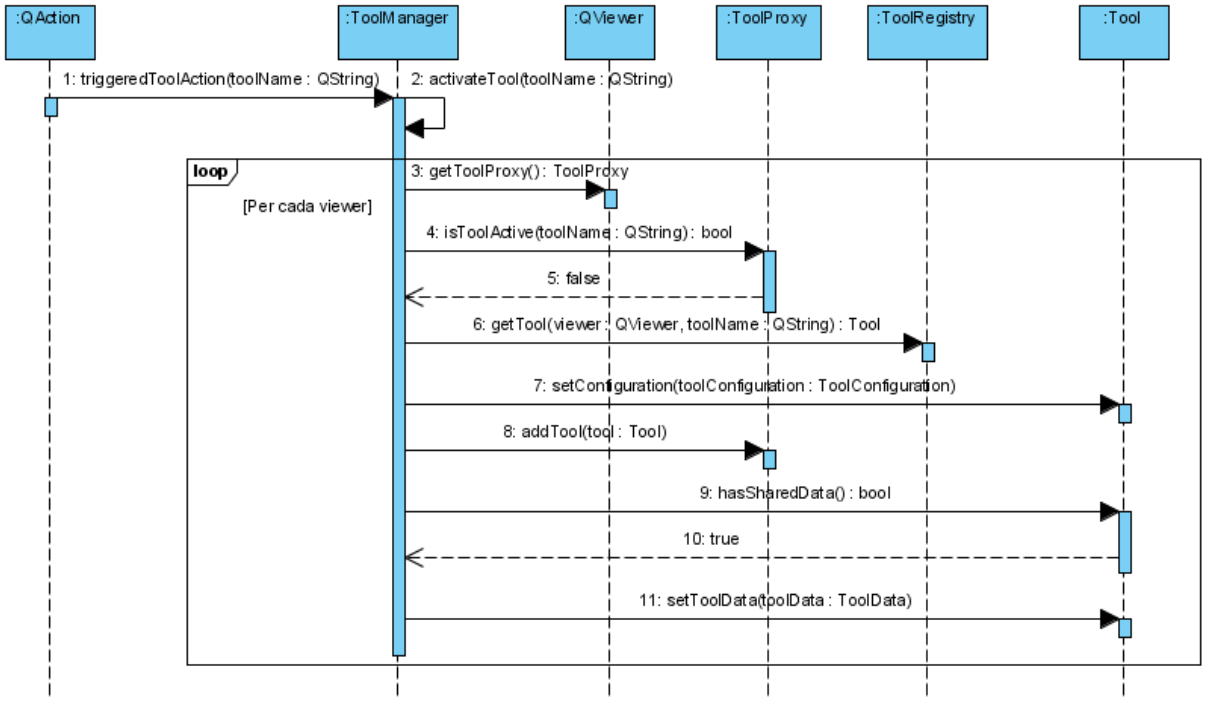
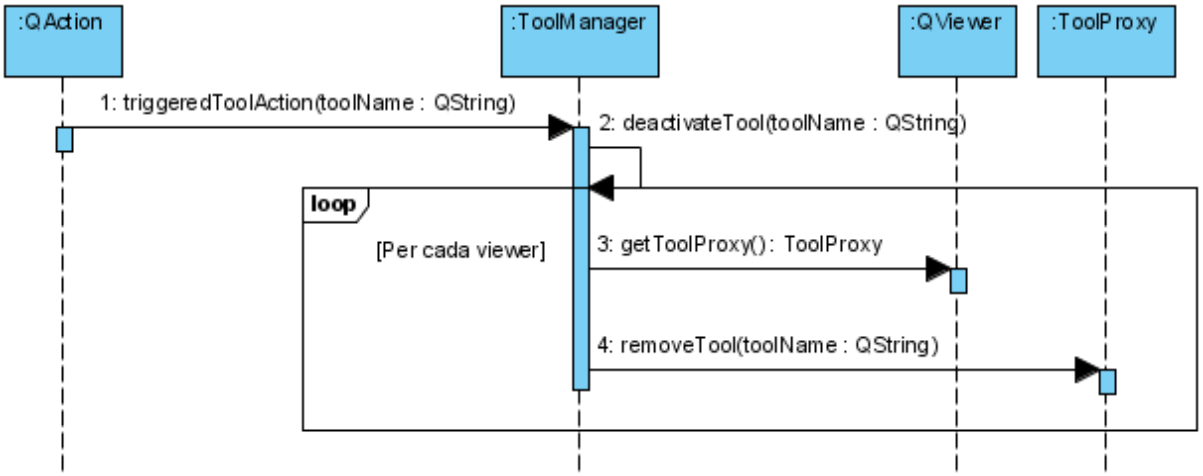


Figura 19 Activació d'una Tool

En aquest diagrama es mostra com s'activa una `Tool`. Un cop llançada l'acció (clicar el botó) per activar la `Tool`, el `ToolManager` comprova que no s'està utilitzant, per tant l'activa. Per cada visor de l'extensió comprova si la `Tool` ja és al `ToolProxy` (ha estat activada abans). En cas que no hi sigui, recupera la `Tool` del `ToolRegistry` (registre de `Tool`'s, encarregat de crear-les), hi afegeix la seva configuració (en cas que en tingui), i afegeix la `Tool` al `ToolProxy`. En cas que la `Tool` tingués informació compartida entre visors se li afegiria aquesta informació.

Figura 20 Desactivació d'una Tool



Quan es vol utilitzar una `Tool` diferent de l'actual, aquesta s'ha de desactivar. Funciona de la mateixa manera que l'activació: el `ToolManager` rep una acció amb el nom de la `Tool` actual. Comprova si està activada, ho està. Per cada visor de l'extensió elimina la `Tool` del `ToolProxy`.

4.2. Disseny del nou mòdul de Tool's

Ara és el moment d'estudiar les possibilitats del disseny actual envers les noves funcionalitats que se li demanen.

El disseny actual ja és "funcional", per tant, s'ha d'intentar canviar-ne el comportament sense que això afecti massa l'estructura actual. Així també s'evita implementar de nou certes funcionalitats que ja estan implementades. El nostre objectiu serà estendre la funcionalitat intentant introduir el mínim de canvis a l'esquema actual i garantir màxima usabilitat.

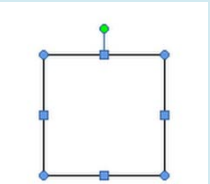
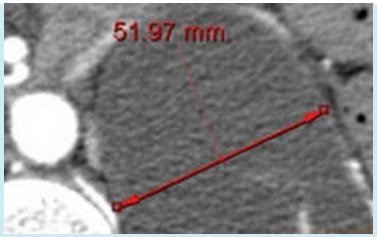
4.2.1. Discussions sobre usabilitat

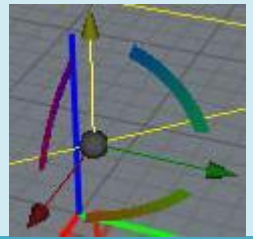
La usabilitat i l'experiència d'usuari són dos aspectes determinants i molt delicats que s'han de tenir en compte a l'hora de dissenyar i implementar una interfície d'usuari. S'està canviant el funcionament de les Tool's de creació, una de les eines bàsiques de l'aplicació i, potser, la més utilitzada, per tant aquests aspectes esmentats s'han de tenir molt en compte.

Per decidir com jugaria el seu paper el factor "edició de Tool's" s'ha fet un estudi de diferents paquets de software que donen solució al mateix problema. D'aquest estudi es pretén extreure certs aspectes de la interacció de l'usuari a l'hora de crear figures a la pantalla i la seva posterior modificació.

4.2.1.1. Detalls de l'estudi

En la taula següent mostrem alguns dels programes que s'han analitzat indicant els aspectes a tenir en compte i algunes imatges que ens ho il·lustren.

Nom del Software	Descripció dels aspectes estudiats	Imatges
Microsoft Office 2007: Word i PowerPoint	Per crear una figura se selecciona l'eina. Es crea amb un sol clic (forma predeterminada) o arrossegant el mouse (modifica llargada, amplada, etc). Un cop creada es torna a l'eina original (selecció). En acostar-se a la figura apareixen els "handlers" depenent de la posició. S'utilitzen per moure la figura (perímetre), girar-la (handler fora de la figura), redimensionar-la (handlers al mig de les arestes que formen el perímetre, i als vèrtexs).	Handlers d'una figura de Microsoft Office 2007. 
Inkscape	Inkscape és un editor de gràfics vectorials.	 Eina de selecció
Philips Workstation	Workstation per al diagnòstic per la imatge. Hi ha una eina per defecte: Select / Scroll. Després de crear una figura (p.ex. una distància) es torna a activar l'eina per defecte. Aquesta mateixa interactua amb la figura dibuixada, fent aparèixer una sèrie de handlers. Aquests permeten moure la figura (segments, perímetre) o modificar-ne els vèrtexs.	
E-Film		
Alma	Funcionalitats molt semblants a la Philips Workstation.	 Representació d'una distància. Es poden apreciars els handlers a ambdós extrems.
Osirix	Funcionalitats molt semblants a la Philips Workstation.	

Paraview	En el Paraview, l'edició funciona de la mateixa manera que en els paquets d'"office": un cop creada la figura es passa a l'eina per defecte i en acostar-se a una zona sensible de ser editada es canvia el cursor i en aquest cas s'ilumina la zona.	 Selecció d'un vèrtex de l'objecte: iluminat en verd.
Gimp	Editor gràfic. No "crea" figures, sinó que les "dibuixa". Per modificar el que s'acaba de dibuixar s'ha de seleccionar i després utilitzar l'eina corrsponent: moure, girar, etc.	 Eines de moviment i rotació
KOffice 2.0: Krita i Kivio	Un altre paquet d'"office" com Microsoft Office. La interactivitat és semblant, sinó idèntica.	
OpenOffice : Draw i Write	Un altre paquet d'"office". Funcionalitats molt semblants a Microsoft Office i a KOffice.	
Photoshop	Funcionalitat molt semblant al Gimp.	
Google SketchUp	Modelatge en 3D. Després de crear una figura (línia, quadrat, cub,etc), es manté l'eina de dibuix. Per editar una figura s'ha de seleccionar amb l'eina de selecció, i posteriorment editar amb les eines pertinents: moviment, rotació. L'edició es duu a terme a partir del punt que tria l'usuari dinàmicament: vèrtex, centre (d'una cara, d'un volum), punt mig d'una aresta, etc...	 Eines de moviment i rotació
Kerkythea	Kerkythea és un sistema de rendering, però també inclou edició de figures. En aquest cas la solució està molt dirigida al modelatge en 3D.	 Eina de moviment: inclou moviment lliure, moviment al llarg dels eixos i rotació sobre els eixos.

4.2.1.2. Conclusions de l'estudi

Un cop fet l'estudi, es va decidir implementar l'edició com una mescla entre els aspectes observats als paquets d'"office" i a les Workstations. És a dir, un cop dibuixat a pantalla s'activarà l'eina per defecte, la que permetrà editar les figures dibuixades fins el moment. Per editar una figura, s'hi haurà de passar per sobre amb el punter del mouse. Llavors apareixeran els handlers pertinents, que permetran editar-la.

4.2.1.3. Noves discussions arran de l'estudi

El resultat que es vol obtenir ja és més o menys clar. Encara, però, queden per discutir certs aspectes relacionats directament amb l'aplicació i el seu funcionament.

Que s'hagi de tornar a una `Tool` per defecte després de dibuixar una figura planteja diferents camins:

1. Tenir dos modes d'aplicació, o de Tools: creació i edició.
2. Tenir una `Tool` "edició" amagada, o sempre activa.

(1) Tenir una `Tool` "edició" suposa una sèrie de problemes:

- En canviar d'imatge amb la `Tool` pertinent, en fer *zoom*, en canviar la lluminositat, què passa si el cursor es mou per sobre una figura? S'activa l'edició i es deixa de fer el que s'estava fent involuntàriament. Per evitar-ho es podria mantenir un indicador de `Tool` en ús, però tampoc té gaire sentit, ja que les Tool's o bé estan activades, o bé desactivades.
- En començar la creació d'una nova figura, si es volgués posar un punt sobre una altra representació, la `Tool` "edició" no ho deixaria fer, ja que començaria l'edició del handler corresponent (extrem, segment, etc...).

S'opta per utilitzar els dos modes a l'aplicació: creació i edició.

El mode edició és el que permet modificar els handlers:

- Per moure la figura s'utilitzarà qualsevol part de la figura que no sigui un handler:
  - Línies (perímetre): distàncies, angles, ROI's, text, fletxes, etc
  - Zona tancada: pròtesis, etc
- Handlers:
  - Distància: punt inicial i punt final
  - Angle: punt inicial, punt mig i punt final
  - ROI: un per vèrtex
  - Text: tota la *bounding box*[4]
  - ...

Les figures amb anotacions pròpies: distàncies, angles, etc, han de permetre desplaçar el text. Caldria dibuixar una nexe d'unió entre figura i text: p. Ex: línia puntejada.

[4] Bounding box: caixa englobant: rectangle que dibuixa un perímetre al voltant del text.

Quan analitzem la interacció entre usuari i les mesures cal tenir en compte diferents qüestions. A continuació ens presentem algunes:

1. Fent un clic de *mouse* se selecciona una representació. Com es distingeix quina representació és la seleccionada?

- Dibuixar una capsa al voltant (*bounding box*)
- Canviar el color: problema de contrast en pantalles monocromes. Resultaria difícil de distingir.

2. Opcions amb més d'una representació seleccionada (Shift+clic per seleccionar més d'una representació):

- Sembla que només té sentit esborrar.
- Un desplaçament simultani no té massa sentit.

3. Esborrar representacions:

- Seleccionar i prémer tecla Supr.
- Amb l'eina "goma d'esborrar".

4. Definir connexions entre representacions. Que ho pugui fer l'usuari no té sentit. Però potser caldria poder definir-ne internament de cares a desplaçaments de representacions compostes (figura + text). Es podria mantenir una relació entre la figura principal, el text i una figura que fes d'unió.

- S'han de poder moure les representacions connectades simultàniament.
- S'han d'esborrar simultàniament.
- Es podrien definir jerarquies: en moure una distància, s'hauria de moure el text simultàniament, però en moure el text la distància no s'hauria de moure.

Totes aquestes qüestions tenen el seu reflex en el codi pertinent, sempre que es puguin dur a terme. En el cas de les disjuntives que apareixen s'haurà d'optar per avançar en una direcció en detriment de l'altra.

4.2.2. Diagrama de classes

Ara, un cop decidit com haurà d'interactuar l'usuari amb l'aplicació (concretament amb l'ús de les eines), cal modificar el diagrama de classes actual per a incorporar-hi el nou sistema per a l'edició. A l'hora de modificar-lo cal tenir en compte el funcionament actual, per tant, l'objectiu principal és estendre la funcionalitat, sense haver de redissenyar les parts que ja ara funcionen.

Fins ara, un cop s'havia "dibuixat" amb l'eina pertinent, aquest dibuix s'enviava a la classe Drawer i aquest s'encarregava de pintar-la a la pantalla, d'amagar-la i mostrar-la quan es canviava la imatge del visor, d'eliminar-la del conjunt de "dibuixos" en esborrar-la de la pantalla (per part de l'usuari), i de tot el que fes referència a aquell dibuix.

D'això s'extreu que un cop l'eina ha estat utilitzada, d'aquesta només en queda el dibuix, que és l'element persistent. Aquest element és una primitiva, una DrawerPrimitive en l'aplicació, i es disposa de les següents:

`DrawerCrossHair`	Dibuixa una "CrossHair", una creu per indicar un punt
`DrawerLine`	Dibuixa una línia simple
`DrawerPoint`	Dibuixa una sol punt
`DrawerPolygon`	Dibuixa una polígon amb nombre de vèrtexs definit per l'usuari
`DrawerPolyline`	Dibuixa una polilínia amb nombre de vèrtexs definit per l'usuari
`DrawerText`	Permet escriure text

L'objectiu, doncs, serà obtenir el control sobre aquestes primitives, més enllà d'amagar-les, mostrar-les i eliminar-les. Aquesta responsabilitat, però, s'escapa de les que ha de tenir el `Drawer`. Es necessita una nova classe que faci de contenidor de primitives i que, alhora, permeti modificar-les (editar-les).

Hi ha altres llibreries que ja han solucionat aquest problema: aquest és el cas de Qt. Per a implementar la nova part d'edició s'ha decidit utilitzar una solució semblant a la d'aquestes llibreries.

4.2.2.1. Descripció de la solució de les llibreries Qt

L'arquitectura *Graphics View*

Graphics View proveeix una aproximació basada en ítems a la programació *model-view*. Diferents *views* (vistes) poden observar una mateixa escena, i l'escena conté ítems de diverses formes geomètriques.

L'escena (The Scene).

`QGraphicsScene` proveeix l'escena de *Graphics View*. L'escena té les següents responsabilitats:

- Donar una interfície rápida per gestionar un gran nombre d'ítems.
- Propagar events a cada ítem.
- Gestionar l'estat de cada ítem, com la selección i l'edició.
- Proveir d'una funcionalitat de *rendering* sense transformació, principalment per impressions.

L'escena serveix de contenidor per objectes de `QGraphicsItem`. Els ítems s'afegeixen a l'escena invocant `QGraphicsScene::addItem()`, i més tard recuperats invocant alguna de les funcions de retorn d'ítems.

L'arquitectura de propagació d'events a `QGraphicsScene` programa els events de l'escena per a ser enviats als ítems i, a més, gestiona la propagació d'events entre ítems. Si l'escena rep un event de click de mouse a certa posició, l'escena propaga l'event a l'ítem que hi hagi en aquella posició.

`QGraphicsScene`, a més, gestiona certs estats d'ítems, com la selecció i el focus. Es poden seleccionar ítems a l'escena invocant `QGraphicsScene::setSelectionArea()`, passant una forma arbitrària. Aquesta funcionalitat també s'utilitza com a base de la selecció per la goma d'esborrar a `QGraphicsView`. Per recuperar la llista de tots els elements seleccionats, s'invoca `QGraphicsScene::selectedItems()`. Un altre estat gestionat per `QGraphicsScene` és saber si un ítem te una entrada per teclat. Es pot assignar el focus a un ítem invocant `QGraphicsScene::setFocusItem()` o `QGraphicsScene::setFocus()`, o recuperant el focus actual amb `QGraphicsScene::focusItem()`.

La vista (The View).

`QGraphicsView` proveeix el visor, des d'on es visualitzen els continguts de l'escena. Es poden concentrar diferents vistes a una mateixa escena, per proveir diferents punts de vista de les mateixes dades.

La vista rep events d'entrada des del teclat i el mouse, i els tradueix a events d'escena (convertint les coordenades utilitzades a coordenades d'escena allà on calgui), abans d'enviar els events a l'escena visualitzada.

Utilitzant la seva matriu de transformació `QGraphicsView::matrix()`, la vista pot *transformar* el sistema de coordenades de l'escena. Això permet efectes de navegació avançats com el *zoom* i la rotació. Per conveni, `QGraphicsView`, també proveeix funcions per *traduir* entre coordenades de vista i d'escena: `QGraphicsView::mapToScene()` i `QGraphicsView::mapFromScene()`.

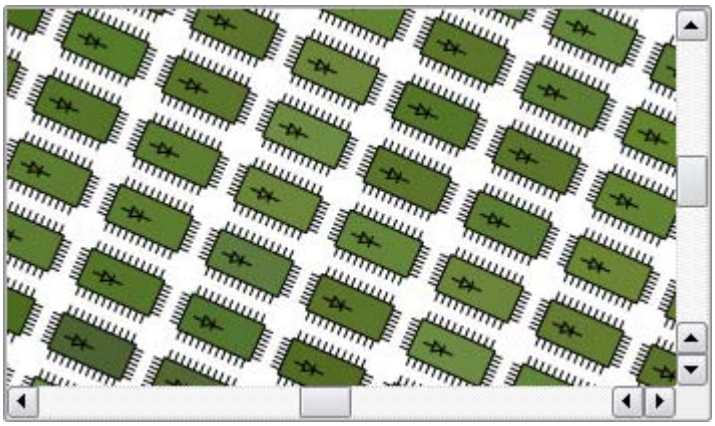


Figura 21 Vista d'una escena amb ítems repetits.

L'ítem (The item).

`QGraphicsItem` és la classe base per a ítems gràfics en una escena. *Graphics View* proveeix una sèrie d'ítems estàndard per formes típiques, com rectangles(`QGrapicsRectItem`), el·lipsis (`QGraphicsEllipseItem`) i text (`QGraphicsTextItem`), però les propietats més interessants de `QGraphicsItem` aparèixen quan s'escriu un ítem propi. Entre d'altre, `QGraphicsItem` suporta les següents propietats.

- Events de mouse: clickar, moure, deixar anar, doble click, a més d'events de la roda, posició i menús contextuals (press, move, release, double click, wheel, hover and context menu).
- Focus d'entrada per teclat i events de teclat.
- Arrossegar i alliberar (Drag and drop).
- Agrupar, a través de relacions pare-fill, i amb QGraphicsItemGroup.
- Detecció de col·lisions.

Els ítems *viuen* en un sistema de coordenades local, i com `QGraphicsView`, també proveeixen una sèrie de funcions per transformar coordenades entre l'ítem i l'escena, i des d'ítem a ítem. A més, com `QGraphicsView`, es pot transformar el seu sistema de coordenades utilitzant una matriu: `QGraphicsItem:matrix()`. Això és útil per escalar i rotar ítems individuals.

Els ítems poden contenir altres ítems (fills). Transformacions als ítems pare són heretades per tots els seus fills. Tot i les transformacions acumulades d'un ítem, totes les seves funcions (p. ex. `QGraphicsItem::contains()`, `QGraphicsItem::boundingRect()`, `QGraphicsItem::collidesWith()`) continuen operant amb coordenades locals.

`QgraphicsItem` suporta detecció de col·lisions a través de la funció `QGraphicsItem::shape()` i `QGraphicsItem::collidesWith()`, que són funcions virtuals.

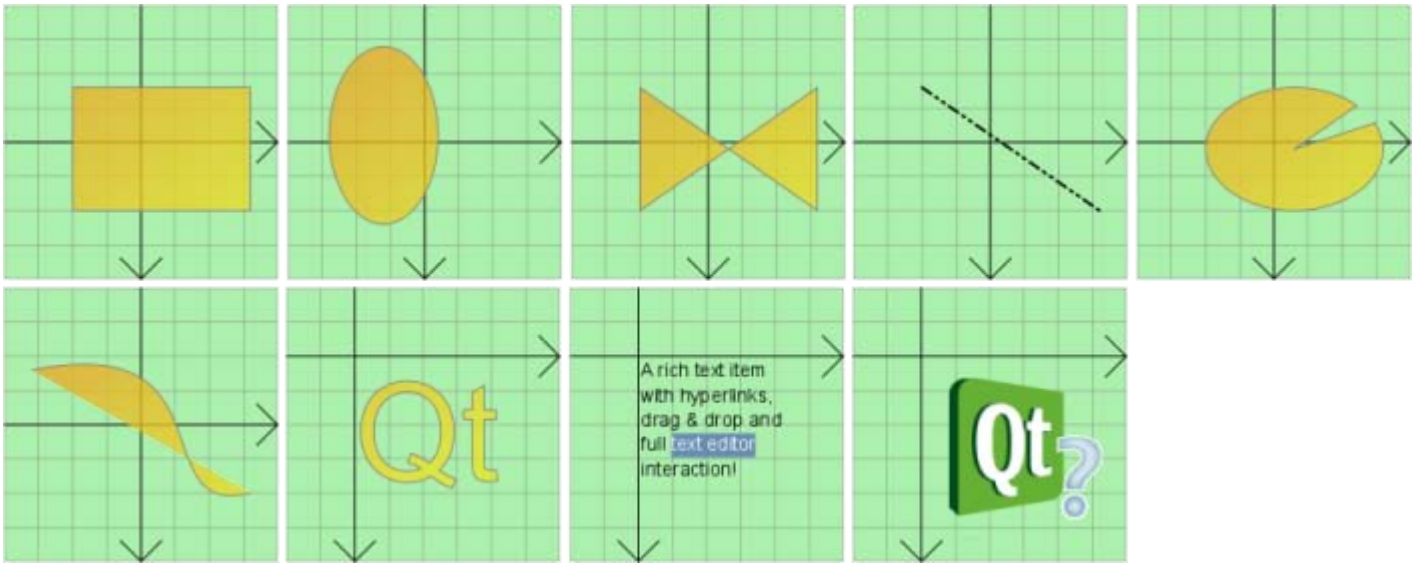


Figura 22 Diferents ítems de QGraphicsItem.

4.2.2.2. Conclusions de la solució de les llibreries Qt

Es considera aquest model una molt bona solució pel problema que s'ha de resoldre. Tot i això, no totes les característiques d'aquesta implementació són necessàries. El que cal fer ara és adaptar aquesta solució al model actual del mòdul de Tool's de l'aplicació actual.

Per començar cal identificar els elements que componen aquesta implementació. Un diagrama de classes amb la solució per part de Qt seria la següent:

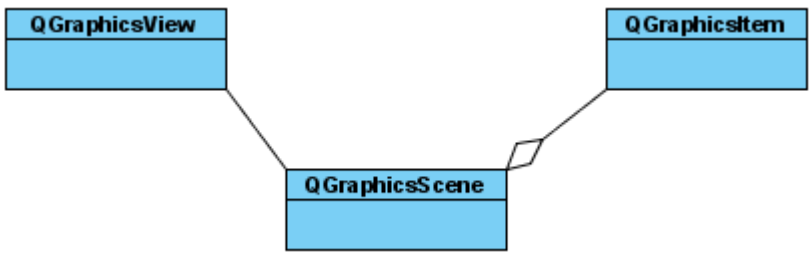


Figura 23 Diagrama de classes del model Graphics View de Qt

En l'aplicació que ens ocupa hi ha dos elements que ja existeixen:

- `QGraphicsItem`: cadascuna de les `DrawerPrimitive`.
- `QGraphicsView`: cadascun dels `QViewer`.

La identificació, però, no és directa. Hi ha certes característiques dels elements de l'aplicació que els allunyen de la solució que s'adoptarà:

- L'element *escena* no existeix. S'haurà de crear.
- Cada `QViewer` té el seu propi `Drawer` i, per tant, les seves pròpies `DrawerPrimitives`. Això indica que caldrà tenir una `Scene` per cada vista.
- Les `DrawerPrimitive` s'emmagatzemen al `Drawer`. Això xoca directament amb les responsabilitats de l'element `Scene`.

4.2.2.3. Canvis de cares a la nova implementació

S'haurà de crear algun element equivalent a l'*escena*. Aquest haurà de ser un contenidor de `DrawerPrimitives`, tal com ara fa el `Drawer`. A més, però, haurà de gestionar els events d'edició de les primitives, i per tant fer recalcular els resultats, ja que la finalitat de les Tool's no és el dibuix en si, sinó el càlcul associat.

Això suposa un altre problema: com sabrà la primitiva quina és l'eina que l'ha creat? Necessitem saber-ho, ja que un cop modificada la primitiva s'haurà de refer el càlcul. Aquest és un problema que amb l'actual disseny és impossible de solucionar, per tant, s'haurà de modificar. Es plantegen diferents solucions:

- Guardar una referència (un nom) de l'eina que l'ha creat. Això planteja un altre problema: de vegades una eina crea més d'una primitiva, com les mantenim unides?
    - Guardar un número amb la referència o un codi de manera que les primitives que tenen el mateix codi corresponguin a la mateixa eina.
- Mantenir les primitives en la pròpia eina i fer l'eina persistent.
- Crear un nou element que contingui les primitives creades i el càlcul pertinent en un sol conjunt.

Per començar, descartar la segona opció per inviable. Fer persistent l'eina xoca directament amb el disseny actual del mòdul i destrossa totalment la filosofia de la classe `Tool`. A més, crearia una gran diferència entre les eines que no tenen cap representació a la pantalla i que, per tant, no han de ser persistents i les que sí.

La primera opció és viable. Però és una solució pèssima de cares al paradigma de l'Orientació a Objectes i les premisses de la coherència i l'acoblament. Que una primitiva tingui coneixement de l'eina que l'ha creat no té sentit. I tampoc té sentit que tingui coneixement de la resta de primitives que com ella mateixa, han estat creades per la mateixa eina. Aquesta solució, però, crea un nou problema: una eina haurà de tenir dos camins d'entrada: el de creació, des d'on es dibuixa la primitiva i es fa el càlcul corresponent, i el d'edició, des d'on la primitiva ja està dibuixada i només cal fer el càlcul.

Així, doncs, queda la tercera opció. Aquesta també planteja un disseny nou de la classe `Tool`: extreure'n el dibuix i el càlcul. Així, doncs, el nou element serà una `Tool`, però amb un altre nom? La classe `Tool` deixarà d'existir tal i com és ara, és a dir, canviant-ne la filosofia? Aquesta sembla ser la solució més viable i correcta, però encara deixa certs dubtes a resoldre.

Actualment una `Tool` funciona de la següent manera:

1. Es crea la primitiva (l'usuari interactua amb la pantalla per crear-la)
2. S'envia al `Drawer`.
3. Es fa el càlcul pertinent.
4. Es crea el text amb el resultat del càlcul.
5. S'envia el text al `Drawer`.
6. S'inicialitza de nou per tornar a començar.

El que sembla més viable per a solucionar aquest problema és crear un element per sobre de les primitives que també sigui persistent i unifiqui l'edició i el càlcul. L'element encarregat de dibuixar la primitiva també s'hauria d'extreure d'aquest model. D'aquesta manera, per una banda es crea el dibuix: la primitiva, i per l'altra es fa el càlcul corresponent sense preocupar-se de com ha estat dibuixada la primitiva. Això, a més, obre un camí cap a l'edició: es pot extreure que l'edició haurà de modificar la primitiva emmagatzemada en aquest element i, llavors, repetir el càlcul sobre la mateixa.

Així doncs, l'equivalent a l'*escena* farà de contenidor d'aquest nou element i no de les primitives directament. Ara cal batejar aquests elements, per tant l'equivalència queda de la següent manera:

- `QGraphicsItem`: cadascuna de les ~~DrawerPrimitive~~ `ToolRepresentation`.
- `QGraphicsView`: cadascun dels `QViewer`.
- `QGraphicsScene`: element `RepresentationsLayer`.

Una `ToolRepresentation` és una representació persistent d'una `Tool`, és a dir, el dibuix d'aquesta (`DrawerPrimitives`) i el càlcul associat.

La `RepresentationsLayer` (capa de representacions) és el magatzem de `ToolRepresentation` i el seu gestor.

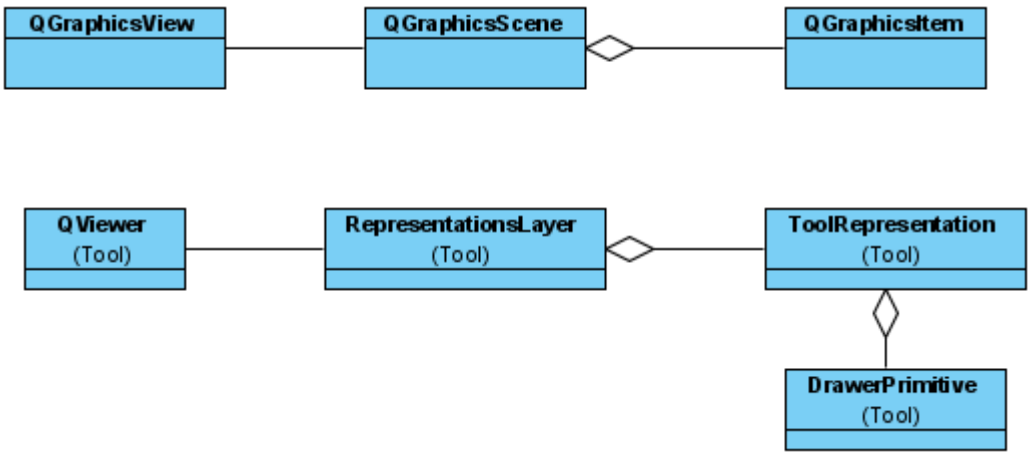


Figura 24 Comparació dels elements de les llibreries Qt amb els de l'aplicació

Queda pendent extreure el dibuix de la classe `Tool` pròpiament dita. Això es farà amb un `Outliner`, que serà qui rebrà els events de la `Tool` i els utilitzarà per dibuixar una `DrawerPrimitive`. D'aquesta manera també s'eviten repeticions de codi entre diferents `Tool`'s que utilitzin les mateixes primitives, ja que aquestes es dibuixaran de la mateixa manera.

I, finalment, es pot plantejar un diagrama de classes que satisfarà totes les demandes de les modificacions al mòdul de `Tool`'s de l'aplicació:

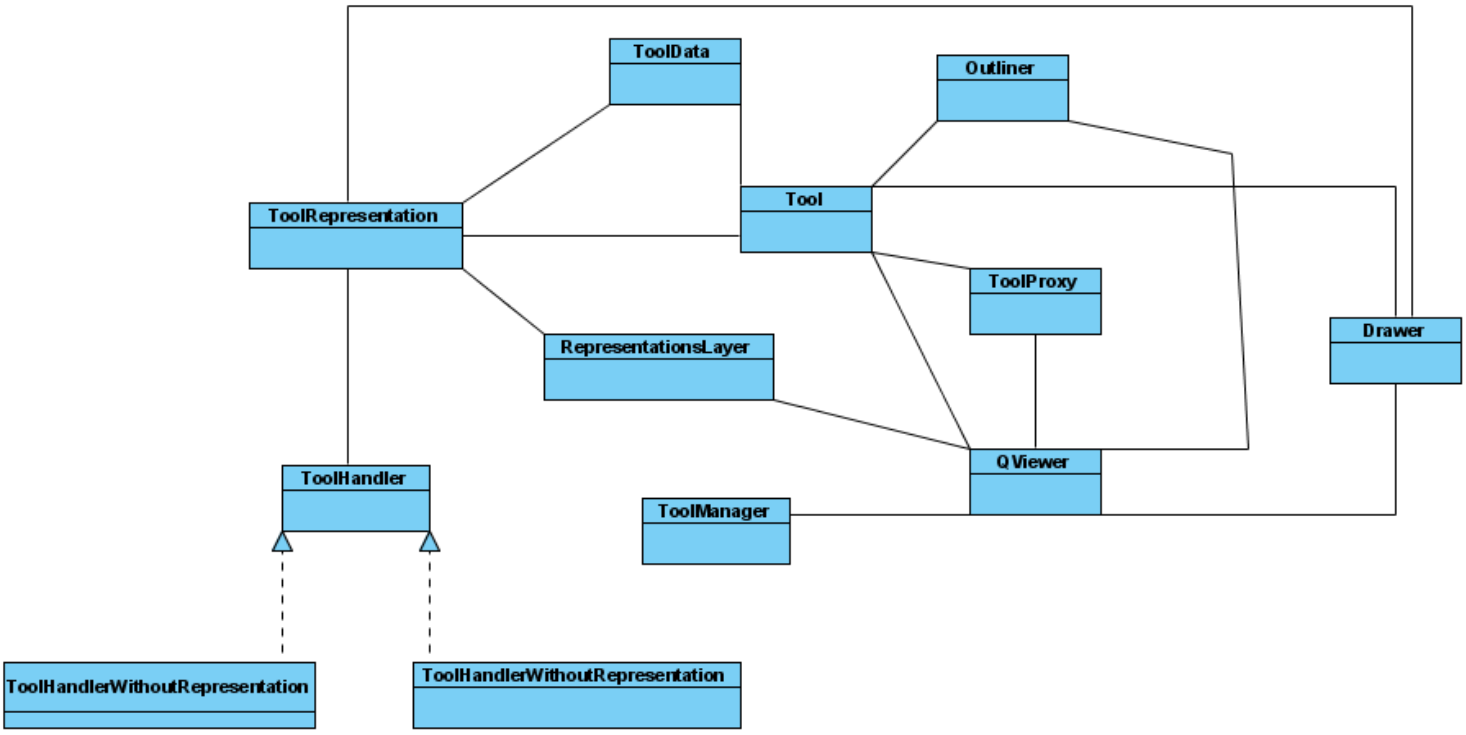


Figura 25 Diagrama de classes final

5. Implementació

Partint de l'aplicació i els dissenys mostrats en l'apartat anterior, s'inicia la implementació de les classes que compondran el nou model.

El primer pas en la implementació de la modificació d'aquest mòdul és transformar la implementació de les Tool's actuals al nou disseny que, més endavant, en permetrà l'edició.

5.1. Implementació de la modificació de les Tool's actuals

En un apartat anterior[5] s'ha explicat com funcionava una `Tool` amb la implementació original. Aquest comportament ha de canviar, així com els elements involucrats. En l'últim diagrama mostrat[6] i en comparació amb el diagrama original[7] d'aquest mòdul es pot apreciar que s'han d'afegir 3 elements: `RepresentationsLayer`, `ToolRepresentation` i `Outliner`. També hi ha el conjunt de `ToolHandler`, però això ja pertany a l'edició pròpiament dita i es tractarà en un altre capítol[8].

El primer que s'ha de fer és establir els passos a seguir per l'aplicació quan un usuari vol "dibuixar" una `Tool`. Això s'il·lustrarà amb una fitxa de cas d'ús i posteriorment el diagrama de seqüència, que s'utilitzarà de referent per a la implementació final.

5.1.1. Fitxa de cas d'ús

Nom	**Creació d'una Tool**
Descripció	L'usuari prem el botó de la `Tool` a crear i la dibuixa a la pantalla
Actor	Usuari
Precondició	Hi ha un estudi obert al visor 2D
Flux Principal	1. L'usuari prem el botó de la `Tool` 2. L'usuari "dibuixa" la figura de la `Tool` a la pantalla 3. La `Tool` fa els càlculs necessaris i els pinta a pantalla
Flux Alternatiu	---
Postcondició	A la pantalla s'hi afegeix una figura dibuixada i els càlculs corresponents.
Comentaris	---

5.1.2. Diagrama de seqüència

Aquest és el diagrama de seqüència que deriva de l'anterior fitxa de cas d'ús, i el diagrama de classes original.

[5] Apartat 4.2.2.3, pàg. 41.
[6] Figura 25, pàg. 42.
[7] Figura 17, pàg. 26.
[8] Apartat 5.2, pàg. 46.

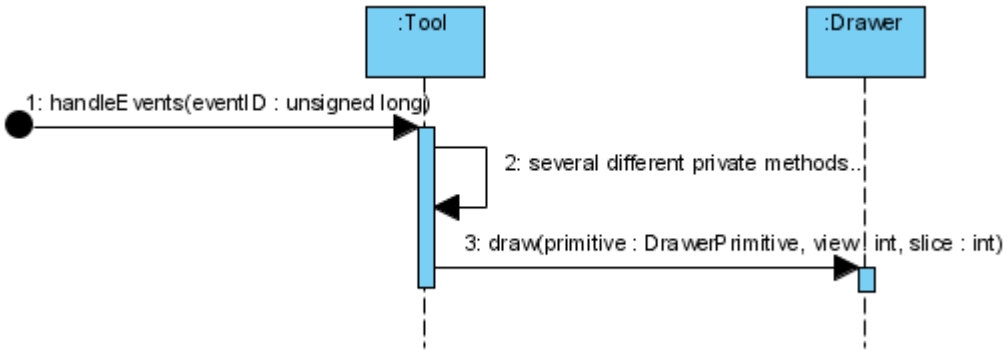


Figura 26 Creació d'una Tool segons la implementació original

Els events provenen de la classe `ToolProxy`, que els rep de la classe `Q2DViewer` i els reenvia a totes les `Tool`'s actives. Aquests events els processa la pròpia classe `Tool` i, amb crides als seus mètodes interns (diferents per a cada una), dibuixa la figura i fa els càlculs pertinents. Mentre dibuixa la figura utilitza el `Drawer` perquè la dibuixi per pantalla.

El següent diagrama mostra la creació d'una Tool segons el diagrama modificat i amb tots els elements nous que en formen part.

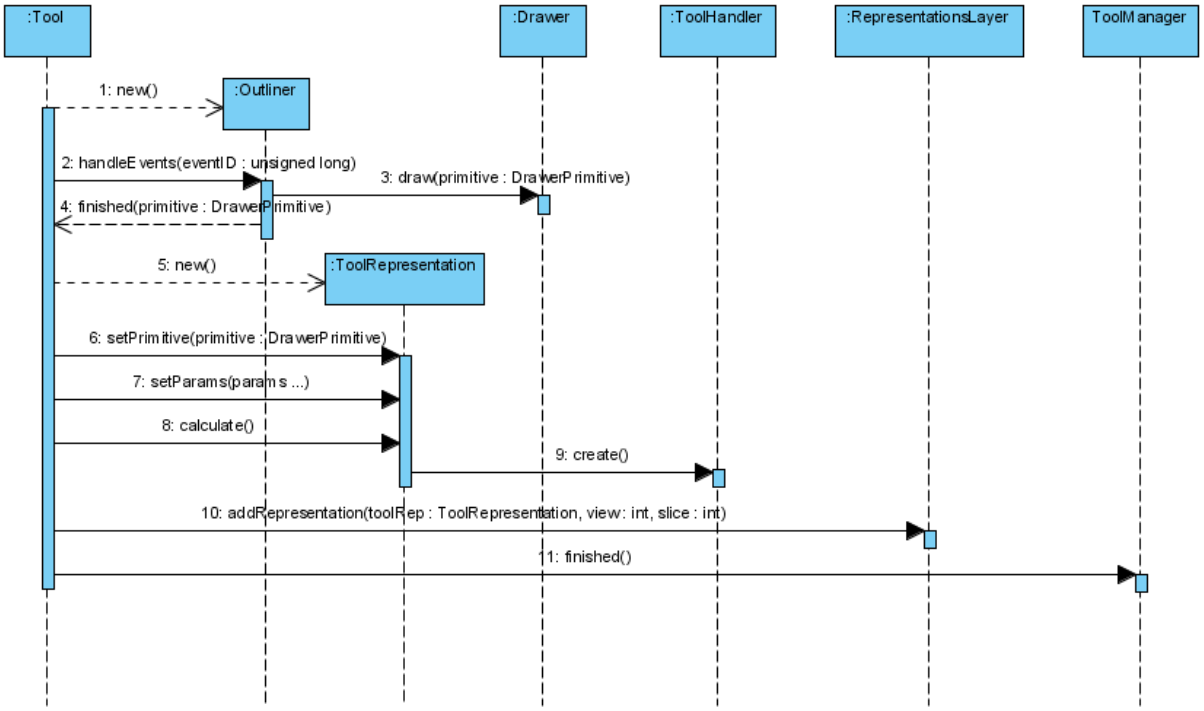


Figura 27 Creació d'una Tool segons la nova implementació

En aquest diagrama la classe `Tool` també rep els events reenviats des de `ToolProxy`. Es pot apreciar com la classe Tool crea un Outliner per dibuixar la figura, que és qui utilitza el Drawer, i una ToolRepresentation, encarregada de conservar la figura dibuixada i de fer el càlcul corresponent.

La classe `Tool` adquireix un nou comportament. Ara ja no és l'encarregada de "dibuixar" la figura, ni tan sols de fer els càlculs pertinents. Ara només gestiona tot el procés.

El procés de dibuixar la `Tool`, és a dir, de crear la, o les, `DrawerPrimitive` corresponent és responsabilitat de la classe `Outliner`. Aquesta és la seva única responsabilitat i, per tant, se la fa usable des de qualsevol altra `Tool`.

`ToolRepresentation` és ara l'encarregada de guardar la `DrawerPrimitive` que fa de dibuix i de fer els càlculs pertinents sobre aquesta. D'aquesta manera es converteix en l'element persistent que s'utilitzarà més tard per a l'edició d'una figura i, conseqüentment, per actualitzar el càlcul associat.

D'aquesta manera s'aconsegueix alliberar al `Drawer` de la responsabilitat d'emmagatzemar les `DrawerPrimitive` i de gestionar-ne la visibilitat. Aquest paper passa a `RepresentationsLayer`, que emmagatzemarà `ToolRepresentation`, amb les `DrawerPrimitive` incloses, i gestionarà la visibilitat de les `ToolRepresentation` senceres.

L'última crida, al `ToolManager`, serveix per indicar-li que la creació ha finalitzat i que s'ha de recuperar l'última `Tool` utilitzada de no – creació. Aquest és el pas de mode creació a mode edició.

5.2. Implementació de l'edició del mòdul de Tool's

Un cop implementada la modificació de mòdul original de Tool's, s'ha de començar a implementar la part d'edició. Aquesta part comprèn la modificació de les figures dibuixades a la pantalla i l'actualització del càlcul associat.

Es començarà identificant la interacció entre l'usuari i l'aplicació.

5.2.1. Fitxa de cas d'ús

Nom	**Edició d'una Tool**
Descripció	L'usuari canvia la forma d'una figura i canvia el càlcul associat.
Actor	Usuari
Precondició	Hi ha un estudi obert al visor 2D i una figura dibuixada, com a mínim.
Flux Principal	1. L'usuari s'acosta a una figura (vèrtex o perímetre) i prem el botó del mouse. 2. Sense deixar anar el botó, l'usuari mou el mouse per la pantalla modificant o movent la forma. 3. L'usuari deixa anar el botó del mouse per acabar l'edició. 4. Si l'edició ha canviat la forma de la figura a. Canvia el text del càlcul associat amb el nou càlcul 5. Altrament (només s'ha mogut), no es recalcula res.
Flux Alternatiu	---
Postcondició	La figura original ha canviat de forma i/o de posició. El càlcul es correspon amb la nova forma / posició.
Comentaris	---

5.2.2. Diagrama de seqüència

Aquest és el diagrama de seqüència que deriva de l'anterior fitxa de cas d'ús i del diagrama de classes proposat.

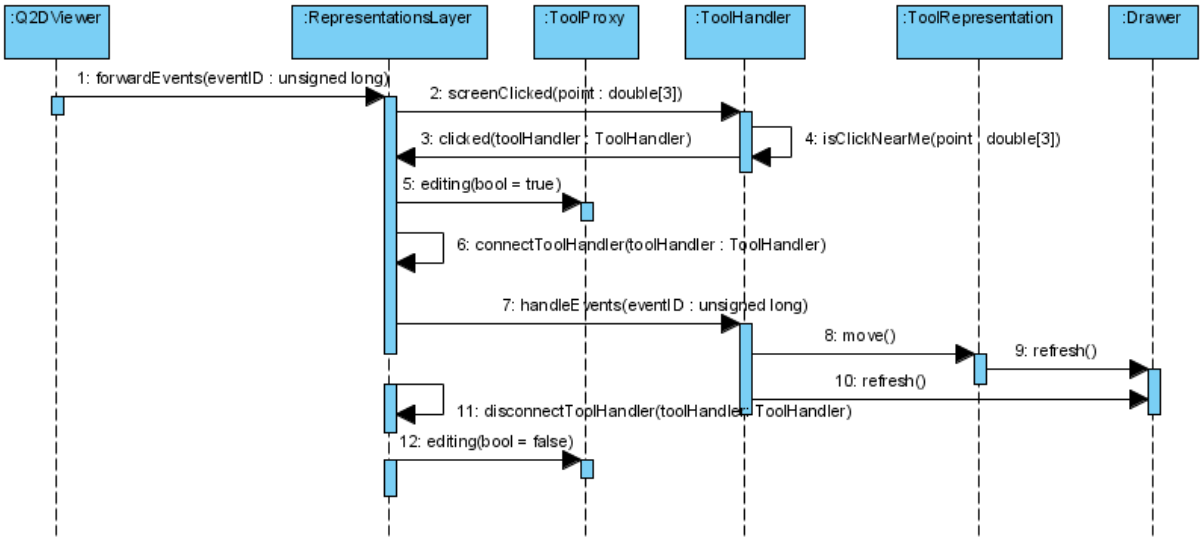


Figura 28 Diagrama de seqüència de l'edició d'una Tool

En aquest diagrama es pot apreciar com la classe `RepresentationsLayer` utilitza els events rebuts des del Q2DViewer per "demanar" a tots els `ToolHandler`'s si els correspon un click de mouse. En cas afirmatiu, el `ToolHandler` corresponent li envia la seva referència a `RepresentationsLayer`, aquesta crea les connexions necessàries per reenviar els events i li indica al `ToolProxy` que està editant i que, per tant, deixi d'enviar events a les Tool's actives. Un cop acabada l'edició, `RepresentationsLayer` desfà les connexions amb el `ToolHandler`, i indica al `ToolProxy` que ja ha deixat d'editar.

Aquest és un diagrama detallat de les connexions i comunicacions entre els tres elements involucrats:

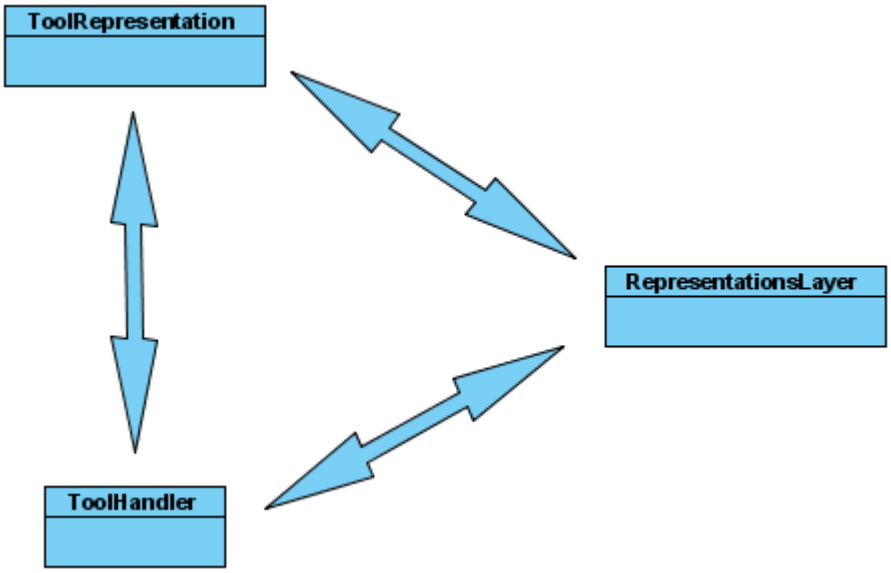


Figura 29 Diagrama amb les comunicacions direccionades entre classes

A continuació s'exposa una taula amb les principals comunicacions entre les classes participants. En la taula es mostra el nom dels senyals enviats, els rebuts, les classes que els envien i reben i la descripció.

Tots els senyals enviats i rebuts que s'expliquen a continuació són la base de l'edició de les `Tool`'s.

Signal	De	Slot	A	Descripció
screenClicked(double[3])	RepresentationsLayer	isClickNearMe(double[3])	ToolHandler	Envia el punt on hi ha hagut el click a la pantalla a tots els ToolHandler perquè comprovin si els correspon.
clicked(ToolHandler)	ToolHandler	connectToolHandler(ToolHandler)	RepresentationsLayer	Envia una referència a ell mateix perquè RepresentationsLayer connecti els senyals restants al ToolHandler i indiqui al ToolProxy que comença a editar.
forwardEvents(unsigned int)	RepresentationsLayer	handleEvents(unsigned int)	ToolHandler	Reenvia els events rebuts.
move(double[3])	ToolHandler	move(double[3])	ToolRepresentation	Mou el punt corresponent al ToolHandler editat al punt passat per paràmetre.
moveAllPoints(double[3])	ToolHandler	moveAllPoints(double[3])	ToolRepresentation	Mou tots els punts de la ToolRepresentation el desplaçament passat per paràmetre.
hide()/show()	ToolRepresentation	hide()/show()	ToolHandler	Amaga/Mostra el punt que fa de ToolHandler quan la ToolRepresentation rep el senyal pertinent.
deleteRepresentation()	RepresentationsLayer	deleteRepresentation()	ToolHandler	S'envia quan RepresentationsLayer rep el senyal d'eliminació des de teclat
destroy()	ToolHandler	destroy()	ToolRepresentation	Indica a la ToolRepresentation que ha d'enviar el senyal de destrucció.
dying(ToolRepresentation)	ToolRepresentation	removeRepresentation(ToolRepresentation)	RepresentationsLayer	Esborra la ToolRepresentation.

6. Proves dels canvis realitzats

Les proves dels canvis realitzats en la implementació d'aquesta modificació del mòdul de `Tool`'s s'han dut a terme per mitjà de la mateixa aplicació, ja que tota la implementació s'ha mantingut integrada dins la plataforma.

A continuació es mostren les `Tool`'s tal i com es veien al visor 2D de la plataforma abans de la modificació comparades amb com es veuen ara, després de la modificació.

La intenció d'aquestes captures és mostrar com les anteriors representacions de les `Tool`'s eren "fixes", és a dir, no es podien editar. I, a diferència d'aquestes, com les noves representacions són editables. S'intenta demostrar l'ús de l'edició en desplaçament de la figura i edició de la forma (per mitjà dels vèrtexs).

6.1. AngleTool o eina d'angles

Aquesta eina mesura angles donats tres punts dibuixats al visor. L'edició d'aquesta permet desplaçar la representació sencera (click a la línia i drag & drop amb el mouse) i canviar la posició dels seus extrems i del vèrtex d'intersecció (click als respectius *handlers* i drag & drop amb el mouse).

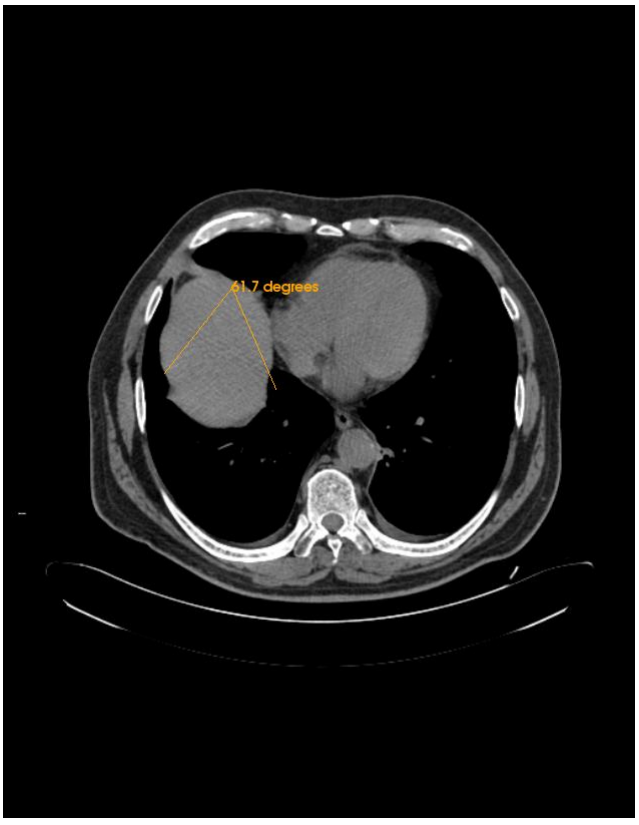


Figura 30 Representació original d'un angle

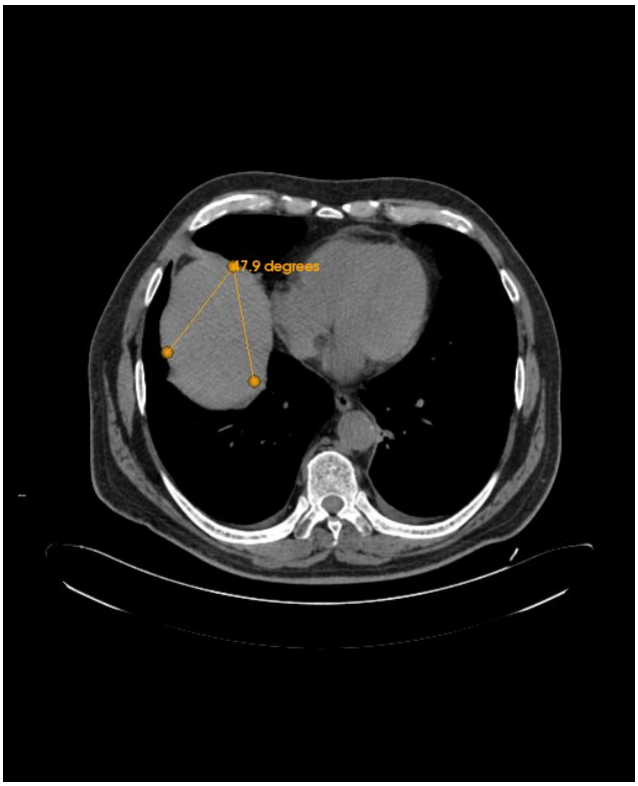


Figura 31 Representació editable d'un angle

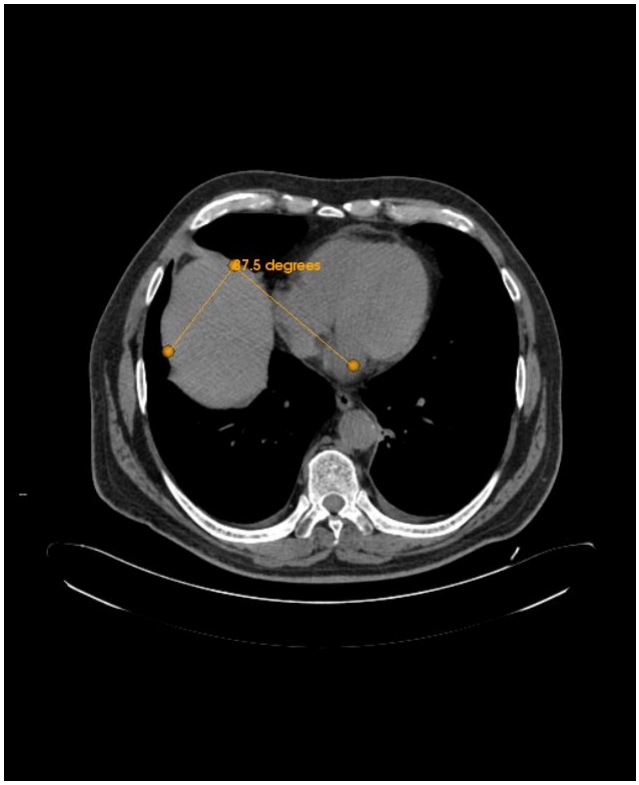


Figura 32 Edició d'un extrem de l'angle incrementant el valor

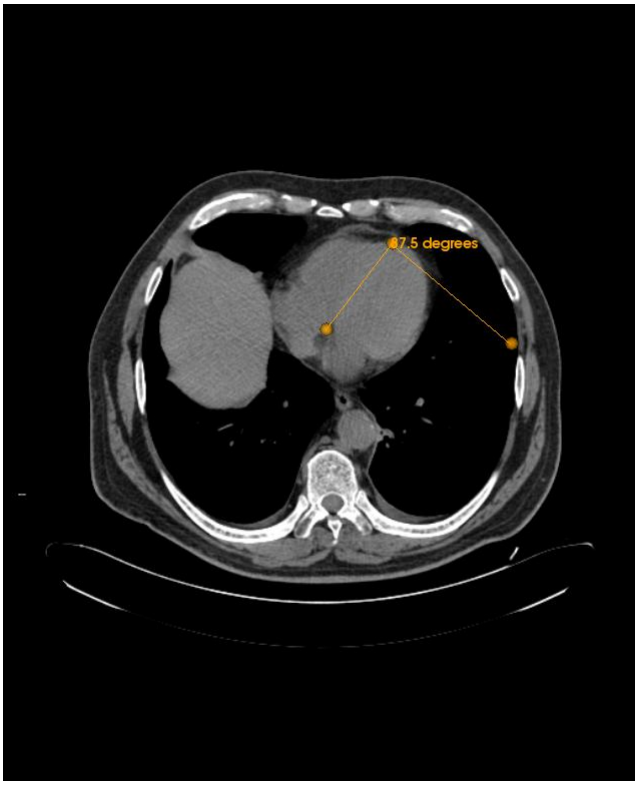


Figura 33 Desplaçament de la representació

6.2. DistanceTool o eina de distància

Aquesta eina mesura la distància entre dos punts dibuixats al visor. L'edició d'aquesta permet desplaçar la representació sencera (click a la línia i drag & drop amb el mouse) i canviar la posició dels seus extrems (click als respectius *handlers* i drag & drop amb el mouse).

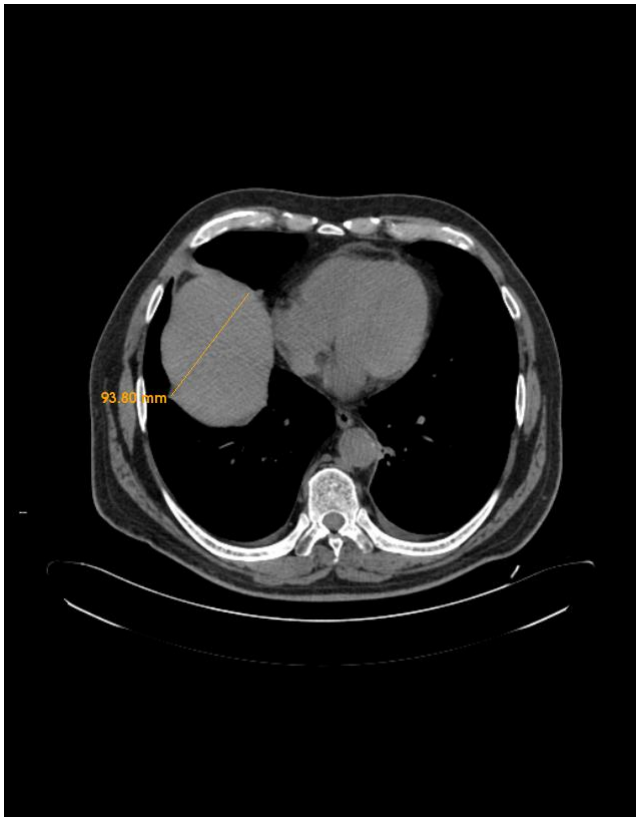


Figura 34 Representació original d'una distància

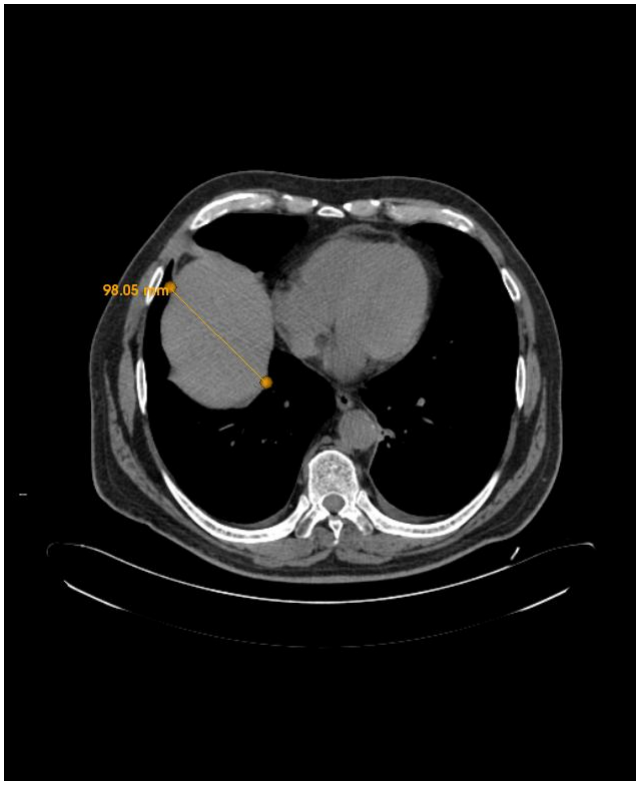


Figura 35 Representació editable d'una distància

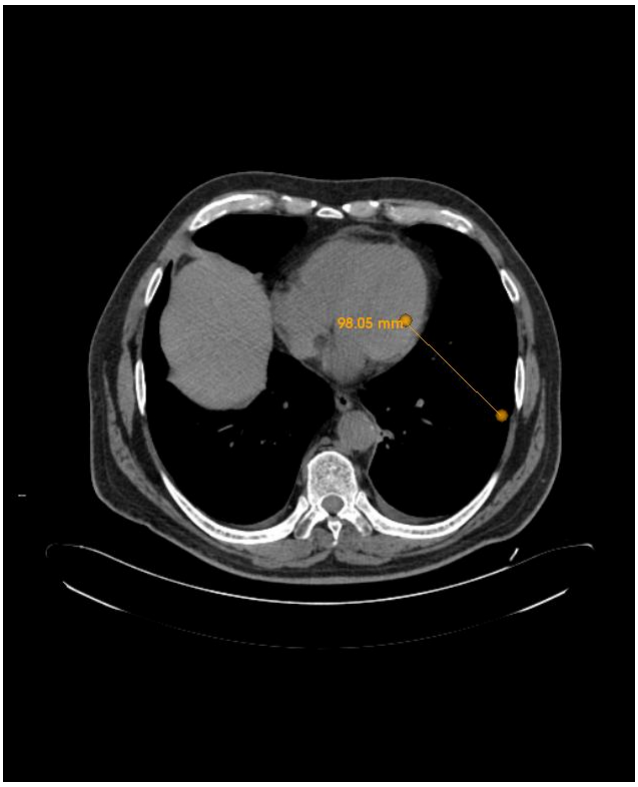


Figura 36 Desplaçament de la representació

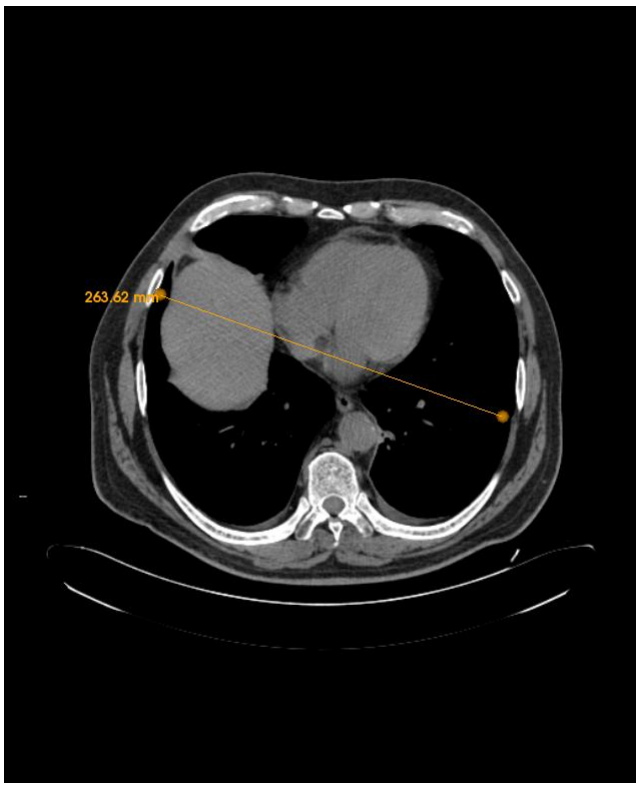


Figura 37 Edició dels extrems de la representació

6.3. NonClosedAngleTool o eina d'angles no tancats

Aquesta eina mesura angles donats dos segments dibuixats al visor. L'edició d'aquesta permet desplaçar la representació sencera (click a una de les línies i drag & drop amb el mouse) i canviar la posició dels extrems d'ambdues línies (click als respectius *handlers* i drag & drop amb el mouse).

Aquesta eina no existia, ha estat creada de nou en aquest projecte com a assistència a la resta d'eines per a un traumatòleg.

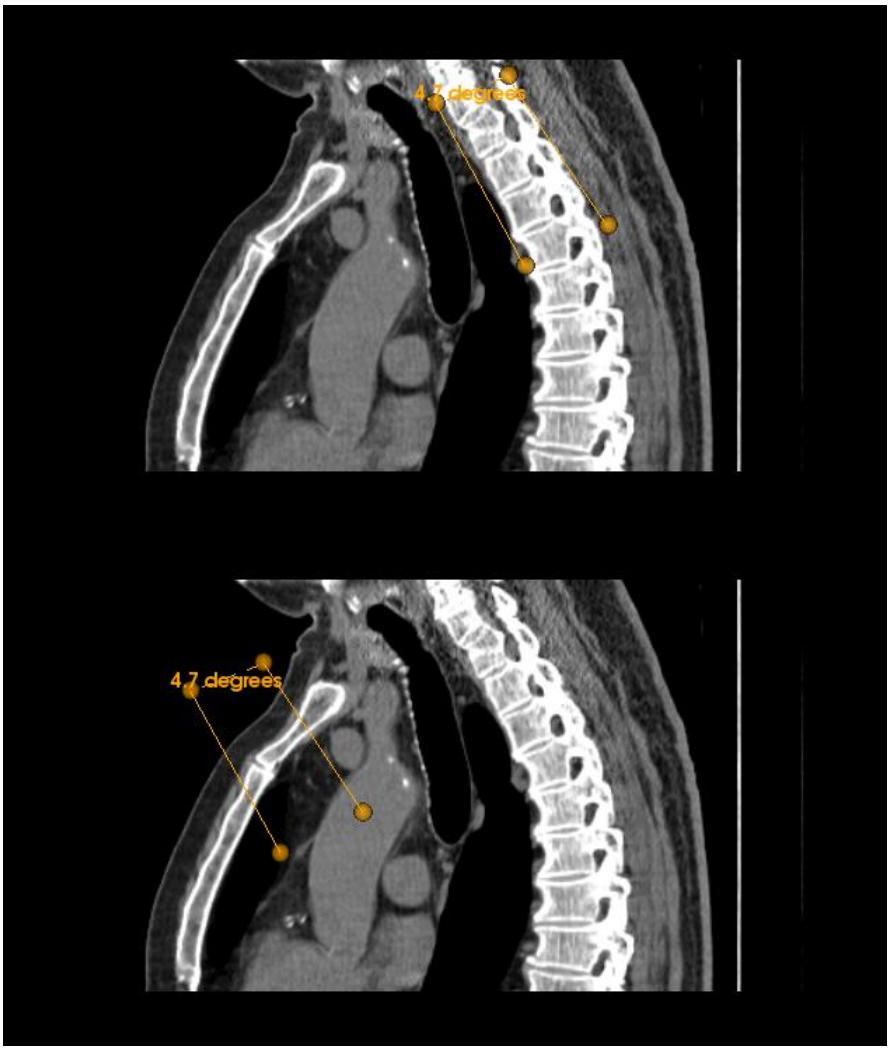


Figura 38 Representació editable d'un angle no tancat

Figura 39 Desplaçament de la representació

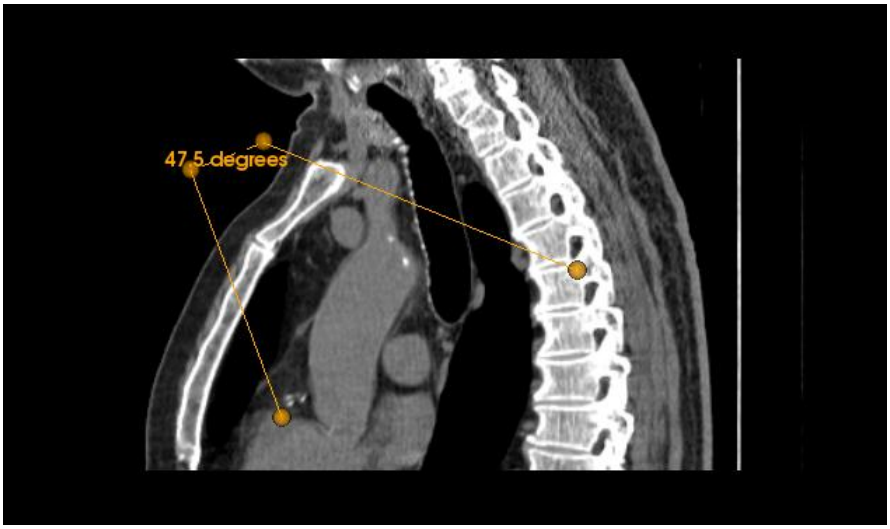


Figura 40 Edició dels extrems de la representació modificant-ne el valor

6.4. PolylineROITool o eina de regions d'interès

Aquesta eina mesura l'àrea d'una zona i la seva mitjana de grisos amb la desviació estàndard a partir d'una polilínia dibuixada al visor. L'edició d'aquesta permet desplaçar la representació sencera (click a la línia i drag & drop amb el mouse) i canviar la posició dels vèrtexs (click als respectius handlers i drag & drop amb el mouse).

Tal com es pot apreciar a la imatge, el càlcul de la desviació estàndard és fruit d'aquest projecte. Anteriorment només es feia el càlcul de l'àrea i de la mitjana de grisos.

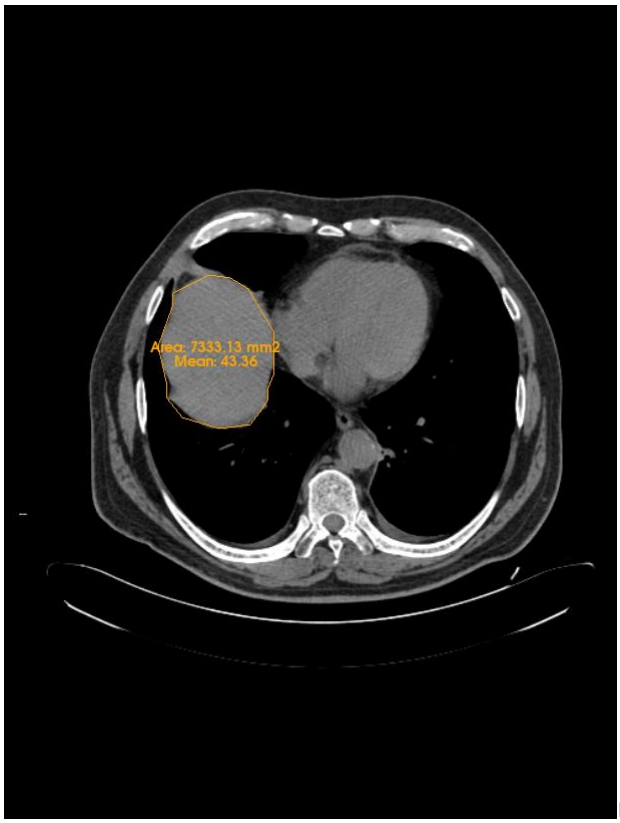


Figura 41 Representació original d'una regió d'interès

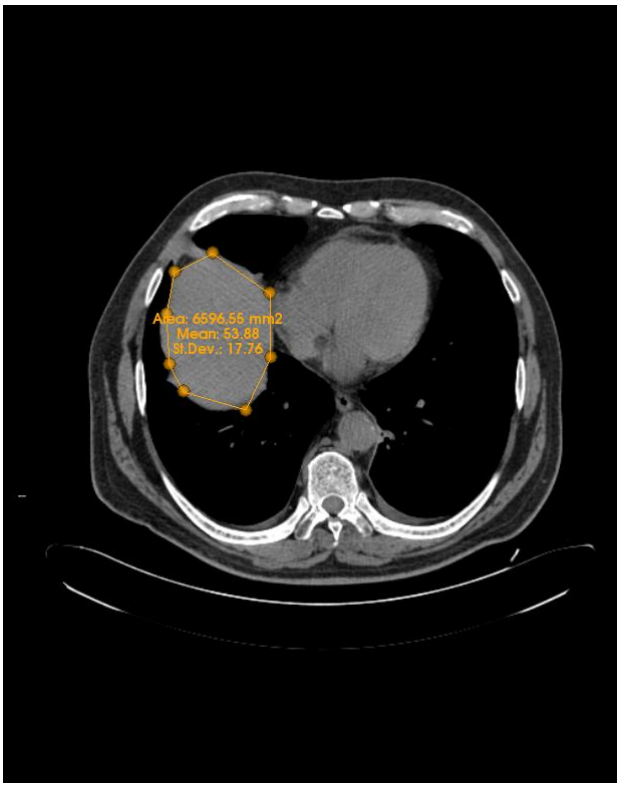


Figura 42 Representació editable d'una regió d'interès

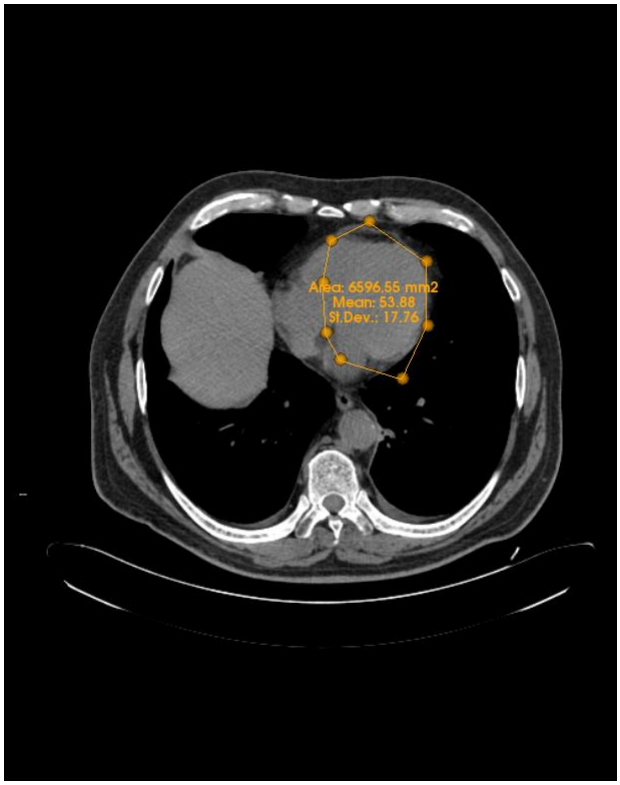


Figura 43 Desplaçament de la representació

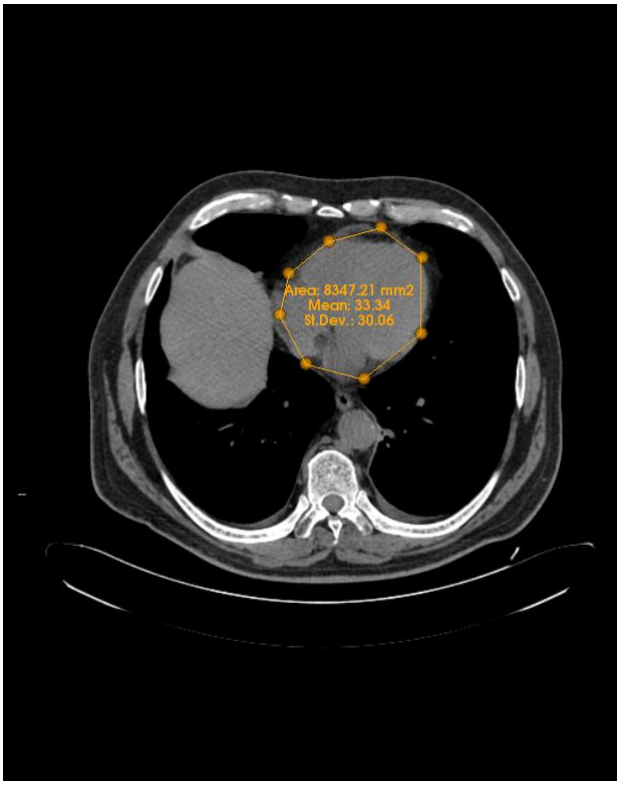


Figura 44 Edició dels vèrtexs de la representació

4. ExtractImageTool

1. Introducció

Un cop donada a les eines la capacitat de l'edició, ara cal crear les noves eines per donar suport a la inserció de pròtesis. Es necessita una eina capaç de visualitzar els ossos fracturats o dislocats al seu lloc habitual i, d'aquesta manera, poder definir les pròtesis i les mides necessàries. Aquestes operacions són les que representàvem en les Figures Figura 4 i Figura 5 de la introducció.

2. Objectius

L'objectiu d'aquest apartat és crear una eina nova donar suport als desplaçaments que cal realitzar sobre els ossos fracturats per col·locar-los en la posició correcte. Per crear una eina caldrà recuperar l'estructura del mòdul de `Tool`'s modificat en l'anterior apartat i crear els elements necessaris.

Aquesta nova eina és la que permet retallar un fragment de la imatge i moure'l i girar-lo a voluntat. Amb això s'aconsegueix situar l'os fracturat o dislocat al seu lloc.

3. Anàlisi de Requeriments

El que necessita el traumatòleg és "retallar" de la radiografia les parts que s'hauran de moure, és a dir, els ossos fracturats o dislocats. La imatge retallada no té perquè ser en forma circular o rectangular, per tant, ha de permetre dibuixar qualsevol tipus de forma. Un cop retallada, la imatge del visor ha de pintar la part retallada en negre. La imatge retallada ha de permetre qualsevol angle de rotació i qualsevol moviment per la pantalla.

4. Disseny

Per començar, el que s'ha de fer és recuperar i mantenir en ment el diagrama de classes del mòdul de `Tool`'s:

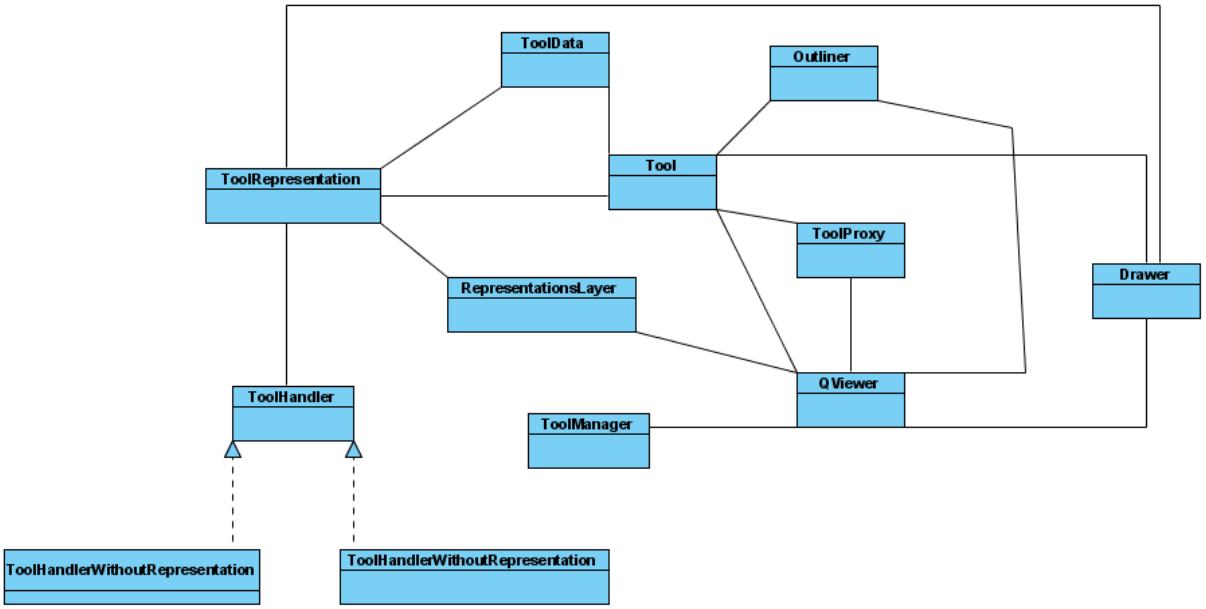


Figura 45 Diagrama de classes del mòdul de Tool's amb edició

Ara, caldrà recuperar el diagrama de seqüència de com funcionen les `Tool`'s, i que ens indicarà per cada element què s'ha d'implementar:

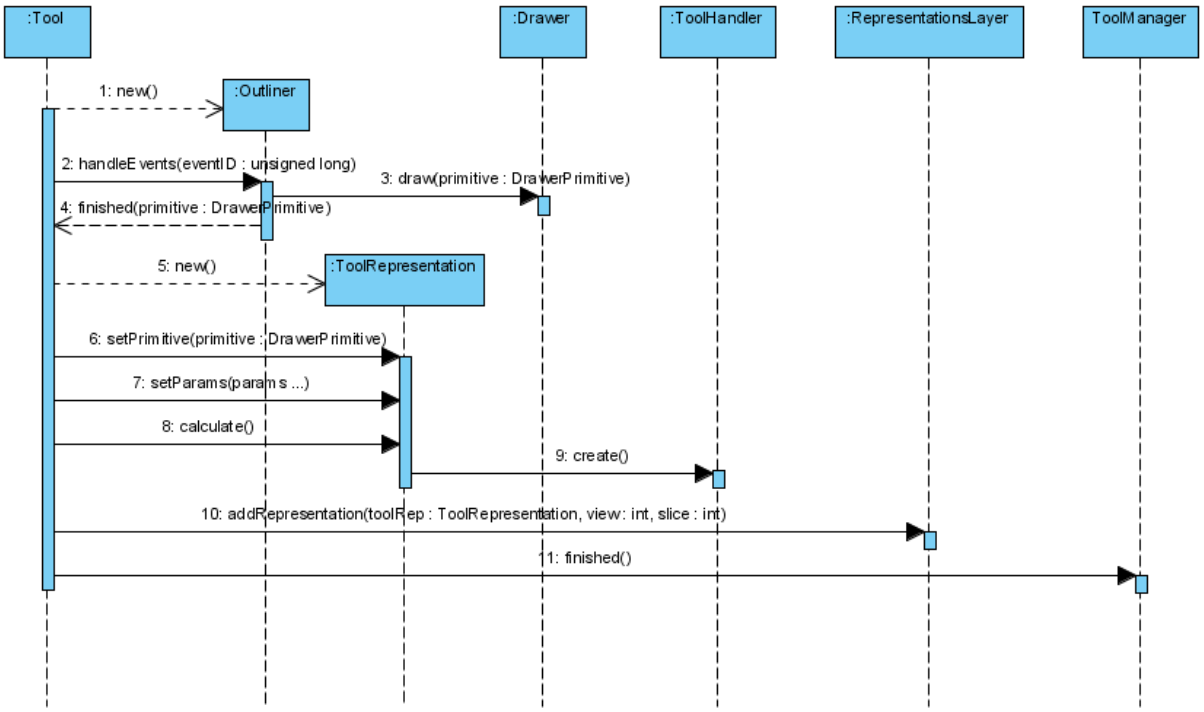


Figura 46 Diagrama de seqüència de la creació d'una Tool amb representació

Com que l'eina nova a construir té representació, és obvi que s'ha de seguir el diagrama anterior.

A més, cal tenir present el diagrama que mostra l'edició de les Tools.

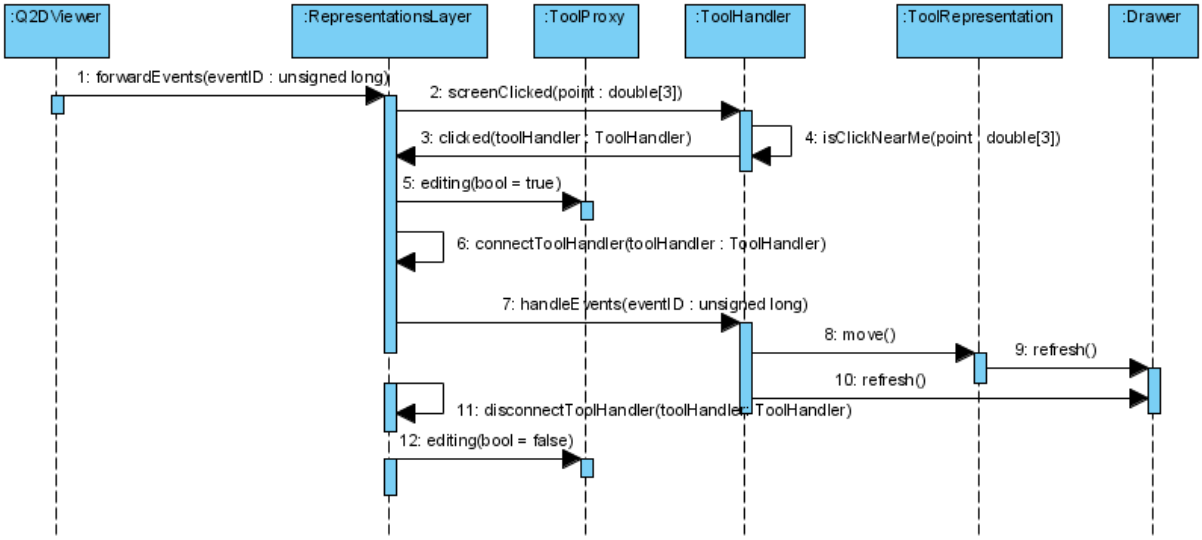


Figura 47 Diagrama de seqüència de l'edició d'una representació d'una Tool

5. Implementació

Partint del mòdul ja modificat i a punt i amb els diagrames de classe i de seqüència definitius, resultat de l'apartat anterior, ens disposem a crear la nova `Tool`.

El primer pas per a la implementació d'aquesta nova eina és definir les classes que caldrà crear.

5.1. Implementació de la creació de l'eina

Per començar, es definirà una fitxa de cas d'ús per poder apreciar els elements involucrats en la creació d'aquesta nova eina en concret.

5.1.1. Fitxa de cas d'ús

Nom	**Creació de la Tool**
Descripció	L'usuari prem el botó de la `Tool` a crear i la dibuixa a la pantalla
Actor	Usuari
Precondició	Hi ha un estudi obert al visor 2D
Flux Principal	1. L'usuari prem el botó de la `Tool` 2. L'usuari "dibuixa" el voltant del tros d'imatge a retallar 3. La `Tool` retalla la imatge i la mostra en pantalla
Flux Alternatiu	---
Postcondició	A la pantalla s'hi afegeix una polilínia amb una imatge al seu interior. Aquesta imatge correspon al que hi havia anteriorment en aquella posició del visor.
Comentaris	---

5.1.2. Identificació de les classes a implementar

Un cop tenim definits els passos per crear l'eina, cal identificar els elements que s'hauran de crear. Segons el diagrama de seqüència[9] general per totes les `Tool`'s, es pot veure que es necessiten les següents classes:

- `Tool`: classe principal, s'encarrega de gestionar la creació de les primitives i de l'element persistent.
- `Outliner`: s'encarregarà de dibuixar la polilínia que farà de límits de la imatge a retallar.
- `DrawerPrimitive`: es necessiten dues primitives: una per la línia que farà de límit i una segona per al imatge en si.
- `ToolRepresentation`: és l'element persistent. Guardarà les primitives.
- `ToolHandler`: s'encarregaran de moure la representació de fer-la girar.

Algunes derivades d'aquestes ja estan creades i en funcionament. Aquestes classes són:

[9] Figura 46, pàgina 57.

- `PolylineROIOutliner`: s'encarrega de dibuixar una polilínia en pantalla.
- `DrawerPolyline`: és la representació persistent d'una polilínia dibuixada.

Així doncs, caldrà definir i batejar la resta de classes:

- `ExtractImageTool`: gestionarà la creació de `PolylineROIOutliner`, per a què aquest dibuixi la polilínia.
- `DrawerImage`: guardarà la imatge i en serà la representació persistent.
- `ExtractImageToolRepresentation`: s'encarregarà de mantenir les primitives i gestionar-ne l'edició.
- `ToolHandlerRotation`: implementarà l'edició per rotació.

5.1.3. Diagrama de seqüència de la nova eina

Aquest diagrama de seqüència mostra la implementació de la creació d'aquesta nova eina.

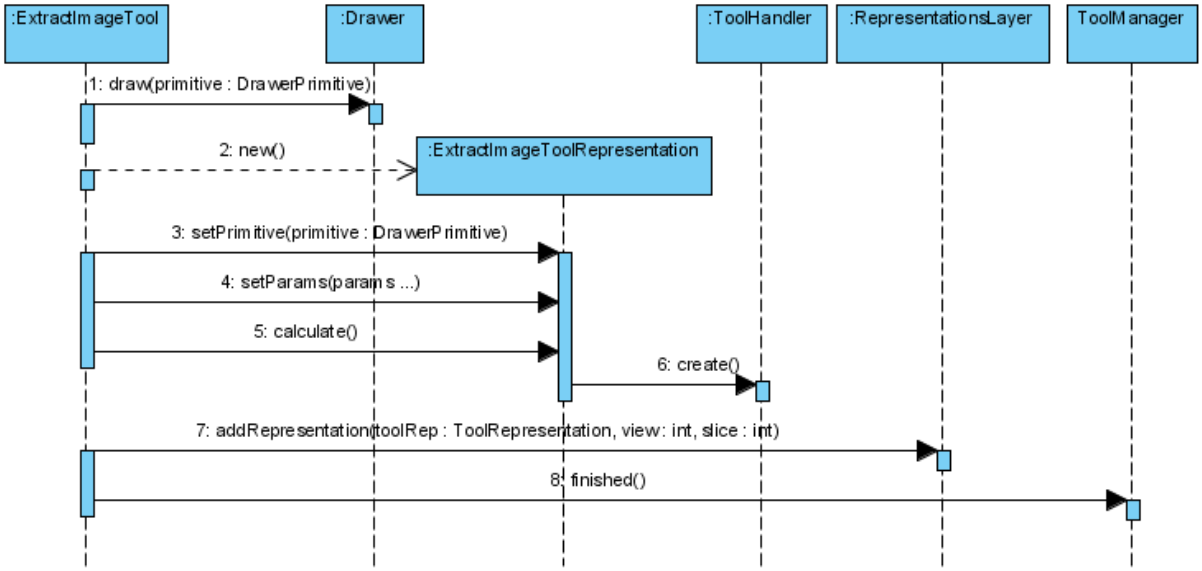


Figura 48 Diagrama de seqüència de la creació de l'eina ExtractImageTool

5.2. Implementació de l'edició de l'eina

Un cop l'eina ha estat creada, es necessitarà implementar la seva edició. Això es durà a terme per mitjà dels `ToolHandler`'s.

Aquesta `Tool` ha de permetre dos tipus d'edició. Les següents fitxes de cas d'ús ho il·lustren:

5.2.1. Fitxa de cas d'ús

Nom	**Desplaçament de l'eina**
Descripció	L'usuari canvia la posició de la representació de l'eina.
Actor	Usuari
Precondició	Hi ha un estudi obert al visor 2D i la representació de l'eina a la pantalla.

Flux Principal	1. L'usuari s'acosta al perímetre de la figura a l'interior i prem el botó del mouse. 2. Sense deixar anar el botó, l'usuari mou el mouse per la pantalla desplaçant la representació. 3. L'usuari deixa anar el botó del mouse.
Flux Alternatiu	---
Postcondició	La representació ha canviat de posició.
Comentaris	---

Nom	**Rotació de l'eina**
Descripció	L'usuari fa girar la representació de l'eina respecte el centre del polígon.
Actor	Usuari
Precondició	Hi ha un estudi obert al visor 2D i la representació de l'eina a la pantalla.
Flux Principal	1. L'usuari s'acosta a algun dels vèrtexs de la línia que envolta la imatge i prem el botó del mouse. 2. Sense deixar anar el botó, l'usuari mou el mouse per la pantalla descrivint trajectòries curvilínies fent girar la representació. 3. L'usuari deixa anar el botó del mouse.
Flux Alternatiu	---
Postcondició	La representació ha girat.
Comentaris	---

5.2.2. Identificació de les classes a implementar

Un cop identificats els tipus d'edició necessaris per a aquesta eina, cal identificar els elements a crear. Segons el diagrama de seqüència[10] general per totes les `Tool`'s, es pot veure que es necessiten la següents classes (la resta ja existeixen, pel pur funcionament del mòdul):

- `ToolHandler`: s'encarrega de l'edició de les `Tool`'s. Les classes que deriven d'aquesta s'encarreguen de l'edició concreta de les seves parts:
- `ToolHandlerWithoutRepresentation`: edita una `Tool` variant-ne la posició, és a dir, és la responsable del seu desplaçament.
- `ToolHandlerWithRepresentation`: edita una `Tool` variant-ne la forma.

D'aquestes implementacions de `ToolHandler` només una és útil: `ToolHandlerWithoutRepresentation`. L'altra no ens interessa, ja que la forma de la `Tool` només s'ha de definir una vegada.

El que sí fa falta i no existeix és un `ToolHandler` que ens permeti fer girar la figura. Així doncs, s'haurà de crear de nou:

- `ToolHandlerRotation`: implementarà l'edició per rotació.

[10] Figura 47, pàgina 58.

Cal tenir en compte que la implementació de `ToolHandlerRotation` afecta a les comunicacions entre `RepresentationsLayer`, `ToolHandler` i `ToolRepresentation`. Anteriorment(pàgina 48) s'havia mostrat una taula amb aquestes comunicacions, de manera que a continuació s'actualitza:

Signal	De	Slot	A	Descripció
`rotate(double)`	`ToolHandler`	`rotateRepresentation(double)`	`ToolRepresentation`	Fa girar tota la representació els graus definits en el paràmetre.

6. Proves dels canvis realitzats

La mateixa aplicació ens dóna el suport per a les proves a realitzar. Aquestes s'han dut a terme sobre estudis reals de pacients.

A continuació es mostren algunes radiografies d'aquests estudis i l'aplicació de l'eina creada en aquest apartat.

Figura 49 Imatge original d'una pelvis

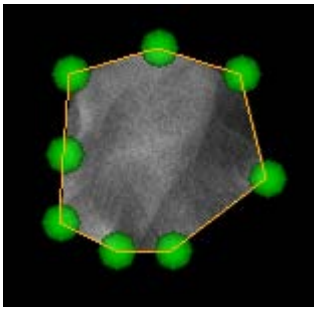


Figura 50 Retall de la imatge anterior

Figura 51 Imatge d'una pelvis amb el fèmur dret fora de lloc

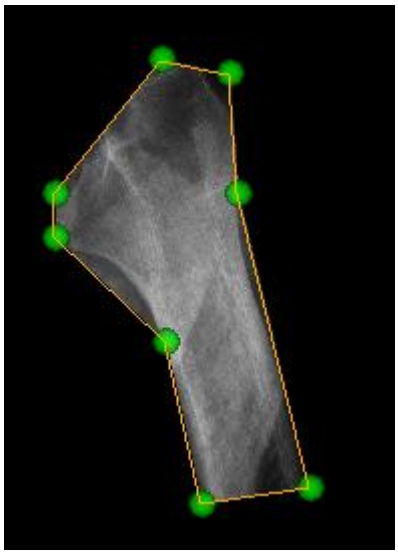


Figura 52 Retall del fèmur dret de la figura anterior

5. SuperimposeImageTool

1. Introducció

Una vegada el traumatòleg ha mogut els ossos fracturats i els ha col·locat en la posició que ell considera la correcta li cal fer una comprovació per veure que realment és correcte. Normalment, el que es fa es comparar-ho amb la part no lesionada del pacient. Se suposa que les dues parts han de ser simètriques.

2. Objectius

L'objectiu d'aquest apartat és crear una eina nova que permeti comprovar que el desplaçament realitzat sigui correcte. Per crear una eina caldrà recuperar l'estructura del mòdul de `Tool`'s modificat en l'anterior apartat i crear els elements necessaris.

Aquesta vegada, l'eina ha de permetre copiar la imatge que s'està veient en el visor i girar-la horitzontalment 180º per així comprovar que la nova posició coincideixi amb la part simètrica del cos del pacient.

3. Anàlisi de Requeriments

El que necessita el traumatòleg és "girar" la radiografia. Aquesta eina ha de permetre sobreposar la radiografia, girada 180º, i poder moure-la a plaer per aconseguir una vista simètrica. Per aconseguir això, la radiografia sobreposada haurà de ser semi-transparent.

4. Disseny

Per començar, el que s'ha de fer és recuperar i mantenir en ment el diagrama de classes del mòdul de `Tool`'s:

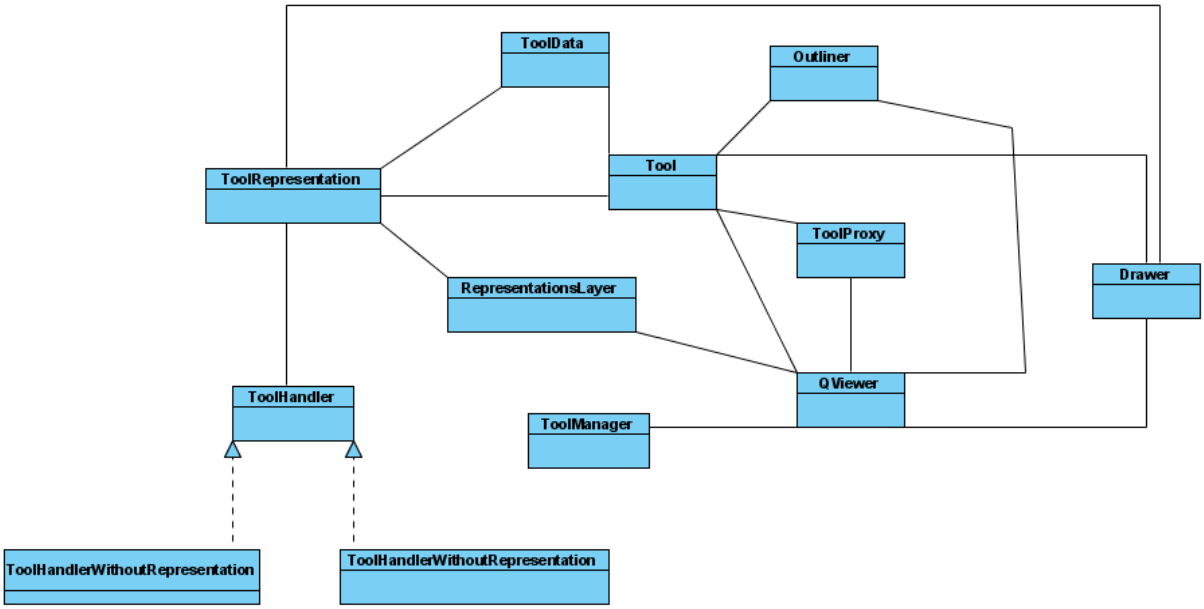


Figura 53 Diagrama de classes del mòdul de Tool's amb edició

Ara, caldrà recuperar el diagrama de seqüència de com funcionen les `Tool`'s, i que ens indicarà per cada element què s'ha d'implementar:

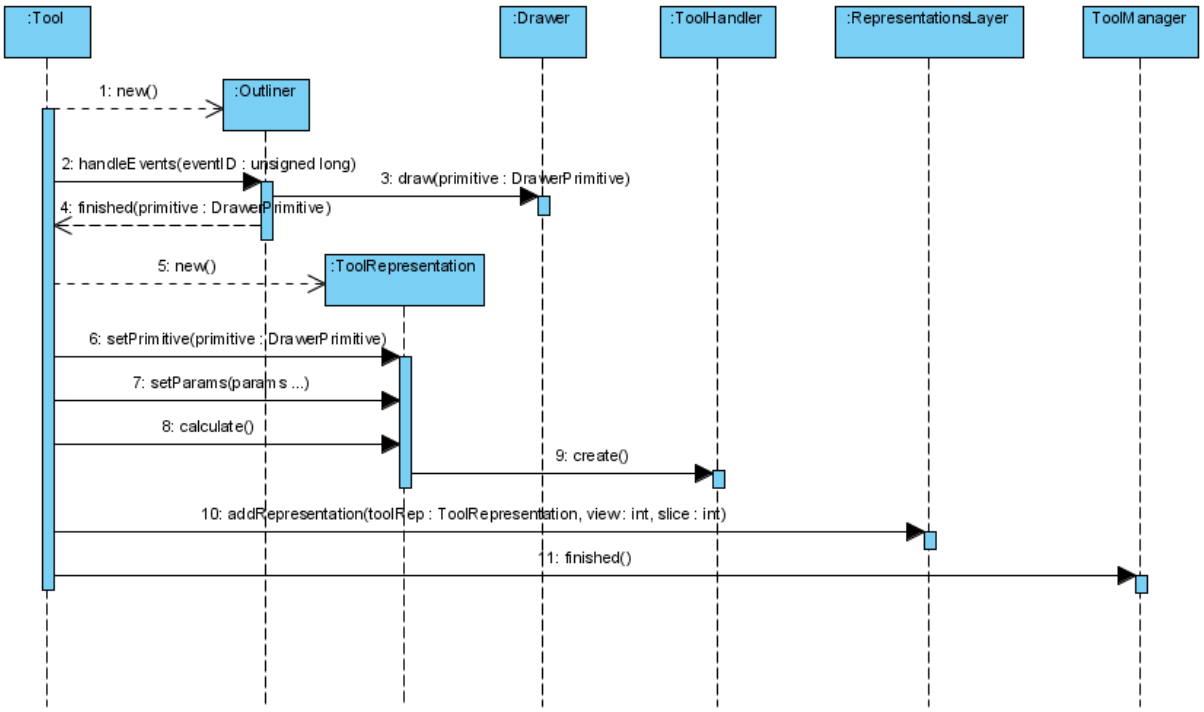


Figura 54 Diagrama de seqüència de la creació d'una Tool amb representació

Com que l'eina nova a construir té representació, és obvi que s'ha de seguir el diagrama anterior.

A més, cal tenir present el diagrama que mostra l'edició de les Tools.

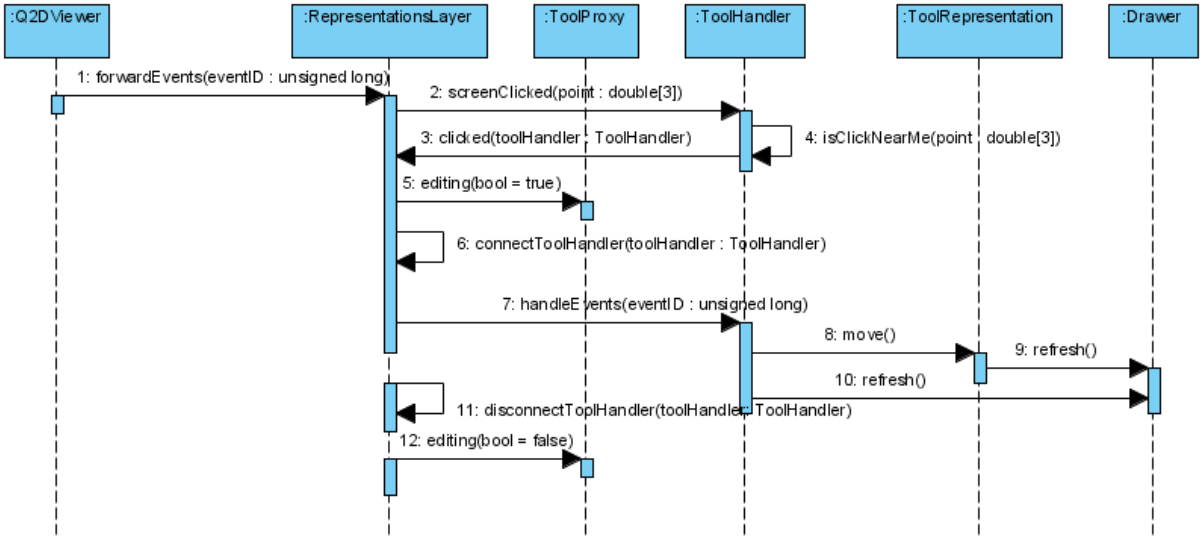


Figura 55 Diagrama de seqüència de l'edició d'una representació d'una Tool

5. Implementació

Partint del mòdul ja modificat i a punt i amb els diagrames de classe i de seqüència definitius, resultat de l'apartat anterior, ens disposem a crear la nova `Tool`.

El primer pas per a la implementació d'aquesta nova eina és definir les classes que caldrà crear.

5.1. Implementació de la creació de l'eina

Per començar, es definirà una fitxa de cas d'ús per poder apreciar els elements involucrats en la creació d'aquesta nova eina en concret.

5.1.1. Fitxa de cas d'ús

Nom	**Creació de la Tool**
Descripció	L'usuari prem el botó de la `Tool` a crear
Actor	Usuari
Precondició	Hi ha un estudi obert al visor 2D
Flux Principal	1. L'usuari prem el botó de la `Tool` 2. Apareix la radiografia duplicada i girada horitzontalment a la pantalla
Flux Alternatiu	---
Postcondició	A la pantalla hi apareix la imatge que es veia al visor invertida horitzontalment i semi-transparent.
Comentaris	---

5.1.2. Identificació de les classes a implementar

Un cop tenim definits els passos per crear l'eina, cal identificar els elements que s'hauran de crear. Segons el diagrama de seqüència[11] general per totes les `Tool`'s, es pot veure que es necessiten les següents classes:

- `Tool`: classe principal, s'encarrega de gestionar la creació de les primitives i de l'element persistent.
- `Outliner`: no és necessari.
- `DrawerPrimitive`: es necessita una primitiva per guardar la imatge.
- `ToolRepresentation`: és l'element persistent. Guardarà la primitiva.
- `ToolHandler`: s'encarregaran de moure la imatge.

Algunes derivades d'aquestes ja estan creades i en funcionament. Aquestes classes són:

- `DrawerImage`: creada en l'anterior eina. Guardarà la imatge invertida.
- `ToolHandlerWithoutRepresentation`: implementarà l'edició de moviment.

[11] Figura 46, pàgina 57.

Així doncs, caldrà definir i batejar la resta de classes:

- `SuperImposeImageTool`: gestionarà la creació de la imatge.
- `SuperImposeImageToolRepresentation`: s'encarregarà de mantenir la primitiva i gestionar-ne l'edició.

5.1.3. Diagrama de seqüència de la nova eina

Aquest diagrama de seqüència mostra la implementació de la creació d'aquesta nova eina.

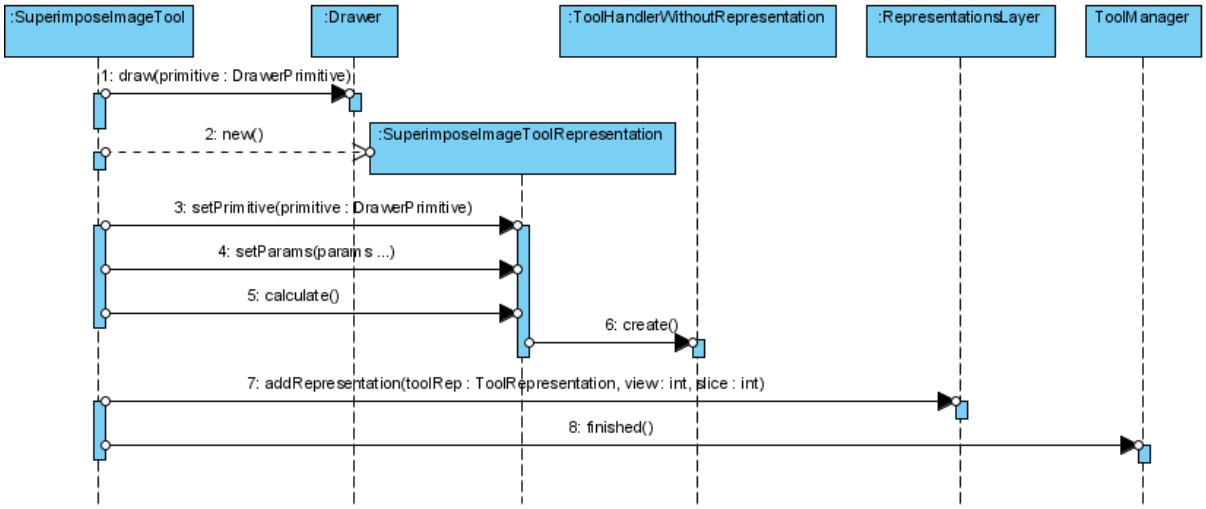


Figura 56 Diagrama de seqüència de la creació de l'eina SuperimposeImageTool

5.2. Implementació de l'edició de l'eina

Un cop l'eina ha estat creada, es necessitarà implementar la seva edició. Això es durà a terme per mitjà dels `ToolHandler`'s.

Aquesta `Tool` ha de permetre un sol tipus d'edició: el moviment.

5.2.1. Fitxa de cas d'ús

Nom	**Desplaçament de l'eina**
Descripció	L'usuari canvia la posició de la representació de l'eina.
Actor	Usuari
Precondició	Hi ha un estudi obert al visor 2D i la representació de l'eina a la pantalla.
Flux Principal	1. L'usuari se situa sobre la imatge i prem el botó del mouse. 2. Sense deixar anar el botó, l'usuari mou el mouse per la pantalla desplaçant la representació. 3. L'usuari deixa anar el botó del mouse.
Flux Alternatiu	---
Postcondició	La representació ha canviat de posició.
Comentaris	---

5.2.2. Identificació de les classes a implementar

Un cop identificats els tipus d'edició necessaris per a aquesta eina, cal identificar els elements a crear. Segons el diagrama de seqüència[12] general per totes les `Tool`'s, es pot veure que es necessiten la següents classes (la resta ja existeixen, pel pur funcionament del mòdul):

- `ToolHandler`: s'encarrega de l'edició de les `Tool`'s. Les classes que deriven d'aquesta s'encarreguen de l'edició concreta de les seves parts:
- `ToolHandlerWithoutRepresentation`: edita una `Tool` variant-ne la posició, és a dir, és la responsable del seu desplaçament.
- `ToolHandlerWithRepresentation`: edita una `Tool` variant-ne la forma.
- `ToolHandlerRotation`: edita una `Tool` variant-ne la rotació original.

D'aquestes implementacions de `ToolHandler` només una és útil: `ToolHandlerWithoutRepresentation`. La resta no ens interessa, ja que la forma de la `Tool` només s'ha de definir una vegada i no ens és útil la rotació. A més, amb aquesta en fem prou.

Aquesta classe ja està implementada i, a més, ja s'utilitza en altres eines. Per aquest motiu no és necessari dissenyar cap implementació nova.

[12] Figura 47, pàgina 58.

6. Proves dels canvis realitzats

La mateixa aplicació ens dóna el suport per a les proves a realitzar. Aquestes s'han dut a terme sobre estudis reals de pacients.

A continuació es mostren algunes radiografies d'aquests estudis i l'aplicació de l'eina creada en aquest apartat.

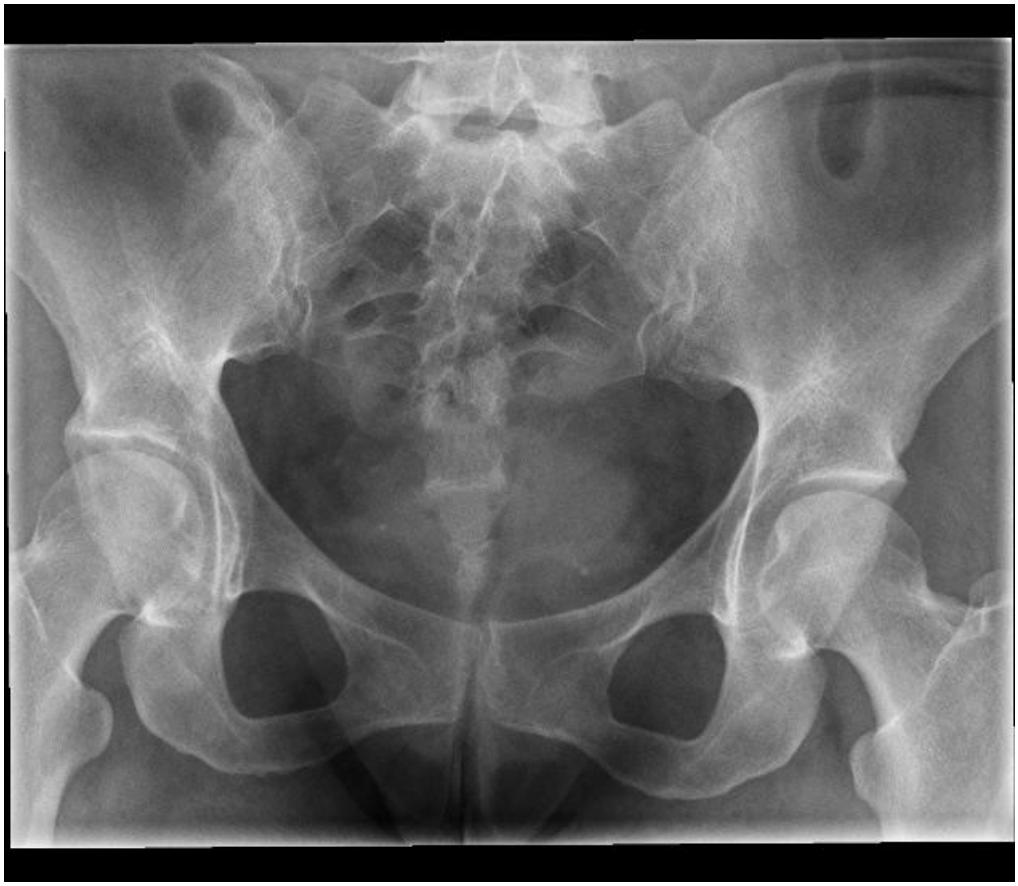


Figura 57 Radiografia d'una pelvis en un visor 2D

Figura 58 Radiografia de la pelvis amb la imatge sobreposada i invertida.

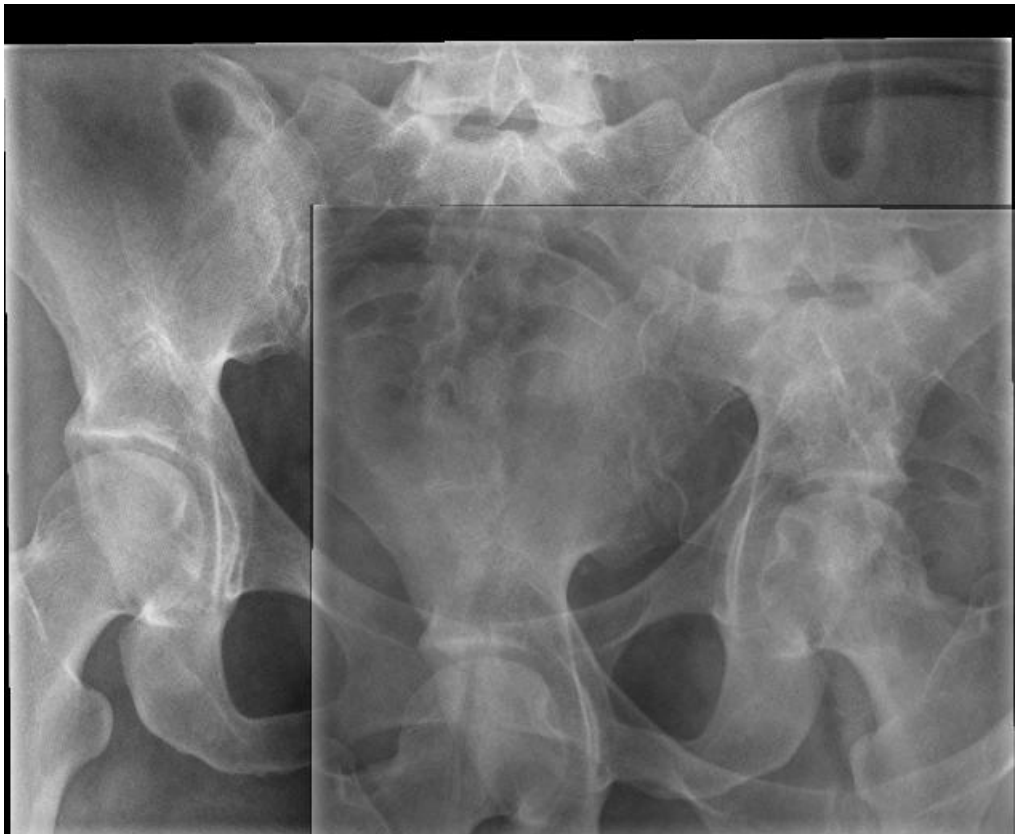


Figura 59 Radiografia de la pelvis amb la imatge sobreposada desplaçada.

6. Conclusions

1. Conclusions Generals

En iniciar aquest projecte es van proposar una sèrie d'objectius. Aquests s'han assolit en gran part.

L'objectiu principal era començar a desenvolupar un mòdul específic per a l'aplicació que servís de suport a la inserció de pròtesis. Per a assolir aquest objectiu general, abans se n'havien d'assolir d'altres, més concrets.

La valoració és positiva, ja que aquests objectius s'han assolit satisfactòriament. A continuació s'exposen els objectius i la seva resolució.

- **Edició de primitives**

  Les eines que s'havien de crear més endavant com a suport a la inserció de pròtesis necessiten poder modificar i moure la seva representació en pantalla un cop creades.

  Per a poder introduir aquesta modificació s'ha hagut d'ampliar el mòdul de `Tool`'s, canviant la filosofia d'alguns elements, reorganitzant les seves responsabilitats, ampliant-ne el cicle de vida, etc. A més s'ha canviat la interacció de l'usuari amb la pantalla.

- **Retallar una imatge**

  S'havia de poder retallar una porció de la imatge visualitzada, extreure-la i col·locar-la en la posició correcta (sana) per definir la pròtesi necessària.

  Seguint el model de `Tool`'s creat anteriorment i modificant recentment, s'ha hagut de crear una nova eina capaç de retallar una imatge d'un volum. S'extreu a partir d'una polilínia, i gràcies als vèrtexs d'aquesta s'utilitza la funcionalitat de rotació.

- **Sobreposar una imatge amb una rotació de 180º...**

  ... per avaluar la correcció de les imatges retallades i col·locades de nou.

  Es necessitava girar 180º horitzontalment la imatge visualitzada i poder sobreposar-la sobre l'original per així comparar si les correccions dutes a terme eren prou correctes (simètriques).

  Seguint el model de `Tool`'s, s'ha creat una nova eina que crea una còpia de la imatge visualitzada, li aplica certa transparència, una rotació de 180º horitzontal i la mostra en el mateix visualitzador. A més, permet moure la imatge per a col·locar-la i ajustar-la a la zona correcta.

- **Integrar-ho a la plataforma Starviewer**

  Tot el procés de disseny i implementació s'ha dut a terme sobre la plataforma en qüestió. Les proves també han estat executades per mitjà de la plataforma.

Cal remarcar que aquest projecte no només ha tractat d'implementar eines noves per al tractament d'imatges. Tota la part de disseny, implementació i proves de la modificació d'un mòdul tant utilitzat com és el de `Tool`'s en l'ús habitual de la plataforma, ha comportat molta feina i molt temps.

Només de començar, i un cop familiaritzat amb la plataforma i el mòdul en concret, es van haver de dur a terme diverses reunions amb l'equip de desenvolupament per discutir el camí a seguir d'aquestes modificacions, tant a l'hora de la seva implementació com a l'hora de l'interacció amb l'usuari.

Donat que la implementació d'aquestes modificacions comportaven molt temps i alhora dificultats, s'ha optat per minimitzar-les i centrar-se, només, en les necessàries per al correcte ús d'aquestes.

Els resultats d'aquestes modificacions està, actualment, en funcionament real en la mateixa plataforma.

Una intenció original en aquest projecte era poder projectar les pròtesis sobre el visor. Per fer això, però s'ha de disposar d'una base de dades de pròtesis per, així, seleccionar la pròtesis adequada en tot moment. Això, però, no ha estat possible donat que les cases comercials s'han negat a facilitar les plantilles d'aquestes pròtesis.

2. Ampliacions i millores futures

La introducció de modificacions en el mòdul de `Tool`'s de l'aplicació ha estat un gran pas cap a l'hora d'interactuar amb la plataforma al permetre modificar les representacions de les eines un cop ja generades. Aquestes modificacions, però, no han estat acabades del tot, s'han cenyit només a l'ús que se n'havia de fer en aquest projecte.

Una possible millora seria acabar de definir l'edició de les representacions de les eines un cop creades. Definir tecles ràpides per a la seva selecció (p. ex: Ctrl+click per a la selecció múltiple, etc.), permetre el desplaçament del text en les eines que mostren càlculs, etc.

Actualment ja estic treballant en aquesta part per així tenir-la del tot enllestida d'aquí a un temps.

3. Conclusions Personals

A nivell personal, aquest projecte m'ha permès acostar-me a una plataforma de visualització de dades científiques, un món que, tot i interessant, encara no havia conegut.

Sempre he tingut simpatia a la informàtica dedicada a la visualització i aquest projecte m'ha permès experimentar l'aplicació pràctica de la implementació d'eines per al diagnòstic de malalties. Durant el temps dedicat he conegut llibreries que s'utilitzen en el procés de visualització i les he utilitzat per a implementar les solucions als objectius proposats.

A més, he pogut aprofundir una mica en l'ús del llenguatge de programació C++ i la llibreria Qt. El primer, molt interessant per ser un dels llenguatges més utilitzats actualment en la implementació de solucions en el camp de la visualització, i el segon, per ser de gran ajuda en la programació en C++ i en la creació d'interfícies gràfiques, un terreny poc explorat en el transcurs de la carrera.

Hi ha un altre aspecte de la programació que no es treballa massa durant el transcurs de la carrera, aquest és el treball en equip en una aplicació de grans dimensions i especialment complexa. Això resulta en una nova forma de treball: l'ús d'un repositori i del control de versions. Ha estat un aspecte nou que he après i que valoro molt positivament, ja que en el món professional és habitual trobar-ho.

Finalment, cal esmentar que aquest projecte és només una part d'una plataforma de visualització que ja està en funcionament en diferents hospitals del país. Valoro molt positivament l'oportunitat d'haver-me pogut unir a l'equip de desenvolupament d'aquesta i d'haver desenvolupat sinó tot, les bases, d'un mòdul nou de l'aplicació per a un sector de professionals a qui encara falta donar suport.

7. Bibliografia

Referències bibliogràfiques per a documentació sobre les llibreries:

Insight Toolkit:

- http://www.itk.org
- Ibáñez L.,Schroeder W., Ng L., Cates J. (2005). *The ITK Software Guide (2nd ed.):* http://biznetnetworks.dl.sourceforge.net/project/itk/itk/2.4/ItkSoftwareGuide-2.4.0.pdf

Visualization Toolkit:

- http://www.vtk.org
- Schroeder W., Martin K., Lorensen B. (2002-2004). *The Visualization Toolkit (3rd ed.).* Kitware
- VTK Mailing List: http://public.kitware.*com*/pipermail/vtkusers/

Qt:

- Qt Reference Documentation: http://doc.trolltech.com/4.0/index.html

Altres pàgines d'interès:

Altres fonts per càlculs matemàtics, informació addicional, etc:

- eXtreme Programming: http://www.monografias.com/trabajos51/programacion-extrema/programacion-extrema.shtml
- Point in Polygon: http://www.visibone.com/inpoly
- Altres solucions matemàtiques: http://mathworld.wolfram.com/