FACULTADE DE INFORMÁTICA

Departamento de Computación

PROYECTO DE FIN DE CARRERA DE INGENIERÍA TÉCNICA EN

INFORMÁTICA DE SISTEMAS

Filtro gráfico para la corrección de distorsiones ópticas

Autor: Freire Riobó, Javier

Tutor: Molinelli Barba, José María

A Coruña, Septiembre de 2004

D. / D^{a}. José María Molinelli Barba

Profesor / a del Departamento de Computación

HACE CONSTAR:

Que el alumno / a *Javier Freire Riobó* ha realizado el Proyecto Fin de Carrera titulado *Filtro gráfico para la corrección de distorsiones ópticas* de acuerdo a la descripción inicialmente propuesta, por lo que, como director / tutor del mismo, autorizo su presentación para que el Proyecto sea defendido en la presente convocatoria.

La Coruña, 24 de septiembre de 2004

Fdo.:

.

Titulo del proyecto

Filtro gráfico para la corrección de distorsiones ópticas

Clase a la que pertenece

Proyecto clásico de Ingeniería Técnica en Informática de Sistemas

Nombre del alumno

Javier Freire Riobó

Nombre del director

José María Molinelli Barba

Miembros del tribunal

Fecha de lectura y defensa

Calificación obtenida

Agradecimientos

Me gustaría dedicar este proyecto a todo aquel que ha estado a mi lado, apoyándome durante el largo y arduo período de su desarrollo. Muy especialmente a mi familia y a mi novia.

A un nivel no tan emotivo sino más bien profesional, querría dedicárselo a Juan Penalta, por sus consejos a nivel matemático.

Prefacio

Los objetivos fotográficos -sobre todo los de distancia focal corta ("gran angular") y los de distancia focal variable ("zoom")- tienden a distorsionar la imagen captada, de forma que las líneas rectas que no pasan por el centro de la imagen aparecen curvadas. Es lo que se suele conocer como distorsión "de barril" o "de almohadilla" (según el sentido de curvatura).

Se trata de construir un filtro digital que corrija este efecto sobre imágenes almacenadas en formato "TIFF".

Para ello habrá que realizar previamente un estudio empírico sobre las distorsiones provocadas por diferentes tipos de lentes fotográficas.

El programa que aplica el filtro contendrá una base de datos de lentes para aplicar de forma automática la corrección, pero también podrá trabajar de forma asistida por el operador (que le indicaría en este caso cuáles son las líneas rectas que aparecen curvadas en la imagen). Asimismo aceptará la inclusión (por parte del usuario) de nuevas lentes en su base de datos y permitirá afinar manualmente las correcciones efectuadas de forma automática.

Índice general

Parte I

El problema y la solución

Capítulo 1

El problema

1.1. Descripción del problema

1.1.1. El efecto "barril" y el efecto "almohadilla"

Los aficionados a la fotografía están acostumbrados a que las figuras que retratan aparezcan algo distorsionadas. Más a menudo de lo que algunos quisieran, una línea, supuestamente recta, aparece curvada, a veces hacia el centro de la imagen y otras en sentido opuesto. Es lo que se conoce como distorsión de almohadilla (*pincushion*) o de barril (*barreling*), según el sentido de la curvatura.

Otras veces son los fotógrafos mismos quienes se aprovechan de este "defecto" y le sacan partido. Son los objetivos conocidos como "ojo de pez", lentes gran angular que acentúan esta imperfección y captan un mayor ángulo de realidad, aunque totalmente distorsionado, dándole a la fotografía un aspecto de burbuja, tal como se aprecia en la imagen 1.2.

Este tipo de distorsión se produce en cualquier lente, aunque existen formas de disminuirla o compensarla. Se presentará con mayor facilidad en las siguientes lentes:

Figura 1.1: Efecto barril y almohadilla (Imágenes obtenidas de lensdoc)

Lentes baratas Son lentes con más imperfecciones y que no utilizan métodos para compensarlas.

Lentes de distancia focal variable Las lentes de distancia focal fija están optimizadas para presentar menor distorsión para dicha distancia, pero cuando ésta varía la optimización de todo el rango focal se complica.

Lentes gran angular Debido a su propia naturaleza introducen bastante distorsión de barril que es difícil de compensar físicamente.

1.1.2. Representación matemática del problema

En la figura 1.3 se aprecia el contexto del problema. Se trabajará con dos planos: uno se corresponde con la proyección de la realidad antes de pasar por la lente y el otro con la obtenida después.

Se trabaja con coordenadas polares, situando el origen en el centro de la imagen (ver figura 1.4). Se toma un punto del plano de proyección $p_1 \equiv (d_1, w_1)$, siendo d_1 la distancia al centro y w_1 el ángulo que forma el vector con el eje de

Figura 1.2: Imagen ojo de pez

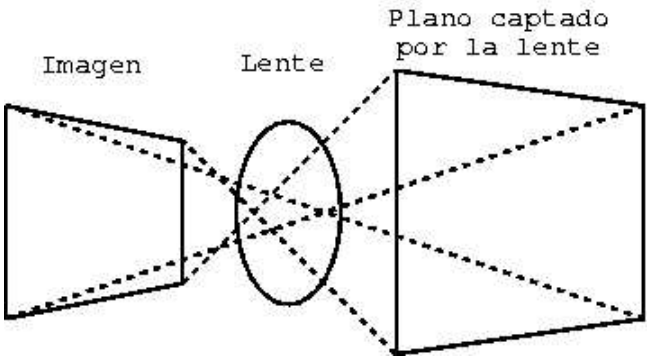


Figura 1.3: Planos de proyección

abscisas. La distorsión consiste en que dicha distancia d_1 se ve alterada. Esta variación depende exclusivamente de d_1; debido a la naturaleza simétrica de la lente, el ángulo w_1 no afecta a la transformación.

Por tanto se puede representar la transformación ejercida por una lente a través de una función $f : \mathbb{R} \to \mathbb{R}$, de modo que $f(d)$ devuelve la distancia al centro de la imagen a la que la lente sitúa un punto localizado, en la proyección, a una distancia d.

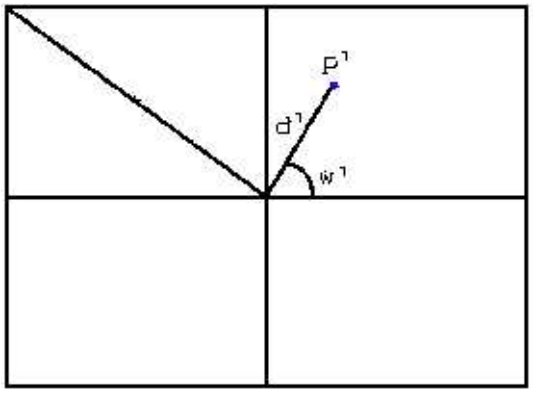


Figura 1.4: Coordenadas de un punto

Ya que las imágenes que se tengan que filtrar pueden tener distintas resoluciones, es necesario asegurar que la función es válida para todas ellas. Además, existe el problema de que las fotografías no tienen las mismas dimensiones. Por estas causas se decidió convertir la imagen en un cuadrado, y trabajar con distancias relativas. De esta forma se homogeniza el problema para cualquier tipo de imagen. El intervalo con el que se va a trabajar es el $[0, 1]$, correspondiéndose el 0 con el centro de la imagen y el 1 con la distancia máxima [1] que podemos obtener de una imagen.

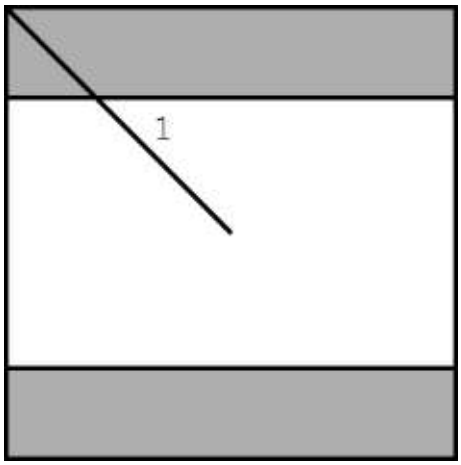


Figura 1.5: Convirtiendo un rectángulo en cuadrado

[1] Se corresponde con la esquina del cuadrado

Para convertir una imagen en un cuadrado es necesario escalar la imagen de forma que la dimensión máxima de la imagen coincida con el tamaño del cuadrado. Además se centrará la imagen en el cuadrado, tal como se aprecia en la figura 1.5.

Se necesitaría, pues, una función definida para el intervalo $[0, 1]$. Lamentablemente, el dominio de f no tiene por que ser éste, ya que una lente generalmente escala la proyección. En la figura 1.6 se aprecia como es una función f para una lente genérica que cometa la distorsión de barril.

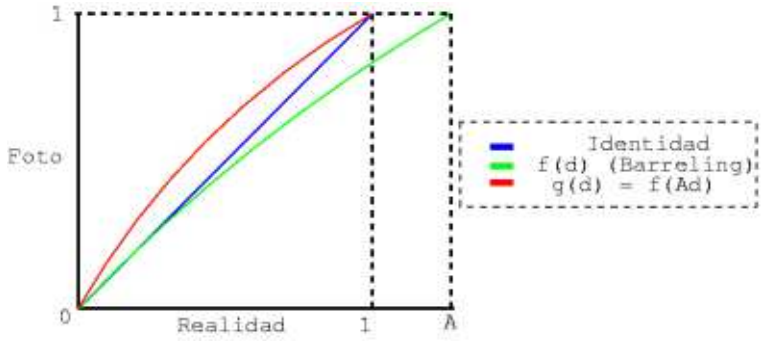


Figura 1.6: Funcion f(d) y $f(Ad)$

De este modo el dominio de f es $[0, A]$, siendo $A = f^{-1}(1)$, o lo que es lo mismo, la distancia máxima en la proyección. Entonces se construye $g : \mathbb{R} \to \mathbb{R}$, cuya definición es:

$$g(d) = f(Ad) \tag{1.1}$$

Contamos con los siguientes datos adicionales sobre la función g:

1. f debe ser continua, por la propia naturaleza de la lente, ya que la transformación no le impone cambios bruscos a la imagen, entonces g es también continua.

2. Una lente no introduce ningún desplazamiento, entonces:

$$f(0) = 0 \iff g(0) = 0 \tag{1.2}$$

3. Por la definición de g

$$g(1) = 1 \tag{1.3}$$

Una supuesta lente perfecta tendría una función f identidad. En las lentes que presenten estos defectos, f se alejará de la identidad a medida que la distancia con respecto al centro del plano de proyección aumente, como se puede apreciar en la figura 1.7.

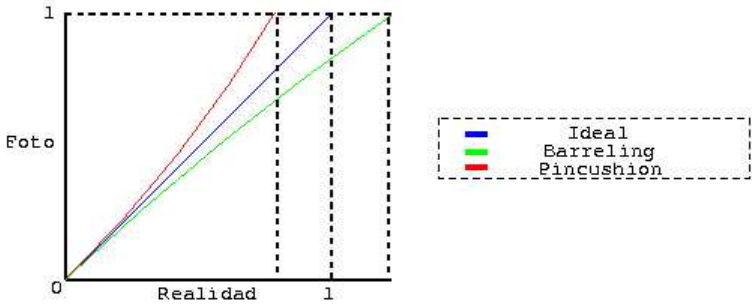


Figura 1.7: Tipos de función

Vistos cada tipo de deformación por separado, se tiene:

Efecto barril Figura 1.8. La lente hace que en la imagen aparezca más plano de proyección que el que captaría una función identidad, destacándose en las esquinas de la imagen, ya que allí la distancia al centro de la imagen es máxima. Al compactarse este área en el recinto rectangular de la imagen hace que las figuras aparezcan abombadas.

Efecto almhoadilla Figura 1.9. Aquí se produce la operación inversa. La función f de este tipo de lentes capta una menor superficie del plano de proyección que la función identidad, enfatizándose en los vértices. Entonces, al ajustarse al recuadro de la imagen es como si se *estirase* el área captada, produciéndose así, la apariencia de almohadilla.

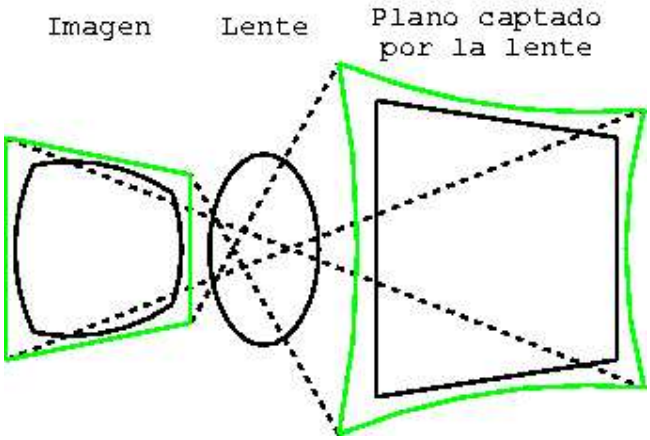


Figura 1.8: Plano efecto barril

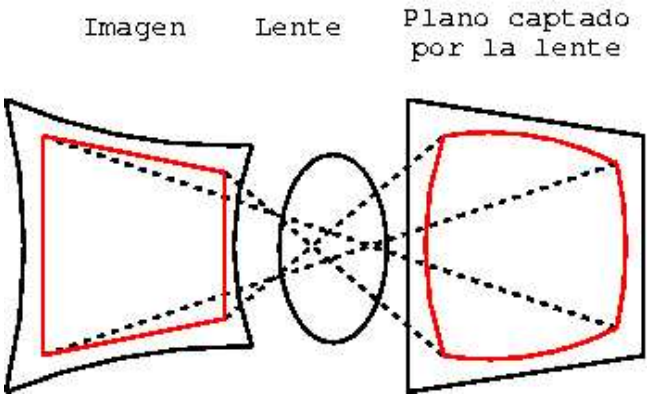


Figura 1.9: Plano efecto almohadilla

1.2. Estudio Empírico

1.2.1. El experimento

Una vez conocido el problema a afrontar, es necesario encontrar la manera de medir empíricamente los efectos de las lentes. Para ello se llevó a cabo el siguiente experimento[2]: Sobre un tablero de dimensiones 100x70 cm, se trazaron una serie de rectas verticales con una separación de 10cm y se situó un metro sobre la diagonal que va de la esquina superior izquierda a la esquina inferior derecha, tal

[2]Basado en el método utilizado por la aplicación LensDoc para describir un objetivo

como se ve en la figura 1.10.

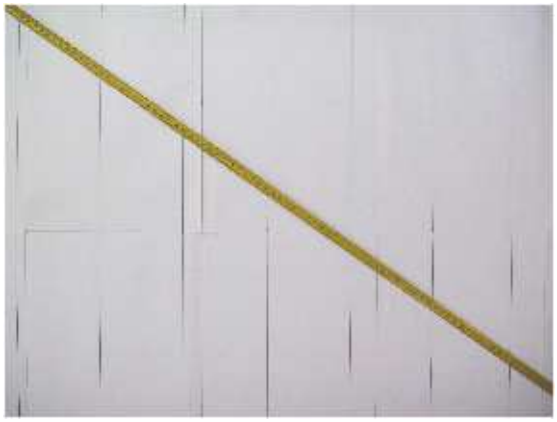


Figura 1.10: Experimento para medir el defecto de un objetivo

Esto sirvió como base para medir la distorsión introducida. Sacando una fotografía, de modo que el plano de la cámara se encontrase paralelo al tablero e intentando que las esquinas de la imagen coincidieran con las esquinas del tablero, se supone que 10 cm en el centro de la imagen deben medir lo mismo que 10 cm en la esquina de la misma. Esto no es correcto. Gracias al metro que atraviesa la imagen, se puede medir cuanto varía un punto con respecto al original.

Las líneas verticales sirven para "ver a ojo" si presenta el efecto barril o almohadilla y para comprobar si un supuesto filtro corrige bien la imagen, en caso de que trabaje mal, las líneas verticales no aparecerán rectas. La figura 1.11 muestra un ejemplo ficticio de una medición.

Al contrario de lo que puede parecer, la función obtenida como resultado no es el error de f, sino el error que comete la función g. Con lo cual, la función de la gráfica es la siguiente:

$$e(d) = g(d) - d \tag{1.4}$$

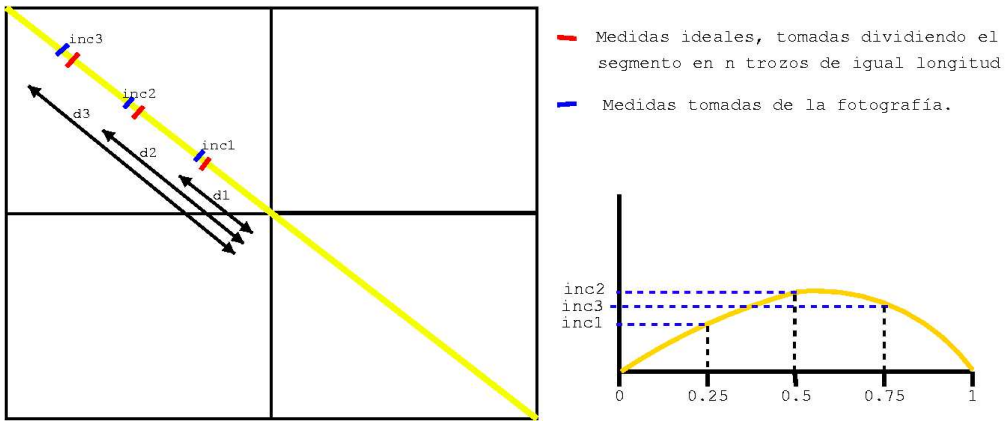


Figura 1.11: Ejemplo ficticio de una medición

1.2.2. Conclusiones

Con este sistema se realizaron múltiples pruebas utilizando una cámara fotográfica Canon EOS D30 y diversos objetivos de distancia focal variable. En este tipo de objetivos es muy complicado optimizar la lente para todo el rango focal, por ello, se tomaron mediciones para distintas distancias focal realizando un muestreo.

Una vez realizadas las pruebas, se comprobó que la distancia focal es el único factor que afecta a la función de error, tal como se había supuesto.

1.3. Estudio de mercado

En el mercado existen varios paquetes que corrigen este tipo de deformaciones:

LensDoc Plugin para Photoshop de pago con las siguientes características:

1. Corrige imagenes que presenten distorsiones de barril o almohadilla producidas por la mayoría de las lentes gran angular y zoom.
2. Identificando dos líneas que deberían ser rectas y poniendo sobre ellas unas guías, LensDoc corrige la imagen.
3. Corrige rotaciones y distorsiones de perspectiva.
4. Tiene una base de datos con curvas de correción para distintos objetivos fotográficos del mercado (además de algunos genéricos) y permite al usuario crear las suyas propias.

Panorama Tools Software para ver, crear y editar imagenes panorámicas. Tiene un plugin para Photoshop y Gimp que permite corregir distorsiones de barril o almohadilla introduciendo los coeficintes de una función polinómica; no tiene base de datos con curvas de corrección para distintos objetivos fotográficos ni permite interactuar con la imagen. Es software libre.

Capítulo 2

La solución

2.1. Ecuación

Es necesario encontrar la función $g : \mathbb{R} \to \mathbb{R}$ para poder construir un filtro que pueda corregir las deformaciones introducidas por una lente [1].

Se puede definir g como

$$g(d) = d + e(d)$$

Siendo la función $e : \mathbb{R} \to \mathbb{R}$ el error cometido por la lente. Por comodidad trabajaremos con la función de error e, siendo:

$$e(d) = g(d) - d \tag{2.1}$$

Esta función e se aproximará a partir de un polinomio de grado n.

$$e(d) = c_1 d^n + c_2 d^{n-1} + \ldots + c_{n-1} d^2 + c_n d \tag{2.2}$$

Por la ecuación 1.2 se puede eliminar el coeficiente c_{n+1}. Y gracias a la ecuación 1.3 se sabe que:

[1] Ver seccion 1.1.2

$$e(1) = 1 - 1 = 0$$

$$c_1 * 1 + \ldots + c_n * 1 = 0$$

$$\sum_{i=1}^{n} c_i = 0 \tag{2.3}$$

Siendo n el número de ecuaciones de que se dispongan (teniendo en cuenta la ecuación 2.3) y m el número de coeficientes que se quieran utilizar, se tiene que:

$$\begin{array}{ccccccccc} e_1 & = & c_1 d_1^m & + & c_2 d_1^{m-1} & + & \ldots & + & c_m d_1 \\ e_2 & = & c_1 d_2^m & + & c_2 d_2^{m-1} & + & \ldots & + & c_m d_2 \\ \vdots & & \vdots & & \vdots & & \ddots & & \vdots \\ e_n & = & c_1 d_n^m & + & c_2 d_n^{m-1} & + & \ldots & + & c_m d_n \end{array}$$

Esto se puede escribir de forma matricial:

$$\begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = \begin{pmatrix} d_1^m & \ldots & d_1 \\ \vdots & \ddots & \vdots \\ d_n^m & \ldots & d_n \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_m \end{pmatrix} \tag{2.4}$$

$$E = DC$$

Es necesario que $m \leq n$ para que el sistema tenga solución.

Si $m = n$ se puede resolver el sistema de la siguiente forma:

$$C = D^{-1}E \tag{2.5}$$

En el caso de que $m < n$ es necesario calcular un estimador de C para no despreciar ninguna ecuación que pueda contener datos valiosos. Por tanto, el sistema puede resolverse mediante el método cuadrático ordinario (ver bibliografía [4]).

$$C = (D'D)^{-1}D'E \tag{2.6}$$

2.2. Cálculo de e a partir de mediciones

Tomando mediciones con la aplicación de la sección 1.2.1 se obtienen una sucesión a con $n+1$ números, tales que $d_i \in [0,1]$ y $d_i < d_{i+1}$, para cualquier $i = 0..n$. Estos números deberían estar equiespaciados, pero en la práctica esto rara vez sucede. De esta forma muestran la transformación que realizó la lente a distintas distancias del centro para una determinada distancia focal. Se puede construir entonces una tabla con las mediciones tomadas, las mediciones ideales y el error cometido; siendo las mediciones ideales:

$$i_j = \frac{1}{n} j \text{ y } e_j = d_j - i_j, \text{ con } j = 0..n$$

Mediciones	Mediciones Ideales	error cometido
d_1	i_1	e_1
$\vdots$	$\vdots$	$\vdots$
d_n	i_n	e_n

Es necesario construir un sistema de matrices del tipo de la ecuación 2.4 para poder calcular los coeficientes de la función g.

$$\begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = \begin{pmatrix} i_1^m & \dots & i_1 \\ \vdots & \ddots & \vdots \\ i_n^m & \dots & i_n \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_m \end{pmatrix}$$

Según sean los valores de n y m se utilizará la ecuación 2.5 o 2.6 para poder calcular g.

En una lente de distancia focal variable se debe realizar el proceso anterior para muestrear todo el rango de la lente. De esta forma, tomando k muestras conseguimos k funciones g_i, con $i = 1..k$.

Distancia focal	Función
d_1	$g_1(d)$
d_2	$g_2(d)$
$\vdots$	$\vdots$
d_k	$g_k(d)$

Ahora, el problema es obtener la función asociada a una distancia focal cualquiera D, con $D \in [d_1, d_n]$. Para realizar esto es necesario "interpolar" funciones (ver apéndice A).

2.2.1. Cambio de escala

El uso de un objetivo fotográfico en distintas cámaras fotográficas puede dar como resultado distintas tranformaciones aunque el error cometido por el objetivo sea el mismo.

Esto se debe a que la superficie de la cámara que registra la imagen es variable, según el tipo y modelo de cámara (ver figura 2.1).

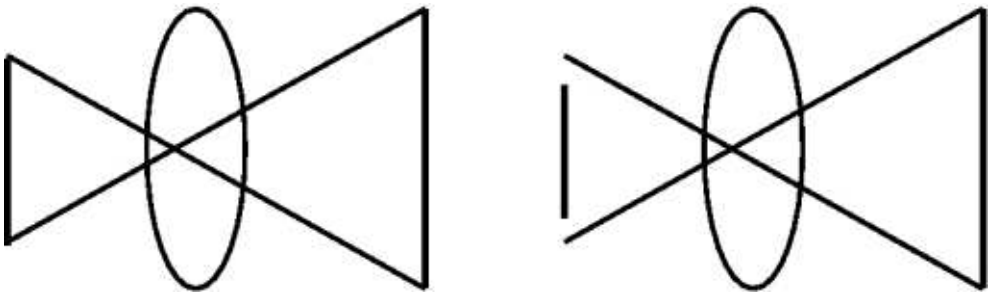


Figura 2.1: Supericie de registro de una imagen en distintas camaras

Siendo A la distancia máxima relativa que registra una determinada cámara y $B = g(A)$, se quiere obtener una función $g_n : \mathbb{R} \to \mathbb{R}$ que respete la forma de g y que cumpla:

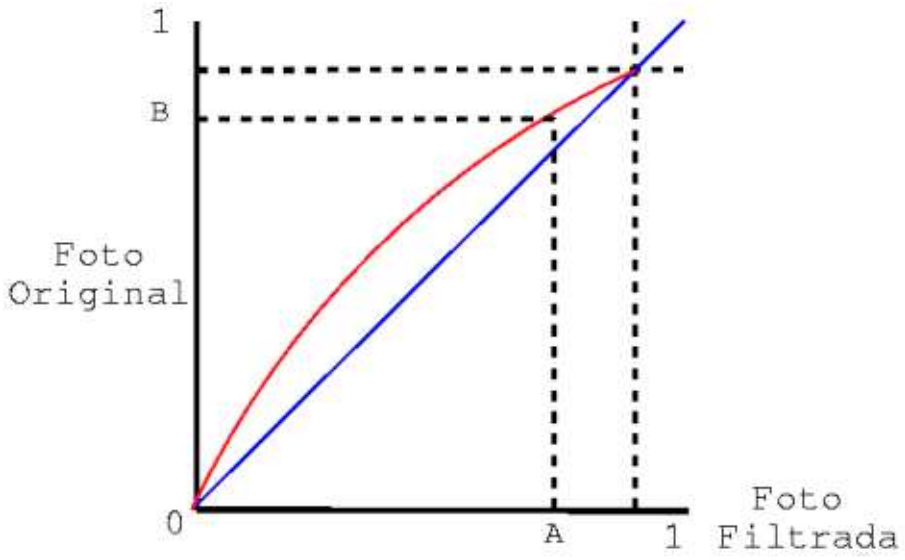


Figura 2.2: $g(x)$

$$g_n(1) = 1$$

De esta manera se deduce:

$$g_n(1) = 1 = \frac{B}{B} = \frac{g(A)}{B}$$

Entonces se define g_n como:

$$g_n(d) = \frac{g(Ad)}{B}$$

Calculando e_n:

$$e_n(d) = g_n(d) - d$$

$$e_n(d) = \frac{g_n(Ad)}{B} - d$$

$$e_n(d) = \frac{e(Ad) + Ad}{B} - d$$

2.3. Cálculo de e mediante rectas

Otra forma de obtener el error cometido por un objetivo fotográfico en una imagen es mediante rectas. Con respecto al método de las mediciones de la seccion 2.2 este sistema tiene la ventaja de no ser necesario saber con que objetivo se tomó la fotografía, y el inconveniente de no ser tan preciso.

Este método implica la existencia de un operador que marque 3 puntos sobre la imagen: $p_1 \equiv (x_1, y_1)$, $p_2 \equiv (x_2, y_2)$ y $p_3 \equiv (x_3, y_3)$; los cuales deberían estar en línea recta pero en la imagen describen una curva.

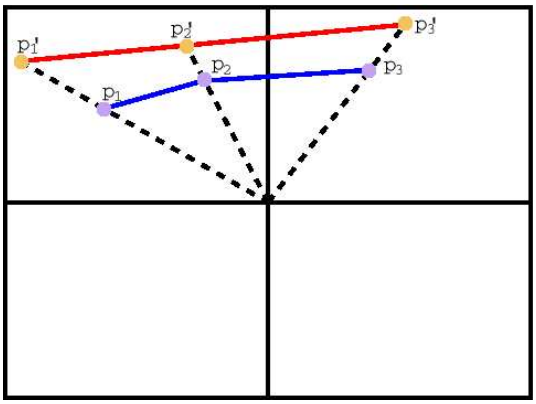


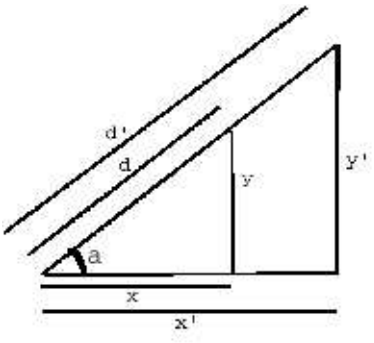
Figura 2.3: Rectas

Sabiendo que los puntos $p_1^{'} \equiv (x_1^{'}, y_1^{'})$, $p_2^{'} \equiv (x_2^{'}, y_2^{'})$ y $p_3^{'} \equiv (x_3^{'}, y_3^{'})$ son, respectivamente, las nuevas posiciones de los puntos anteriores, se puede calcular g^{-1} a partir de la siguiente ecuación:

$$\frac{y_3' - y_1'}{x_3' - x_1'} = \frac{y_2' - y_1'}{x_2' - x_1'} \tag{2.7}$$

Por otra parte se sabe que las nuevas posiciones de p_1', p_2' y p_3' están en la rectas que pasa por el centro de la imagen y por el correspondiente punto.

Utilizando triángulos tenemos que:



$$\text{sen}(a) = \frac{y}{d} = \frac{y'}{d'}$$
$$y' = y\frac{d'}{d} = yi$$
$$\cos(a) = \frac{x}{d} = \frac{x'}{d'}$$
$$x' = x\frac{d'}{d} = xi$$

Por lo cual:

$$\frac{y_3 i_3 - y_1 i_1}{x_3 i_3 - x_1 i_1} = \frac{y_2 i_2 - y_1 i_1}{x_2 i_2 - x_1 i_1} \tag{2.8}$$

$$i_2 i_3 (x_2 y_3 - x_3 y_2) + i_1 i_2 (x_1 y_2 - x_2 y_1) + i_1 i_3 (x_3 y_1 - x_1 y_3) = 0$$

Para simplificar la ecuación, se definen las constantes A_1, A_2 y A_3 de la siguiente forma:

$$\begin{aligned} A_1 &= (x_2 y_3 - x_3 y_2) \\ A_2 &= (x_3 y_1 - x_1 y_3) \\ A_3 &= (x_1 y_2 - x_2 y_1) \end{aligned}$$

Quedando la ecuación siguiente:

$$i_2 i_3 A_1 + i_1 i_2 A_3 + i_1 i_3 A_2 = 0 \tag{2.9}$$

Donde i_n es el factor de incremento de la distancia d_n.

$$i_n = \frac{g^{-1}(d_n)}{d_n} \tag{2.10}$$

2.3.1. Una recta

Como sólo hay una ecuación[2], como máximo g^{-1} se puede aproximar por un polinomio de grado 2 de la forma:

$$g^{-1}(d) = ad^2 + bd$$

Con la ecuación 1.3 se tiene que :

$$a + b = 1$$

$$b = 1 - a$$

Por tanto

$$g^{-1}(d) = ad^2 + (1 - a)d = a(d^2 - d) + d$$

Entonces, según la ecuación 2.10:

$$i_n = \frac{a(d_n^2 - d_n) + d_n}{d_n} = a(d_n - 1) + 1$$

Sustituyéndolo en la ecuacion 2.9:

$$\begin{aligned}(a^2(d_2 - 1)(d_3 - 1) + a(d_2 - 1) + a(d_3 - 1) + 1)A_1 + \\ (a^2(d_1 - 1)(d_2 - 1) + a(d_1 - 1) + a(d_2 - 1) + 1)A_2 + \\ (a^2(d_1 - 1)(d_3 - 1) + a(d_1 - 1) + a(d_3 - 1) + 1)A_3 = 0\end{aligned}$$

Reagrupando la ecuación :

[2] Además de la ecuación 2.3

$$(A_1(d_2-1)(d_3-1)+A_3(d_1-1)(d_2-1)+A_2(d_1-1)(d_3-1))a^2+$$
$$(A_1(d_2+d_3-2)+A_3(d_1+d_2-2)+A_2(d_1+d_3-2))a+$$
$$(A_1+A_2+A_3)=0$$

Ésta es una ecuación polinómica de grado 2 del tipo: $ax^2+bx+c=0$, la cual se puede resolver mediante:

$$x=\frac{-b+/-\sqrt{b^2-4ac}}{2a}$$

Donde

$$\begin{aligned} a &= A_1(d_2-1)(d_3-1)+A_3(d_1-1)(d_2-1)+A_2(d_1-1)(d_3-1) \\ b &= A_1(d_2+d_3-2)+A_3(d_1+d_2-2)+A_2(d_1+d_3-2) \\ c &= A_1+A_2+A_3 \end{aligned}$$

2.3.2. Dos rectas

Tenemos dos ecuaciones junto con la ecuación 2.3. Por tanto, se puede aproximar g^{-1} por un polinomio de grado 3 de la forma:

$$g^{-1}(d)=ad^3+bd^2+cd$$

Con la ecuación 1.3 se tiene que :

$$a+b+c=1$$

$$c=1-b-a$$

$$g^{-1}(d) = ad^3 + bd^2 + (1 - a - b)d$$

Entonces según la ecuación 2.10:

$$i_n = \frac{ad_n^3 + bd_n^2 + (1 - a - b)d_n}{d_n} = a(d_n^2 - 1) + b(d_n - 1) + 1$$

Sustituyéndolo en la ecuación 2.9:

$$\begin{aligned}
A_1[a^2(d_2^2-1)(d_3^2-1) + b^2(d_2-1)(d_3-1) + 1+ \\
a[(d_2^2-1) + (d_3^2-1)] + b[(d_2-1) + (d_3-1)]+ \\
ab[(d_2^2-1)(d_3-1) + (d_2-1)(d_3^2-1)]]+ \\
A_3[a^2(d_2^2-1)(d_1^2-1) + b^2(d_2-1)(d_1-1) + 1+ \\
a[(d_2^2-1) + (d_1^2-1)] + b[(d_2-1) + (d_1-1)]+ \\
ab[(d_2^2-1)(d_1-1) + (d_2-1)(d_1^2-1)]]+ \\
A_2[a^2(d_1^2-1)(d_3^2-1) + b^2(d_1-1)(d_3-1) + 1+ \\
a[(d_1^2-1) + (d_3^2-1)] + b[(d_1-1) + (d_3-1)]+ \\
ab[(d_1^2-1)(d_3-1) + (d_1-1)(d_3^2-1)]] = 0
\end{aligned}$$

Reagrupando

$$a^2[A_1(d_2^2-1)(d_3^2-1)+A_3(d_2^2-1)(d_1^2-1)+A_2(d_1^2-1)(d_3^2-1)]+$$
$$b^2[A_1(d_2-1)(d_3-1)+A_3(d_2-1)(d_1-1)+A_2(d_1-1)(d_3-1)]+$$
$$A_1+A_2+A_3+$$
$$a\{A_1[(d_2^2-1)+(d_3^2-1)]+A_3[(d_2^2-1)+(d_1^2-1)]+$$
$$A_2[(d_1^2-1)+(d_3^2-1)]\}+$$
$$b\{A_1[(d_2-1)+(d_3-1)]+A_3[(d_2-1)+(d_1-1)]+A_2[(d_1-1)+(d_3-1)]\}+$$
$$ab\{A_1[(d_2^2-1)(d_3-1)+(d_2-1)(d_3^2-1)]+$$
$$A_3[(d_2^2-1)(d_1-1)+(d_2-1)(d_1^2-1)]+$$
$$A_2[(d_1^2-1)(d_3-1)+(d_1-1)(d_3^2-1)]\}=0$$

Para simplificar la ecuación, se definen ahora las constantes k_1, k_2, k_3, k_4, k_5 y k_6 de la siguiente forma:

$$\begin{aligned}
k_1 &= A_1(d_2^2-1)(d_3^2-1)+A_3(d_2^2-1)(d_1^2-1)+\\
&\quad A_2(d_1^2-1)(d_3^2-1)\\
k_2 &= A_1[(d_2^2-1)+(d_3^2-1)]+A_3[(d_2^2-1)+(d_1^2-1)]+\\
&\quad A_2[(d_1^2-1)+(d_3^2-1)]\\
k_3 &= A_1[(d_2^2-1)(d_3-1)+(d_2-1)(d_3^2-1)]+A_3[(d_2^2-1)(d_1-1)+\\
&\quad (d_2-1)(d_1^2-1)]+A_2[(d_1^2-1)(d_3-1)+(d_1-1)(d_3^2-1)]\\
k_4 &= A_1[(d_2-1)+(d_3-1)]+A_3[(d_2-1)+(d_1-1)]+\\
&\quad A_2[(d_1-1)+(d_3-1)]\\
k_5 &= A_1(d_2-1)(d_3-1)+A_3(d_2-1)(d_1-1)+A_2(d_1-1)(d_3-1)\\
k_6 &= A_1+A_2+A_3
\end{aligned}$$

Como son dos rectas:

$$\begin{aligned} k_1a^2 + k_2a + k_3ab + k_4b + k_5b^2 + k_6 &= 0 \\ k'_1a^2 + k'_2a + k'_3ab + k'_4b + k'_5b^2 + k'_6 &= 0 \end{aligned}$$

Se tienen dos ecuaciones bicúbicas. La manera escogida para encontrar una solución es el método de Newton-Raphson aproximado.

2.3.3. Calculo de e

Una vez calculado g^{-1} es necesario calcular e para poder realizar el proceso de filtrado. Para ello se calcula g y, por definición:

$$e(d) = g(d) - d$$

El problema radica en calcular g.

Una posible solución sería muestrear g^{-1} y aplicar lo visto en la sección 2.2.

2.4. Filtrado

Se parte de una imagen de dimensiones (w, h) y de la función del error e cometido por el objetivo fotográfico.

Para poder pasar la distancia de un punto de la imagen al intervalo $[0, 1]$ es necesario calcular primero la distancia máxima de un punto de la imagen al centro de la misma.

$$d_{max} = maxd(p, (\frac{w}{2}, \frac{h}{2}));$$

Esta distancia se corresponde con la distancia a la que se encuentran las esquinas de la imagen.

Siendo $c \equiv (c_x, c_y)$ las coordenadas del centro de la imagen, donde

$$c_x = \frac{w}{2}$$

$$c_y = \frac{h}{2}$$

Entonces, las distancia máxima d_{max} es:

$$d_{max} = \sqrt{c_x^2 + c_y^2}$$

Para un punto de la imagen destino $p_d \equiv (x_d, y_d)$ se pretende saber que punto $p_o \equiv (x_o, y_o)$ de la imagen origen le corresponde.

El primer paso es calcular la distancia relativa del punto p_d al centro de la imagen:

$$d_d = \frac{\sqrt{(x_d - c_x)^2 + (y_d - c_y)^2}}{dmax}$$

Seguidamente se calculará la distancia relativa d_o que le corresponde a d_d tras pasar por el objetivo fotográfico:

$$d_o = g(d_d) = d_d + e(d_d)$$

Si se define el incremento de distancia como

$$i = \frac{d_o}{d_d}$$

Entonces

$$p_d = (c_x + (x_d - c_x)i, c_y + (y_d - c_y)i)$$

2.4.1. Escalado

Ya que la función e está definida de modo que $e(1) = 0$, cuando se intenta corregir el efecto barril pueden aparecer zonas en negro. En la figura 2.4 se corresponderían con el área encerrada entre las líneas azules y negras de la primera imagen.

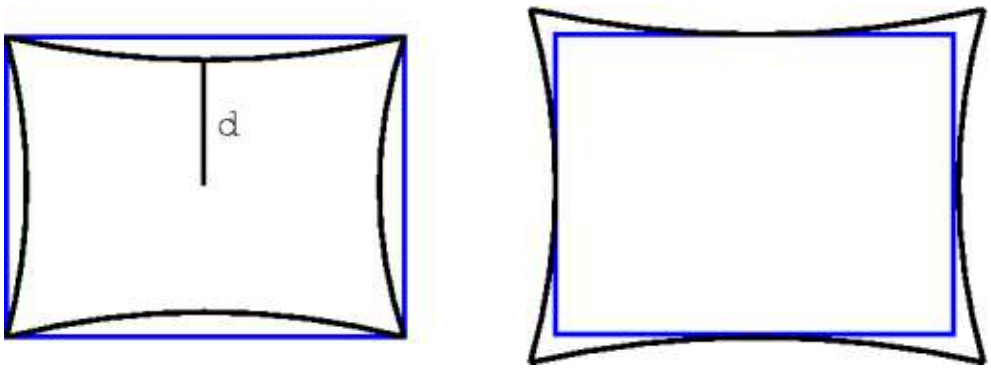


Figura 2.4: Defecto sobre Barreling

Por tanto, el primer paso para filtrar una imagen será escalar la función para

corregir este defecto, con lo cual se obtendrán unos resultados similares a la segunda imagen de la figura 2.4.

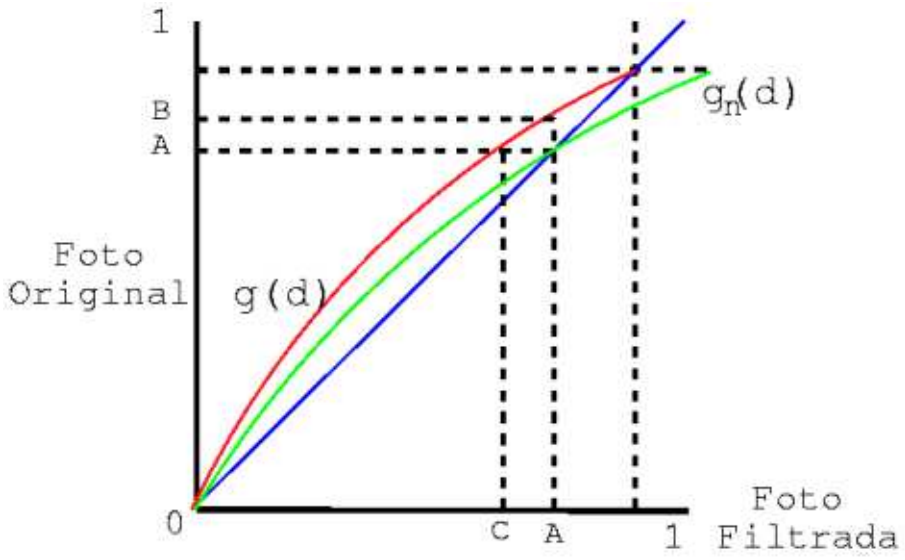


Figura 2.5: Cambio de escala

Siendo A la distancia al centro mas corta donde aparece el borde negro, tenemos:

$$g(A) = B$$

$$g(C) = A$$

Por tanto, es necesario utilizar una función g_n en vez de g que cumpla que:

$$g_n(A) = A$$

Entonces:

$$g_n(x) = g(\frac{C}{A}x)$$

Calculando la función e_n:

$$e_n(d) = g(x) - x$$

$$e_n(d) = g(\frac{C}{A}x) - x$$

$$e_n(d) = \frac{C}{A}x + e(\frac{C}{A}x) - x$$
$$e_n(d) = e(\frac{C}{A}x) + (\frac{C}{A} - 1)x$$

Parte II

Desarrollo

Capítulo 3

Metodología

Cuando se va a construir un sistema software no basta con conocer un lenguaje de programación. Si se quiere que el sistema sea robusto y mantenible es necesario que el problema sea analizado y la solución sea cuidadosamente diseñada. Se debe seguir un proceso robusto. Si se sigue un proceso de desarrollo que se planee cómo se realiza el análisis y el diseño, entonces la construcción de sistemas software va a poder ser planificable y repetible, y la probabilidad de obtener un sistema de mejor calidad al final del proceso aumenta considerablemente.

Para esta aplicación se va a seguir el método de desarrollo orientado a objetos que propone Craig Larman [1]. Este proceso no fija una metodología estricta, sino que define una serie de actividades que pueden realizarse en cada fase, las cuales deben adaptarse según las condiciones del proyecto que se esté llevando a cabo. La decisión de seguir este proceso radica en que pone en prácitca los últimos avances en Ingeniería del Software, y que adopta un enfoque eminentemente práctico, aportando soluciones a las principales dudas y/o problemas con los que se enfrenta el desarrollador. Su mayor aportación consiste en atar los cabos sueltos que anteriores métodos dejan.

La notación usada para los distintos modelos es la proporcionada por UML,

que ha llegado a ser el estándar de facto en cuanto a notación orientada a objetos.

Se va a abarcar todo el ciclo de vida, empezando por los requisitos y acabando en el sistema funcionando, proporcionando así una visión completa y coherente de la producción de sistemas software. El enfoque que toma es el de un ciclo de vida iterativo incremental, el cual permite una gran flexibilidad a la hora de adaptarlo a un proyecto y a un equipo de desarrollo específicos. El ciclo de vida está dirigido por casos de uso, es decir, por la funcionalidad que ofrece el sistema a los futuros usuarios del mismo. Así no se pierde de vista la motivación principal que debería estar en cualquier proceso de construcción de software: el resolver una necesidad del usuario/cliente.

3.1. ¿Que es la programación orientada a objetos?

La programación Orientada a Objetos (POO) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, procedimientos o métodos). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas. Esto difiere de los lenguajes procedurales tradicionales, en los que los datos y los procedimientos estan separados y sin relación. Estos métodos están ideados para conseguir programas y módulos más sencillos de escribir, mantener y reutilizar.

Explicado de otro modo, la programación orientada a objetos anima al programador a pensar en los programas principalmente en términos de tipos de datos, y, en segundo lugar, en las operaciones ("métodos") específicas a esos tipos de datos. Los lenguajes procedurales animan al programador a pensar, sobre todo, en términos de procedimientos, y, en segundo lugar, en los datos que esos procedimientos manejan.

Los programadores que emplean lenguajes procedurales, escriben funciones y después les pasan datos. Los programadores que emplean lenguajes orientados a objetos definen objetos con datos y métodos y después envian mensajes a los objetos diciendo que realicen esos métodos en sí mismos.

Hay un cierto desacuerdo sobre exactamente que características de un método de programación o lenguaje le califican como "orientado a objetos". En cambio hay un consenso general sombre las características más importantes:

Abstracción Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas

características. Los procesos, las funciones o los métodos pueden tambien ser abstraídos y cuando los están, una variedad de técnicas son requeridas para ampliar una abstracción.

Encapsulación También llamada "ocultación de la información". Se encarga de asegurar que los objetos no pueden cambiar el estado interno de otros objetos de maneras inesperadas; solamente los propios métodos internos del objeto pueden acceder a su estado. Cada tipo de objeto expone una interfaz a otros objetos que especifica cómo otros objetos pueden interactuar con él. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción.

Polimorfismo Las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del referente. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

Herencia Organiza y facilita el polimorfismo y la encapsulación permitiendo definir y crear objetos como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que reimplementar su comportamiento. Esto se suele hacer agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común.

Los conceptos de la programación orientada a objetos tienen su origen en Simula 67, un lenguaje diseñado para hacer simulaciones, creado por Ole-Johan Dahl y Kristen Nygaard del Centro de Cómputo Noruego en Oslo. Según se informa, la historia cuenta que trabajaban en simulaciones de naves, y fueron confundidos por la explosión combinatoria de cómo las diversas cualidades de

diversas naves podían afectar unas a las otras. La idea surgió para agrupar los diversos tipos de naves en distintas clases de objetos, siendo responsable cada clase de objetos de definir sus propios datos y comportamiento. Fueron refinados más tarde en Smalltalk, que fue desarrollado en Simula en Xerox PARC pero diseñado para ser un sistema completamente dinámico en el cual los objetos se podrían crear y modificar "en marcha" en lugar de tener un sistema basado en programas estáticos.

La programación orientada a objetos se posicionó como la metodología de programación dominante a mediados de los años ochenta, en gran parte debido a la influencia de C++ , una extensión del lenguaje de programación C. Su dominación se consolidó gracias al auge de las Interfaces gráficas de usuario, para los cuales la programación orientada a objetos está particularmente bien adaptada.

Las características de orientación a objetos fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo Ada, BASIC, Lisp, Pascal, y otros. La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas condujo, a menudo, a problemas de compatibilidad y a la capacidad de mantenimiento del código. Los lenguajes orientados a objetos "puros" por otra parte, carecían de las características. Para superar este obstáculo, se desarrollaron muchas tentativas para crear nuevos lenguajes basados en métodos orientados a objetos, pero permitiendo algunas características procedurales de maneras "seguras". El Eiffel de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos pero ahora ha sido esencialmente reemplazado por Java, en gran parte debido a la aparición de Internet, para la cual Java tiene características especiales.

Apenas como programación procedural conducida a los refinamientos de la técnica, tales como la programación estructurada, los métodos modernos de diseño de software orientado a objetos incluyen refinamientos como el uso de los patrones

de diseño, diseño por contrato, y lenguajes de modelado (tales como UML).

La programación procedimental clásica presenta ciertos problemas, que se han ido agravando poco a poco, a medida que se iban construyendo aplicaciones y sistemas informáticos más complejos, destacando los siguientes:

- Modelo mental anómalo. Nuestra imagen del mundo se apoya en los seres, a los que asignamos nombres sustantivos, mientras la programación clásica se basa en el comportamiento, representado usualmente por verbos.
- Es difícil modificar y extender los programas, pues suele haber datos compartidos por varios subprogramas, que introducen interacciones ocultas entre ellos.
- Es difícil mantener los programas. Casi todos los sistemas informáticos grandes presentan errores difíciles de localizar hasta que han pasado muchas horas de funcionamiento.
- Es difícil reutilizar los programas. Resulta prácticamente imposible aprovechar, en una aplicación nueva, las subrutinas que diseñadas para otra.

La programación orientada a objetos (OOP, por las siglas inglesas de Object-Oriented Programming) es una nueva forma de programar que proliferó a partir de los años ochenta y trata de encontrar solución a estos problemas utilizando los siguientes conceptos:

Objetos Entidades complejas provistas de datos (propiedades, atributos) y comportamiento (funcionalidad, programas, métodos).Corresponden a los objetos reales del mundo en el que nos encontramos.

Clases Conjuntos de objetos que comparten propiedades y comportamiento.

Herencia Las clases no están aisladas, sino relacionadas entre sí, formando una

jerarquía de clasificación. Los objetos heredan las propiedades y comportamientos de todas las clases a las que pertenecen.

Encapsulamiento Cada objeto se encuentra aislado del exterior, es un módulo natural, y la aplicación entera se reduce a un agregado o rompecabezas de objetos. El aislamiento protege los datos asociados a un objeto contra su modificación por quien no tenga derecho a acceder a ellos, eliminando efectos secundarios e interacciones.

Polimorfismo Programas diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, aunque el significado del programa varíe según el objeto al que se aplica.

La programación orientada a objetos introduce nuevos conceptos, aunque a veces solamente se trate de nuevos nombres aplicados a conceptos ya conocidos. Entre ellos destacan los siguientes:

Método: es un programa asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena mediante un "mensaje".

Mensaje: es una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros.

Propiedad, atributo o variable: datos asociados a un objeto o a una clase de objetos.

En la programación orientada a objetos pura deben utilizarse solamente mensajes, nunca llamadas a subrutinas.

Por ello, a veces recibe el nombre de programación sin CALL, igual que la programación estructurada se llama también programación sin GOTO.

Sin embargo, no todos los lenguajes orientados a objetos prohíben la instrucción CALL (o su equivalente), permitiendo realizar programación híbrida,

procedimental y orientada a objetos a la vez.

Entre los lenguajes orientados a objetos destacan los siguientes:

- Smalltalk
- Objective-C
- C++
- Ada 95
- Java
- Python
- Delphi
- Lexico (en castellano)
- C Sharp
- Eiffel
- Ruby
- ActionScript
- Visual Basic.NET
- PHP

3.2. Visión General de la metodología

El proceso a seguir para realizar desarrollo orientado a objetos es complejo, debido a la complejidad presente al intentar desarrollar cualquier sistema software de tamaño medio-alto. El proceso se compone de actividades y subactividades, aplicadas repetitivamente a distintos elementos.

Las tres fases al nivel más alto son las siguientes:

Planificación y Especificación de Requisitos Planificación, definición de requisitos, construcción de prototipos, etc.

Construcción La construcción del sistema. Dentro de esta estapa se diferencian las fases siguientes:

- **Análisis** Se analiza el problema a resolver desde la perspectiva de los usuarios y de las entidades externas que van a solicitar servicios al sistema.
- **Diseño** Especificación al detalle del sistema, describiendo cómo va a funcionar internamente para satisfacer lo especificado en el análisis.
- **Implementación** Se lleva lo especificado en el diseño a un lenguaje de programación.
- **Pruebas** Se llevan a cabo una serie de pruebas para corroborar que el software funciona correctamente y que satisface lo especificado en la etapa de Planificación y Especificación de Requisitos.

Instalación La puesta en marcha del sistema en el entorno previsto de uso.

De todas las fases, la que requiere más esfuerzo y tiempo en un proyecto de desarrollo es Construir. Para llevarla a cabo se adoptará un enfoque iterativo, tomando en cada iteración un subconjunto de los requisitos (agrupados según casos de uso) y llevándolo a través del análisis y diseño hasta la implementa-

ción y pruebas, tal y como se muestra en la Figura 3.1. El sistema va creciendo incrementalmente en cada ciclo.

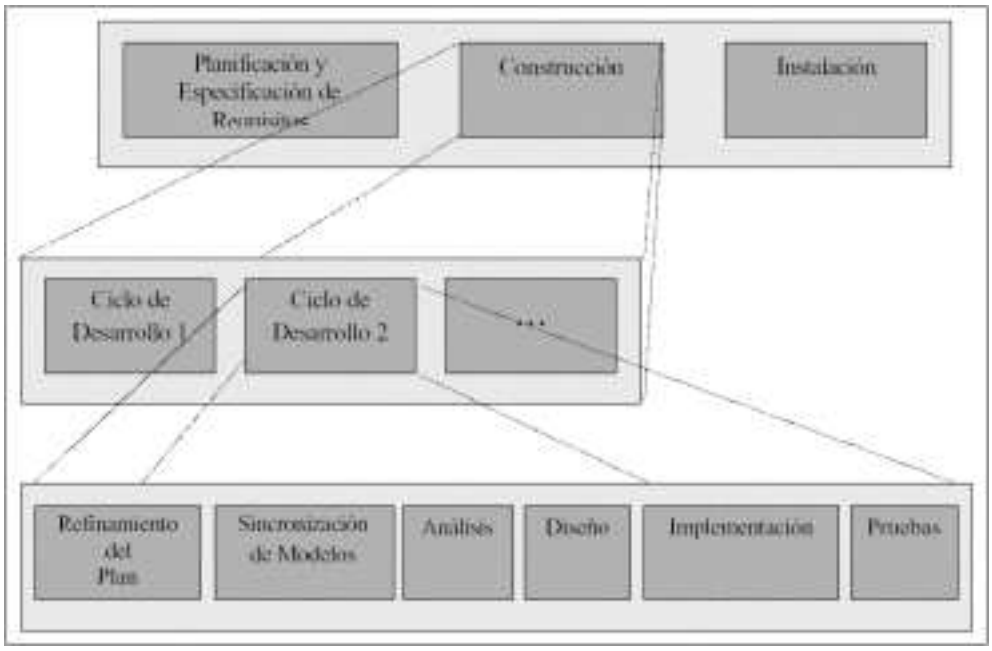


Figura 3.1: Fases de desarrollo

Con esta aproximación se consigue disminuir el grado de complejidad que se trata en cada ciclo, y se tiene pronto en el proceso una parte del sistema funcionando que se puede contrastar con el usuario/cliente.

Con esta aproximación se logra reducir la complejidad de cada ciclo y, en etapas tempranas de proyecto, se pueden obtener partes del sistema operativas, que se pueden contrastar con el usuario/cliente.

3.3. ¿Que es UML?

UML es una especificación de notación orientada a objetos. Se basa en las anteriores especificaciones BOOCH, RUMBAUGH y COAD-YOURDON. Divide cada proyecto en un número de diagramas que representan las diferentes vistas del proyecto. Estos diagramas juntos son los que representa la arquitectura del proyecto.

Con UML nos debemos olvidar del protagonismo excesivo que se le da al diagrama de clases. Éste representa una parte importante del sistema pero sólo es una vista estática, muestra al sistema parado. Sabemos su estructura pero no conocemos lo que le sucede a sus diferentes partes cuando el sistema empieza a funcionar. UML introduce nuevos diagramas que representan esa visión dinámica del sistema. Por todo esto, gracias al diseño de la parte dinámica del sistema podemos darnos cuenta en la fase de diseño de problemas de la estructura al propagar errores o de las partes que necesitan ser sincronizadas, así como del estado de cada una de las instancias en cada momento. El diagrama de clases continúa siendo muy importante, pero no se debe olvidar que su representación es limitada, y que ayuda a diseñar un sistema robusto con partes reutilizables, pero no a solucionar problemas de propagación de mensajes ni de sincronización o recuperación ante estados de error. En resumen, un sistema debe estar bien diseñado, pero también debe funcionar bien.

UML también intenta solucionar el problema de propiedad de código que se da con los desarrolladores. Al implementar un lenguaje de modelado común para todos los desarrollos se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo.

UML es ahora un standard, no existe otra especificación de diseño orientado a objetos, ya que es el resultado de las tres opciones existentes en el mercado. Su

utilización es independiente del lenguaje de programación y de las características de los proyectos, ya que UML ha sido diseñado para modelar cualquier tipo de proyectos, ya sean informáticos, de arquitectura o de cualquier otra rama.

UML permite la modificación de todos sus miembros mediante estereotipos y restricciones. Un estereotipo nos permite indicar especificaciones del lenguaje al que se refiere el diagrama de UML. Una restricción identifica un comportamiento forzado de una clase o relación, es decir, mediante la restricción se fuerza el comportamiento que debe tener el objeto al que se le aplica.

La explicación se basará en los diagramas, en lugar de en vistas o anotación, ya que son éstos la esencia de UML. Cada diagrama usa la anotación pertinente y la suma de estos diagramas crean las diferentes vistas. Las vistas existentes en UML son:

Vista casos de uso Se forma con los diagramas de casos de uso, colaboración, estados y actividades.

Vista de diseño Se forma con los diagramas de clases, objetos, colaboración, estados y actividades.

Vista de procesos Se forma con los diagramas de la vista de diseño. Recalcando las clases y objetos referentes a procesos.

Vista de implementación Se forma con los diagramas de componentes, colaboración, estados y actividades.

Vista de despliegue Se forma con los diagramas de despligue, interacción, estados y actividades.

Se dispone de dos tipos diferentes de diagramas: los que dan una visión estática del sistema y los que dan una visión dinámica. Los diagramas estáticos son:

Diagrama de clases Muestra las clases, interfaces, colaboraciones y sus relaciones. Son los más comunes y dan esa vista estática del proyecto.

Diagrama de objetos Se trata de un diagrama de instancias de las clases mostradas en el diagrama de clases. Muestra las instancias y sus interrelaciones dando una visión de casos reales.

Diagrama de componentes Muestra la organización de los componentes del sistema. Un componente se corresponde con una o varias clases, interfaces o colaboraciones.

Diagrama de despliegue Muestra los nodos y sus relaciones. Un nodo es un conjunto de componentes y se utiliza para reducir la complejidad de los diagramas de clases y componentes de un gran sistema. Sirve como resumen e índice.

Diagrama de casos de uso Muestra los casos de uso, actores y sus relaciones. Muestra quien puede hacer que y las relaciones existentes entre acciones(casos de uso). Son muy importantes para modelar y organizar el comportamiento del sistema.

Los diagramas dinámicos son:

Diagrama de secuencia, Diagrama de colaboración Muestra los diferentes objetos y las relaciones presentes entre ellos, además de los mensajes que se puedan enviar. Se trata de dos diagramas diferente que dan puntos de vista diferentes del sistema, pero permiten pasar perfectamente de uno a otro sin perdida de información. En resumen, cualquiera de los dos es un Diagrama de Interacción.

Diagrama de estados Muestra los estados, eventos, transiciones y actividades de los diferentes objetos. Son útiles en sistemas que reaccionen a eventos.

Diagrama de actividades Es un caso especial del diagrama de estados. Muestra el flujo entre los objetos. Se utilizan para modelar el funcionamiento del sistema y el flujo de control entre objetos.

A la vista está que número de diagramas es muy alto, en la mayoría de los casos incluso excesivos, y UML permite definir sólo los necesarios, ya que no todos son necesarios en todos los proyectos.

3.4. JAVA

Como lenguaje de programación para la realización de la aplicación se ha escogido Java. Los motivos de dicha elección han sido sus especiales características. A continuación se describen las mas importantes:

Lenguaje simple Java posee una curva de aprendizaje muy rápida. Resulta relativamente sencillo escribir aplicaciones interesantes desde el principio. Todos aquellos familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros. Debido a su semejanza con C y C++, y dado que la mayoría de la gente los conoce aunque sea de forma elemental, resulta muy fácil aprender Java. Los programadores experimentados en C++ pueden migrar muy rápidamente a Java y ser productivos en poco tiempo.

Orientado a objetos Java fue diseñado como un lenguaje orientado a objetos desde el principio. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos. La tendencia del futuro, a la que Java se suma, apunta hacia la programación orientada a objetos, especialmente en entornos cada vez más complejos y basados en red.

Distribuido Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Interpretado y compilado a la vez Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre

cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time).

Robusto Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo de los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria.

Seguro Dada la naturaleza distribuída de Java, donde las applets se bajan desde cualquier punto de la Red, la seguridad se impuso como una necesidad de vital importancia. A nadie le gustaría ejecutar en su ordenador programas con acceso total a su sistema, procedentes de fuentes desconocidas. Así que se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.

Indiferente a la arquitectura Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows Nt, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar requisitos de ejecución tan variopintos, el compilador de Java genera bytecodes: un formato intermedio indiferente a la arquitectura diseñado para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.

Portable La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas. Estas dos últimas características se conocen como la Máquina Virtual Java (JVM).

Alto rendimiento Hoy en día ya se ven como terriblemente limitadas las apli-

caciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (multithreading) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario, mientras otro presenta una animación en pantalla y otro realiza cálculos.

Dinámico El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

Capítulo 4

Planificación y Especificación de Requisitos

Esta fase se corresponde con la Especificación de Requisitos tradicional, aunque ampliada con un Borrador de Modelo Conceptual y con una definición de Casos de Uso de alto nivel. En esta fase se decidiría si se aborda la construcción del sistema mediante desarrollo orientado a objetos o no, por lo que, en principio, es independiente del paradigma empleado posteriormente.

4.1. Especificación de requisitos

Este proyecto consiste en la realización de una aplicación que cumpla los siguientes requisitos:

1. Debe tener una base de datos que almacene la función de error cometido por distintos objetivos fotográficos, de manera que se puedan corregir automáticamente las imágenes.

2. Debe permitir añadir nuevas funciones del error cometidas por distintos objetivos fotográficos a la base de datos.

3. Debe poder filtrar imágenes, corrigiendo los defectos añadidos por el objetivo fotográfico, de las siguientes formas:

   Automáticamente Según la base de datos de funciones de error cometidos por objetivos fotográficos, simplemente especificando con que objetivo fotográfico y con que distancia focal se tomó la imagen.

   Manualmente Indicando líneas que deberían ser rectas pero que en la fotografía aparecen curvadas.

4.2. Casos de uso de alto nivel

Inicialiciación

Actores Usuario

Descripción Al inicializar la aplicación deben cargarse la configuración y las preferencias del usuario.

Salir

Actores Usuario

Descripción Antes de salir la aplicación guarda las preferencias del usuario.

Gestionar imagenes

Actores Usuario

Descripción Un usuario puede abrir y guardar imágenes.

Filtrar imagen

Actores Usuario

Descripción El usuario le indica a la aplicación la función del error cometido en la toma de la fotografía, se realiza una previsualización y el usuario acepta los resultados.

Filtrar por función

Actores Usuario

Descripción El usuario introduce los coeficientes de la función de error cometido en la fotografía, lo que le permite a la aplicación corregir la imagen.

Filtrar por objetivo

Actores Usuario

Descripción El usuario especifica el modelo de objetivo y la distancia focal con los que se tomaron la fotografía y la aplicación calcula la función del error cometido en la imagen.

Filtrar por rectas

Actores Usuario

Descripción El usuario introduce una o dos rectas que aparecen curvadas en la imagen y la aplicación construye una función que describe el error cometido en la fotografía.

Gestión de objetivos

Actores Usuario

Descripción Un usuario puede dar altas, bajas y modificar la base de datos con descriptores de objetivos fotográficos.

4.3. Diagrama de Casos de uso

A partir de los requisitos especificados para la aplicación se obtienen los casos de uso de la figura 4.1.

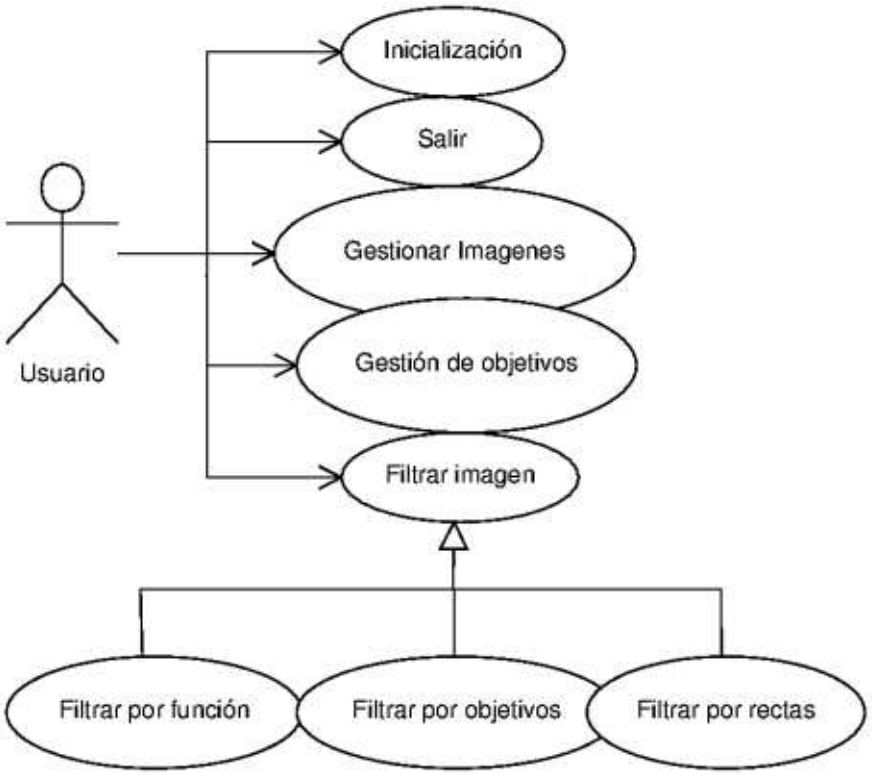


Figura 4.1: Casos de uso

4.4. Planificación de Casos de Uso según Ciclos de Desarrollo

Si se agrupan los casos de uso por funcionalidad se puede dividir el desarrollo en 3 fases o ciclos.

4.4.1. Primer ciclo

Se construirá la estructura común de la aplicación y el caso de uso Filtrar por función, lo que permitirá probar la estructura de la aplicación.

- Inicialización
- Salir
- Gestionar imágenes
- Filtrar imagen
- Filtrar por función

4.4.2. Segundo ciclo

Se desarrollará el caso de uso:

- Filtrar por rectas

4.4.3. Tercer ciclo

Se desarrollarán los dos casos de uso que están relacionados con la base de datos de objetivos.

- Filtrar por objetivos
- Gestión de objetivos

Capítulo 5

Primer Ciclo

En este ciclo se desarrollará la base de la aplicación, la cual comprende una plataforma básica que permita gestionar imágenes y realizar correcciones de imágenes fotográficas.

Además se diseñará e implementará una pequeña herramienta para corregir imágenes, que permita que el usuario introduzca directamente un polinomio que represente el error cometido en la toma de la fotografía, de manera que la aplicación pueda corregirla. Esta herramienta permitirá probar y validar la base de la aplicación.

5.1. Análisis

5.1.1. Casos de uso expandidos

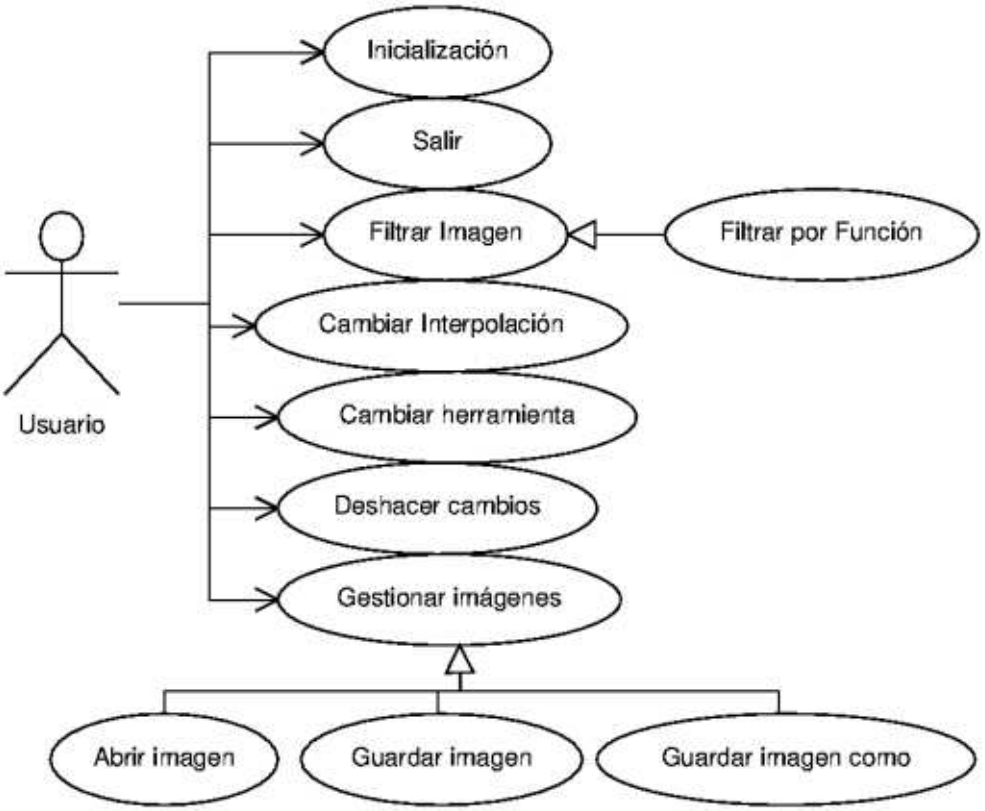


Figura 5.1: Casos de uso primer ciclo

Inicialización

Actores Usuario

Descripción Al inicializar la aplicación deben cargarse la configuración y las preferencias del usuario.

Precondiciones Debe estar fijada una variable de entorno indicando cual es el path de la aplicación.

Postcondiciones Aplicación inicializada

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Lanza la aplicación	2. Carga la configuración y las preferencias del usuario

Salir

Actores Usuario

Descripción El usuario indica que quiere salir de la aplicación y ésta, antes de salir, verifica que no existan cambios hechos sobre la imagen y no estén guardados y guarda las preferencias del usuario.

Precondiciones NA

Postcondiciones Preferencias guardadas

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Cierra la aplicación	2. Combrueba si hay cambios hechos en la imagen. 3. Se guardan las preferencias del usuario.

Cursos alternativos

Línea 2 Hay cambios no guardados. Le pregunta al usuario si desea guardar los cambios o cancelar la operación.

Abrir imagen

Actores Usuario

Descripción Un usuario pide abrir una imagen, el sistema verifica que no existan cambios no guardados en la edición de una imagen anterior y se la muestra.

Precondiciones NA

Postcondiciones Imagen mostrada

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Solicita abrir una imagen	2. Comprueba que no existan cambios.
3. Escoge la imagen a abrir	4. Muestra la imagen.

Cursos alternativos

Línea 2 Hay cambios no guardados. Le pregunta al usuario si desea guardar los cambios o cancelar la operación.

Guardar imagen

Actores Usuario

Descripción El usuario después de hacer correcciones en una imagen le indica a la aplicación que guarde la imagen, sobreescribiendo la existente.

Precondiciones Debe haber una imagen abierta y modificada.

Postcondiciones Cambios guardados.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1.Pide guardar la imagen	2. Guarda la imagen.

Guardar imagen como

Actores Usuario

Descripción El usuario le indica a la aplicación que guarde la imagen con otro nombre de archivo, pudiendo escoger el formato. Si el archivo ya existe pide confirmación.

Precondiciones Debe haber una imagen abierta.

Postcondiciones Imagen guardada.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Pide guardar como	2. Verifica que no exista el archivo. 3. Guarda la imagen.

Cursos alternativos

Línea 2 El fichero existe. Le pregunta al usuario si sobreescribe el archivo o cancela la operación.

Filtrar por función

Actores Usuario

Descripción El usuario le indica a la aplicación la función del error cometido en la toma de la fotografía introduciendo directamente los coeficientes de la función, se realiza una previsualización y el usuario acepta los resultados.

Precondiciones Hay una imagen abierta.

Postcondiciones Se muestra la imagen corregida.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Introduce la función de error	2. Realiza una previsualización
3. Acepta la previsualización	4. Filtra la imagen, guardando el estado anterior

Cursos alternativos

Línea 3 No acepta la previsualización. El sistema vuelve a mostrar la imagen sin filtrar.

Deshacer cambios

Actores Usuario

Descripción El usuario le indica a la aplicación que quiere deshacer los cambios hechos en la imagen. Los cambios se almacenarán en una pila y se podrán deshacer uno a uno.

Precondiciones Hay una imagen abierta y con cambios realizados.

Postcondiciones Cambios deshechos.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Indica que quiere deshacer los cambios hechos	2. Deshace los cambios

Cambiar interpolación

Caso de Uso

Actores Usuario

Descripción El usuario escoge el método de interpolación utilizado para filtrar una imagen.

Precondiciones NA

Postcondiciones NA

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Escoge la interpolación	2. La interpolación queda seleccionada.

Cambiar herramienta

Actores Usuario

Descripción El usuario escoge la herramienta que quiere utilizar con la aplicación.

Precondiciones NA

Postcondiciones NA

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Escoge una herramienta	2. Se muestra la interfaz de la herramienta.

5.1.2. Modelo conceptual

Se puede apreciar, según el diagrama 5.2, como los conceptos del dominio aparecidos en la análisis anterior son: Herramienta, Interpolación, Funcion, Imagen y Preview. Model es el elemento que engloba a todos los demás. Model tiene una lista de elementos Imagen, que guarda los sucesivos cambios sobre una imagen, de manera que puedan deshacerse las modificaciones.

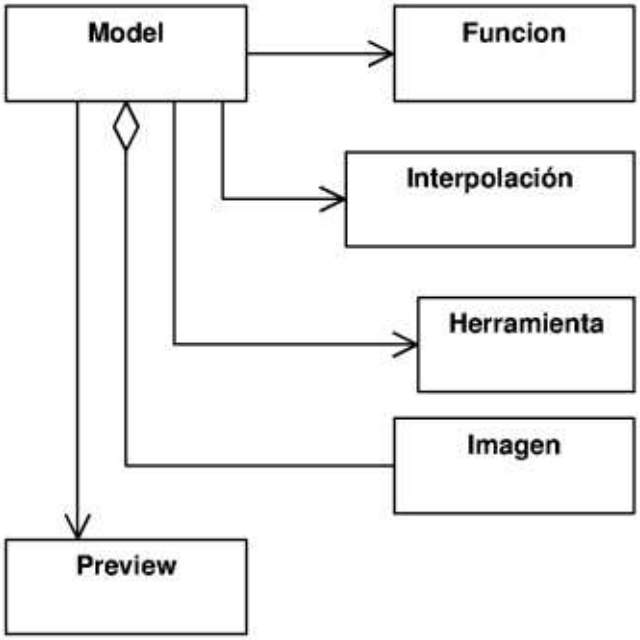


Figura 5.2: Modelo conceptual

5.2. Diseño

5.2.1. Componente

Con la finalidad de construir una aplicación que sea robusta y que se pueda modificar fácilmente, se opta por dividirla en componentes. Cada uno se encargará de una tarea distinta, intentando que exista el mínimo acoplamiento posible entre ellos. De esta forma se podría reemplazar un componente sin que el funcionamento del sistema se viese afectado. Ver figura 5.3.

Figura 5.3: Componente

En un mundo tan global como el de hoy en día no se pueden seguir insertando los mensajes en medio del código fuente. El hacer ésto limita el programa a un idioma, lo cual reduce el número de usuarios potenciales, reduciendo, de esta manera, los posibles beneficios. Por tanto, se debería añadir a todos los componentes la capacidad de internacionalización.

Una posible solución sería llevar para cada idioma un archivo con los mensajes indexados por una clave. Pero, ya que la aplicación se va a estructurar en distintos componentes, es más limpio que cada componente gestione sus propios mensajes.

De esta manera se puede diseñar la clase Componente, a la cual extenderán los componentes de la aplicación. Una instancia de esta clase cargará, en el cons-

tructor, los mensajes asociados a ese componente y tendrá métodos para acceder a ellos. Ver figura 5.4.

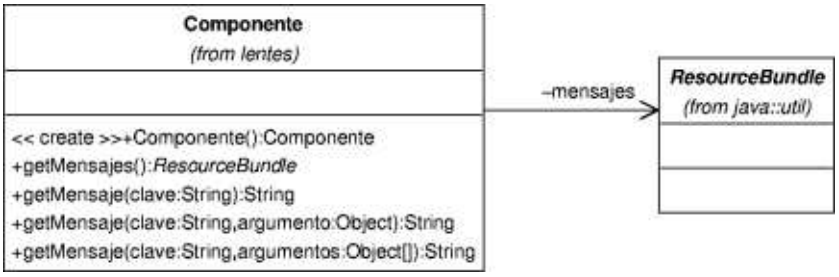


Figura 5.4: Componente refinado

Cuando se instancia esta clase se llamará al método getMensajes(), el cual devolverá un objeto ResourceBundle, que encapsula una familia de ficheros .properties, cada uno de ellos con los mensajes en un idioma.

Por defecto, el método getMensajes() estará implementado de manera que cargue el bundle cuyo nombre coincida con el nombre completo de la clase. En caso de que se desee cargar otro bundle, sólo es necesario sobreescribir este método.

Además, dispondrá de tres métodos para obtener los mensajes. Estos métodos utilizarán la clase java.text.MessageFormat para contruir los mensajes, sustituyendo las partes variables por su correspondiente argumento.

5.2.2. Capas

Contando ya con la aplicación estructurada en componentes, el siguiente paso lógico para construir un buen diseño sería dividir la aplicación en capas. De este modo se conseguiría una mayor facilidad de compresión y de mantenimiento de la aplicación.

Se ha optado por un diseño en dos capas, como muestra la figura 5.5.

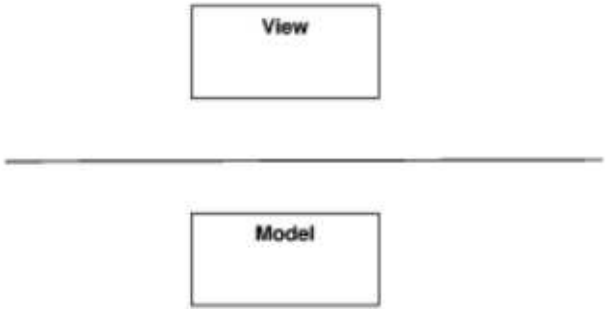


Figura 5.5: Capas

De esta forma se cuenta con las siguientes capas:

1. El modelo de la aplicación.
2. La interfaz de usuario.

Cada una de las capas será representado por un componente. Además, se va a diseñar el componente Lentes, que será el encargado de lanzar la aplicación, y se creará un tipo de componente que funcione como una especie de plugin sobre la aplicación y que permita aumentar su funcionalidad fácilmente. Este tipo de componente será conocido como Herramienta.

En la figura 5.6 aparecen detallados los diferentes componentes.

Si se refina un poco más la estructura de la aplicación, se puede definir la clase Widget, que es un elemento gráfico para colocar sobre la imagen a filtrar,

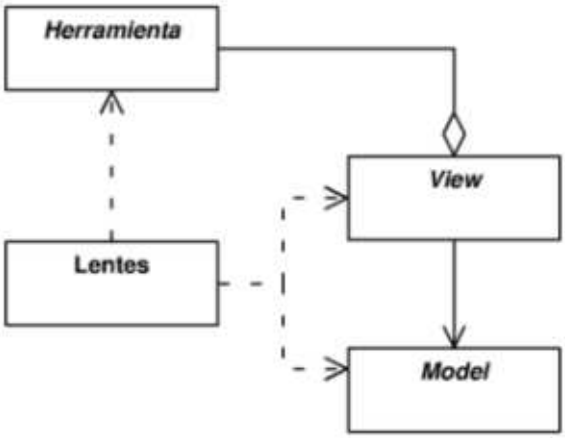


Figura 5.6: Estructura de la apicación

de modo que el usuario y la herramienta puedan interactuar.

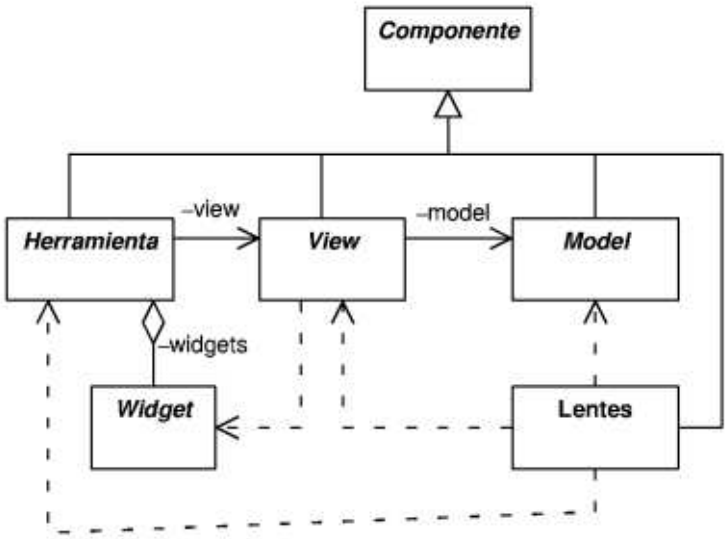


Figura 5.7: Estructura refinada

También se pueden definir las relaciones del componente Herramienta. La view tiene varias herramientas y una herramienta conoce directamente a la view e indirectamente al componente Model, a través de la view. Ver figura 5.7.

La clase Widget Un widget es un elemento gráfico que se muestra sobre una imagen y tiene como función que el usuario pueda interactuar con las herramientas.

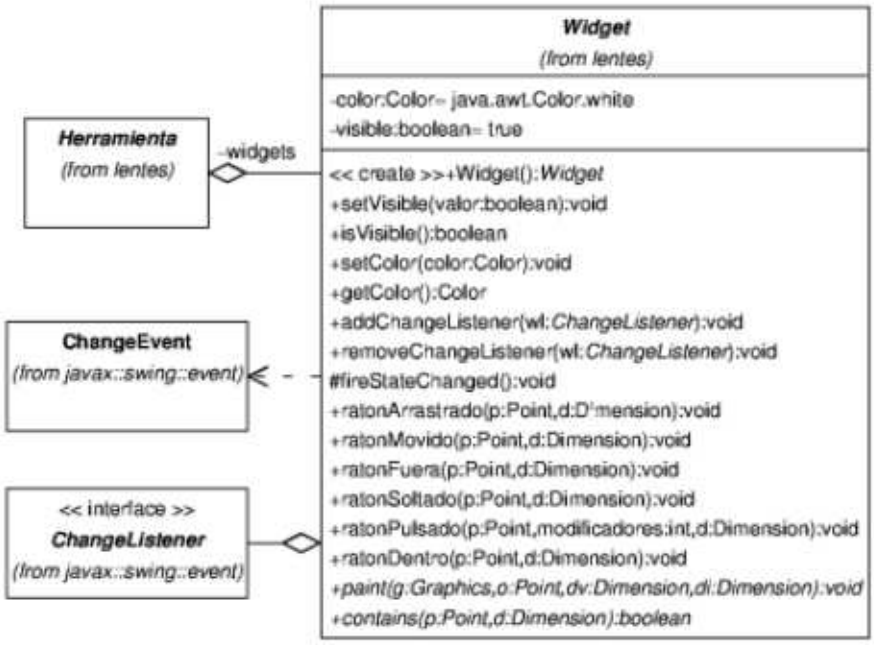


Figura 5.8: La clase Widget

Cuando una herramienta se activa colocará una serie de widgets sobre la imagen a filtrar, de manera que el usuario, a través del ratón, puede modificar los widgets. Éstos enviarán eventos, los cuales serán recogidos por la herramienta. De esta manera la herramienta puede obtener información de lo que ve el usuario. Ver figura 5.8.

5.2.3. Paquete lentes.funciones

Esta aplicación tiene una gran base matemática, por lo que se ha decidido diseñar un paquete especializado en esta área. En él se englobarán una serie de clases que simplifican y encapsulan el cálculo matématico para el resto de la aplicación.

El paquete se puede dividir en tres partes que se desarrollan a continuación:

Básico

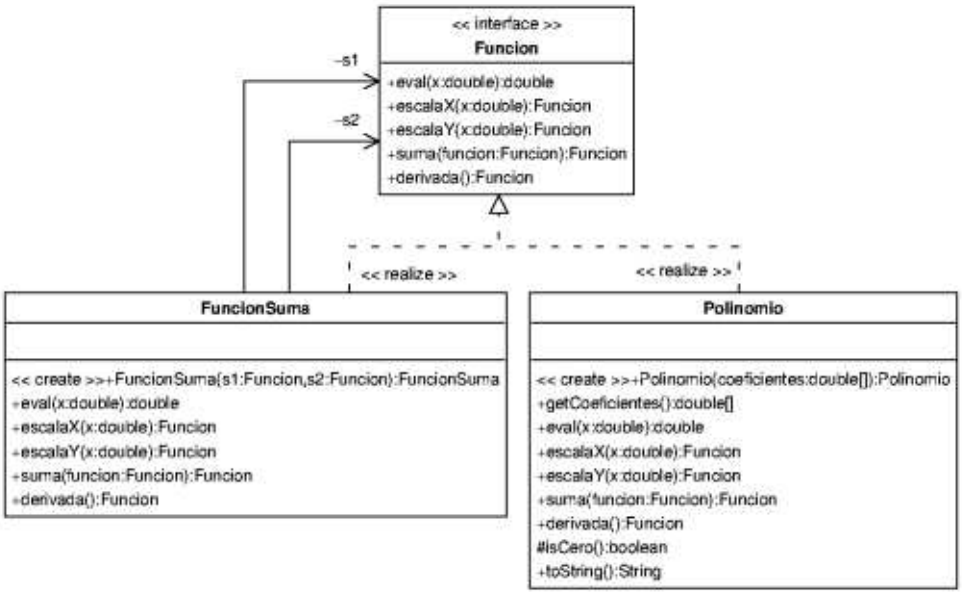


Figura 5.9: Paquete lentes.funciones

Interfaz Funcion Interfaz que representa una función matemática de una variable $f(x)$. Esta función se podrá evaluar en un punto. Las instancias de esta clase, una vez creadas, no son modificables.

Esta clase tiene métodos para poder obtener:

- La función suma de esta función más otra.

- La derivada.
- Una función g tal que g(x) = f(Ax)
- Una función g tal que g(x) = Af(x)

Además, tiene dos elementos estáticos que son instancias de este interfaz, se corresponden con la función CERO y con la función IDENTIDAD.

Clase FuncionSuma Implementación del interfaz Funcion. Representa una función suma de otras dos: $s(x) = f(x) + g(x)$

Clase Polinomio Implementación del interfaz Funcion. Representa una función polinómica del tipo $p(x) = c_0x^n + c_1x^{n-1} + ... + c_{n-1}x + c_n$.

Matrices

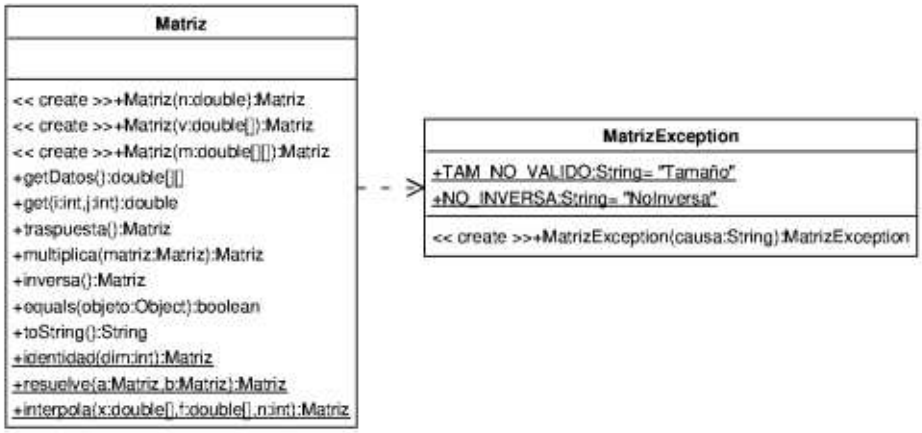


Figura 5.10: Paquete lentes.funciones

Clase Matriz Clase que representa una matriz de números reales. Las instancias de esta clase, una vez creadas, no son modificables. Esto es así con el fin de prevenir posibles errores.

Se crea a partir de un matriz de dimension mxn de coeficientes.

Esta clase cuenta con métodos para realizar las siguientes operaciones:

- Obtener la matriz traspuesta
- Obtener la matriz inversa
- Obtener la matriz resultado de multiplicarla por otra.
- Comprobar si es igual a otra matrices.

Además, tiene dos métodos estáticos, uno para obtener la matriz identidad pasándole como parámetro la dimensión, y otro para resolver ecuaciones del tipo $AX = B$. La solución a las ecuaciones se calculará como $X = A^{-1}B$ o como $X = (A'A)^{-1}A'B$ (en el caso de tener mayor número de ecuaciones que de incógnitas).

Clase MatrizException Excepción lanzada cuando se producen errores en el uso de matrices, tales como que una matriz no tenga inversa o que dos matrices no tengan dimensiones compatibles a la hora de multiplicarlas.

Métodos para resolver funciones

Clase Metodo Clase abstracta que representa un método iterativo para resolver funciones. Tiene las siguientes propiedades:

1. La función a resolver
2. El error máximo que debe cometer la solución
3. El número de iteraciones máximo para intentar resolver la función.

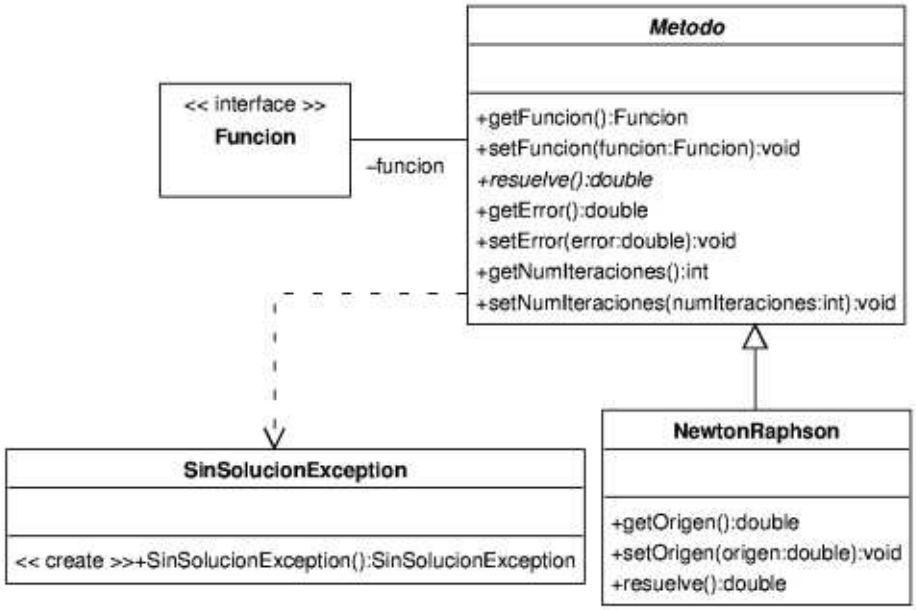


Figura 5.11: Paquete lentes.funciones

Clase NewtonRaphson Concretización de Metodo. Implementa el método de Newton-Raphson para encontrar ceros en funciones.

Añade la propiedad Origen, es el punto desde donde se desea comenzar a buscar la solución.

Clase SinSolucionException Excepción lanzada por un Método cuando se alcanzan el número máximo de iteraciones y no se consigue una solución válida.

5.2.4. El modelo

El modelo es un bean que guarda el estado de la aplicación y, además de métodos para obtener y modificar las propiedades, tiene métodos para realizar las operaciones de tratamiento de imágenes necesarias. Estos últimos métodos son abstractos, de modo que se pueden construir distintas implementaciones que utilicen sistemas distintos para realizar las operaciones.

El modelo no conoce la estructura de la aplicación ni la interfaz de usuario, pero lanza eventos cuando alguna de sus propiedades es modificada. Utiliza el patrón Listener. Ver figura 5.12.

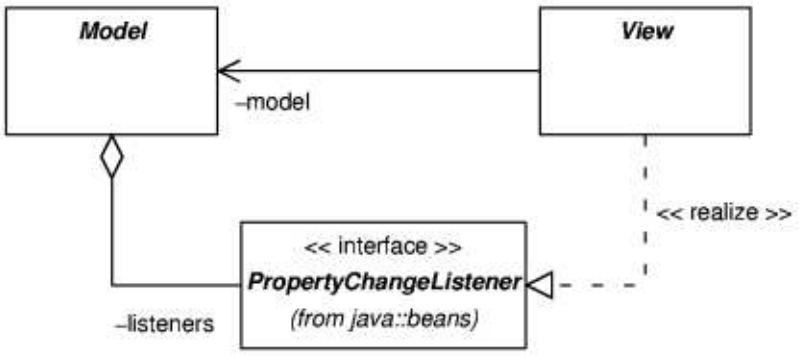


Figura 5.12:

Este bean tiene las siguientes propiedades:

- Nombre del fichero de la imagen utilizada.
- Nombre de la herramienta activa.
- Nombre de la interpolación activa.
- Función utilizada.
- La previsualización realizada.
- Si la imagen está modificada.

- Pila de imágenes. Tras cada cambio en una imagen se añade a la cima el resultado de la operación, de manera que en la cima está la imagen actual.

El diseño del modelo queda como se aprecia en la figura 5.13.

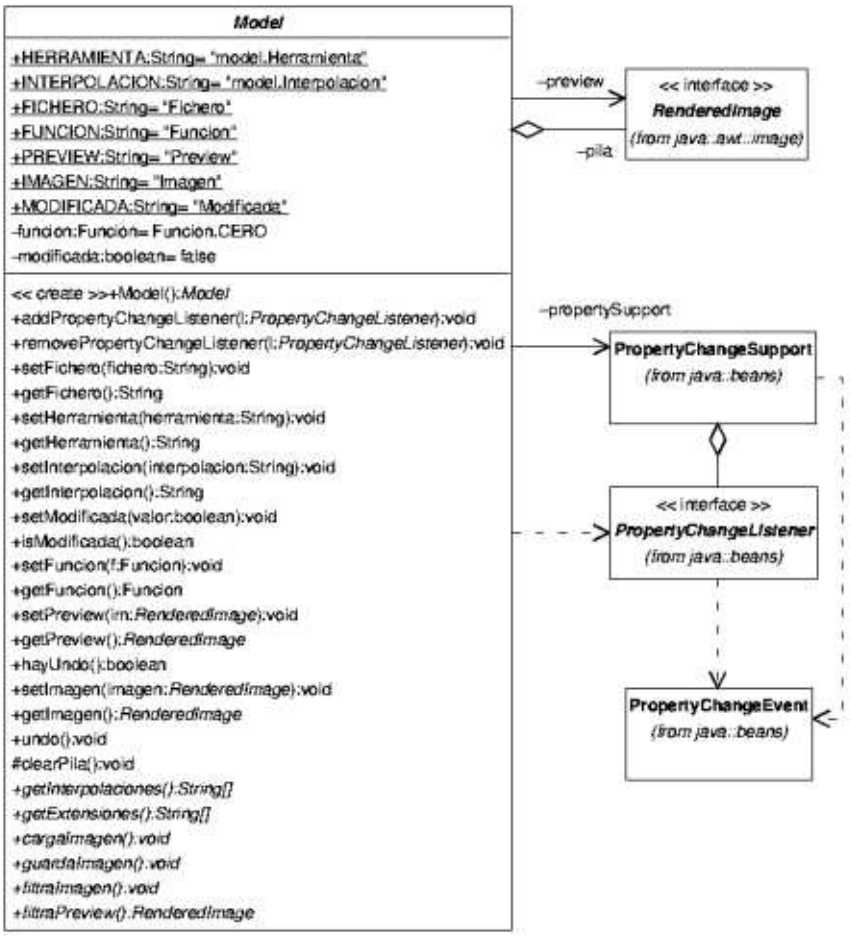


Figura 5.13: Componente model

Modos

La aplicación tiene dos modos de funcionamiento:

- Modo normal.
- Modo preview (cuando se realice una previsualización).

La propiedad preview clase Model indica en que modo se encuentra la aplicación. Si su valor es null indica que está en modo normal y si es distinto está en modo preview.

El paquete lentes.modeljai

En el paquete lentes.modeljai se encontrará una posible implementación del modelo de la aplicación. Esta implementación utilizará el API de Sun JAI para la manipulación de imágenes.

Concretamente, para filtrar una imagen utilizará la clase Warp. Esta clase representa una operación que, dadas las coordenadas de un punto de la imagen origen, devuelve las coordenadas del punto destino que le corresponde.

Esto se adapta perfectamente a la operación que se quiere conseguir, ya que el filtrado que se quiere realizar consiste simplemente en un cambio de disposición de los puntos de la imagen.

Para que la corrección sea válida y no se aprecien saltos en los puntos se utilizarán algoritmos de interpolación. El API de Sun JAI tiene definidos los siguientes métodos de interpolación:

1. Lineal
2. Bilinear
3. Bicúbica
4. Bicúbica2

Además, también se utilizarán las posibilidades de esta API en la carga y almacenamiento de imágenes. Los formatos de imágenes soportados son:

1. JPG

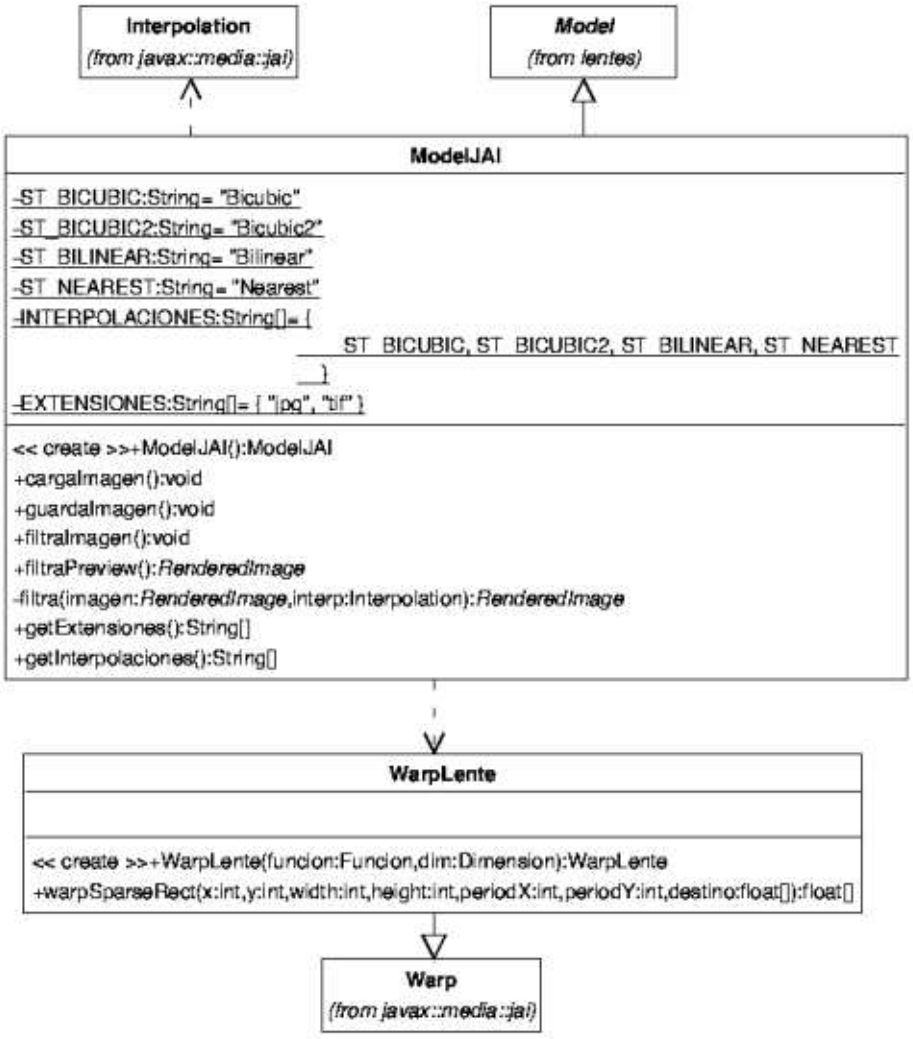


Figura 5.14: Paquete lentes.modeljai

2. TIFF

Clase ModelJAI Es la clase que extiende la clase model e implementa los métodos de carga y almacenamiento de imágenes y de filtrado. Para esto utiliza API de Sun JAI, como se ha dicho anteriormente. El método utilizado para el filtrado ha si explicado anteriormente en la sección 2.4.

Clase WarpLente Esta clase extiende a la clase Warp del API de Sun JAI. Se crea a partir de la función del error cometido por el objetivo fotográfico y las dimensiones de la imagen a filtrar. Esto es necesario porque la función está definida para el intervalo $[0, 1]$ y la imagen tiene otras dimensiones.

Realiza el escalamiento de la función de error explicado en la sección 2.4.1.

5.2.5. El interfaz de usuario

El interfaz de usuario es el componente que interacciona con el usuario, escuchando sus peticiones y mostrando los resultados que éste pide. Además conoce el modelo, lo modifica y recoge los eventos que lanza, actuando en consecuencia.

La clase View es la encargada de definir la estructura del interfaz de usuario. Esta clase es abstracta y extiende a Componente, de manera que tiene la gestión de mensajes garantizada.

Una implementación de esta clase será la encargada de crear el interfaz de usuario e interactuar con el mismo.

Clase View La estructura de la clase View se puede apreciar en la figura 5.15.

Esta clase tiene un método abstracto para obtener la ventana del interfaz de usuario. Además tiene otros dos métodos abstractos para mostrar mensajes y errores.

El paquete lentes.simpleview

En el paquete lentes.simpleview se implementará un interfaz de usuario simple para la aplicación Lentes. Este interfaz está dividido en varias superficies de trabajo, tal como se puede observar en la figura 5.16.

Las distintas superficies se muestran en la siguiente lista:

Título Aquí aparecen el nombre de la aplicación y el del fichero de la imagen que se está editando. Además si la imagen ha sido modificada y los cambios no han sido guardados aparecerá un asterisco.

Barra de Menú Contiene los siguientes menús:

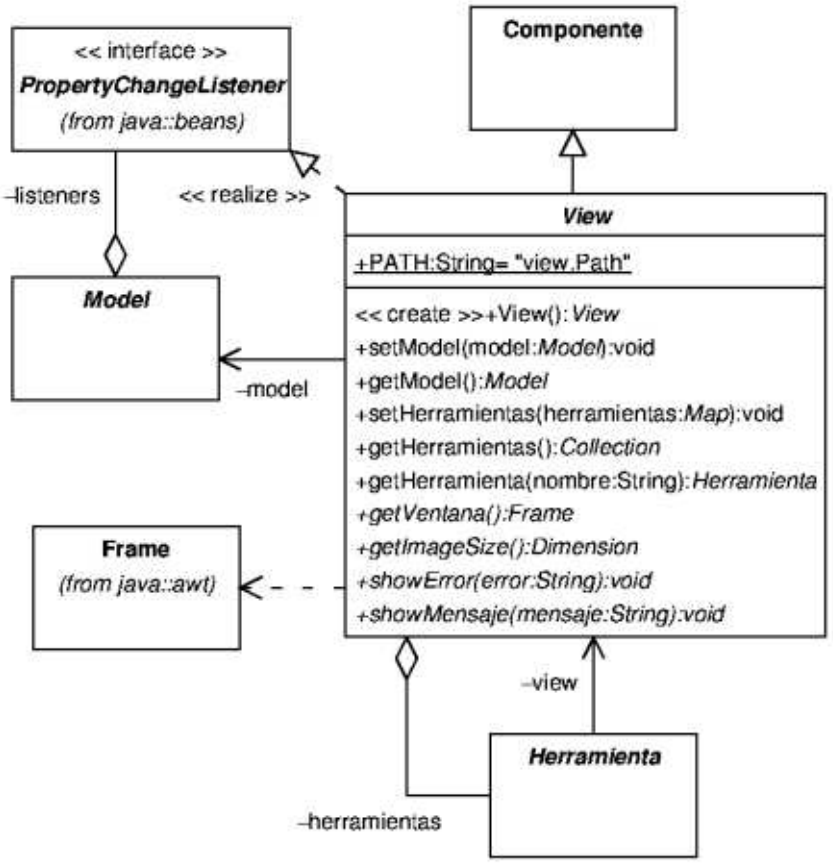


Figura 5.15: View

Archivo Con las opciones de abrir, guardar y salir.

Herramientas Con las distintas herramientas disponibles.

Interpolación Con los diferentes métodos disponibles de interpolación.

Ayuda Con la ayuda.

Visor de imagen En esta zona se aprecia una visión completa de la imagen. Cuando el interfaz está en modo preview se pueden ver los posibles resultados del filtrado. Y cuando está en modo normal se puede interaccionar con los elementos gráficos (widgets) de las herramientas.

Visor de detalle Cuando el interfaz está en modo normal se aprecia a escala real la zona del visor de imagen enmarcada por un recuadro verde.

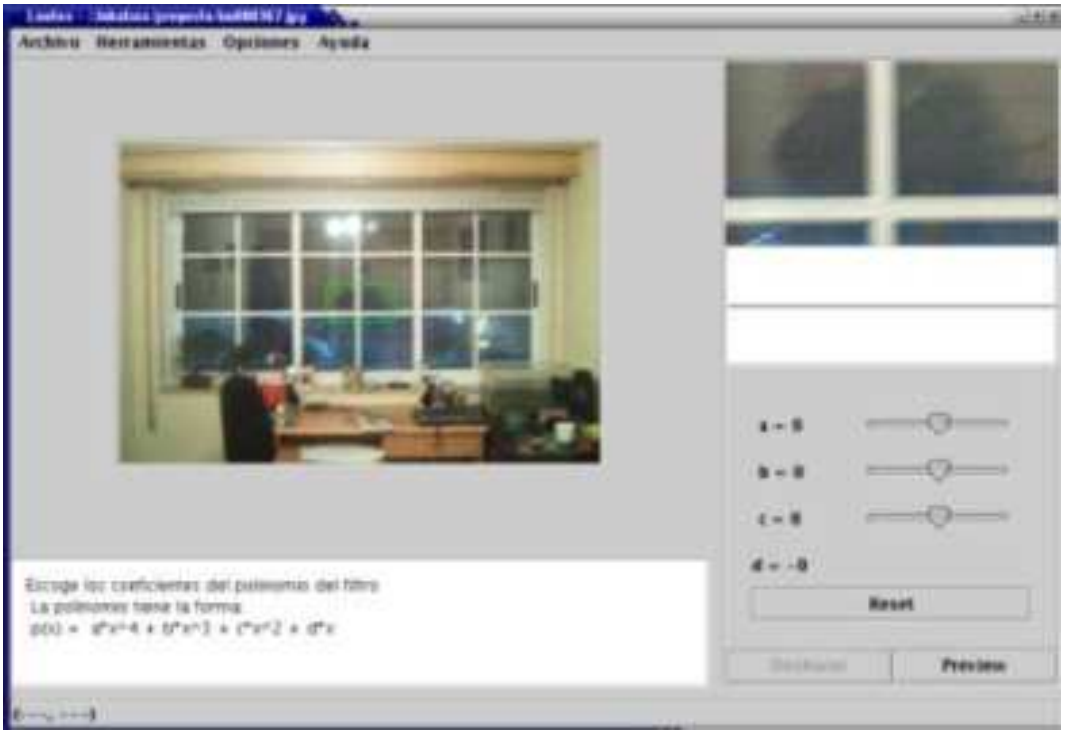


Figura 5.16: Interfaz de usuario

Gráfica Es la gráfica del error cometido por el objetivo.

Panel de Herramientas Emplazamiento del interfaz de la herramienta activa. Si la herramienta es un filtro se añadirán dos botones que cambian según el estado. En estado normal son:

Deshacer Deshace los cambios hechos anteriormente.

Preview Realiza una previsualización de la imagen corregida.

En estado preview son:

Cancelar Vuelve al estado normal.

Aceptar Acepta la previsualización y realiza la corrección.

Ayuda Panel donde las herramientas muestran información sobre como usarlas.

Barra de posición Cuando el ratón se encuentra sobre uno de los visores indica la posición en pixels.

El diagrama de clases del paquete lentes.simpleview se puede apreciar en la figura 5.17.

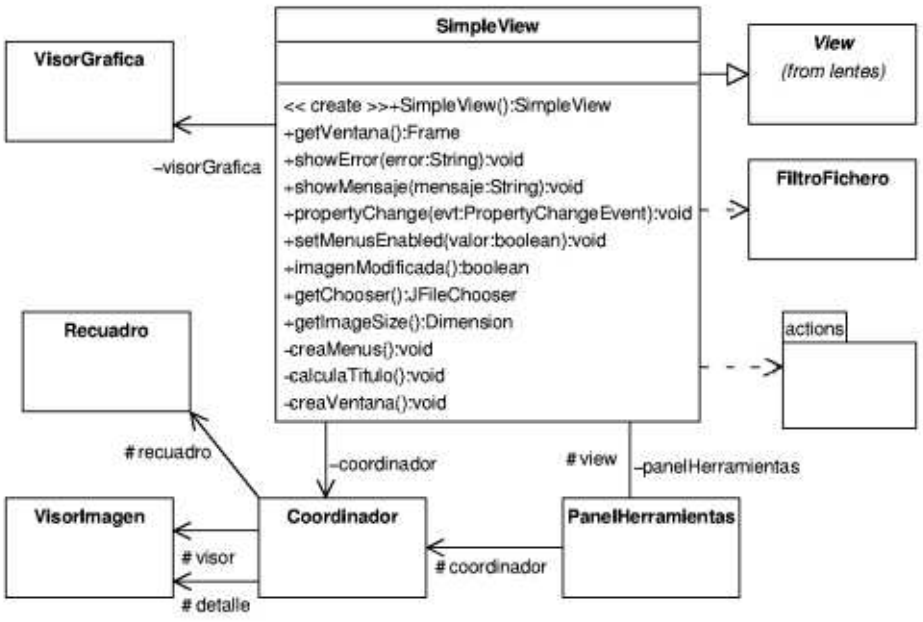


Figura 5.17: Paquete lentes.simpleview

Clase SimpleView Esta clase extiende e implementa a lentes.View. Crea la interfaz de usuario y sirve de medio de comunicación entre el usuario y el modelo. Para crear el interfaz gráfico se vale del API Swing desarrollado por Sun.

Clase VisorImagen Componente gráfico que extiende a javax.swing.JComponente. Tiene como función mostrar una imagen y permtir que el usuario interactúe, a través del ratón, con una serie de herramientas, llamadas widgets, que se mostrarán sobre la imagen. De esta manera algún objeto puede obtener información sobre lo que el usuario ve en la imagen.

Para ver una definición detallada ver la figura 5.18.

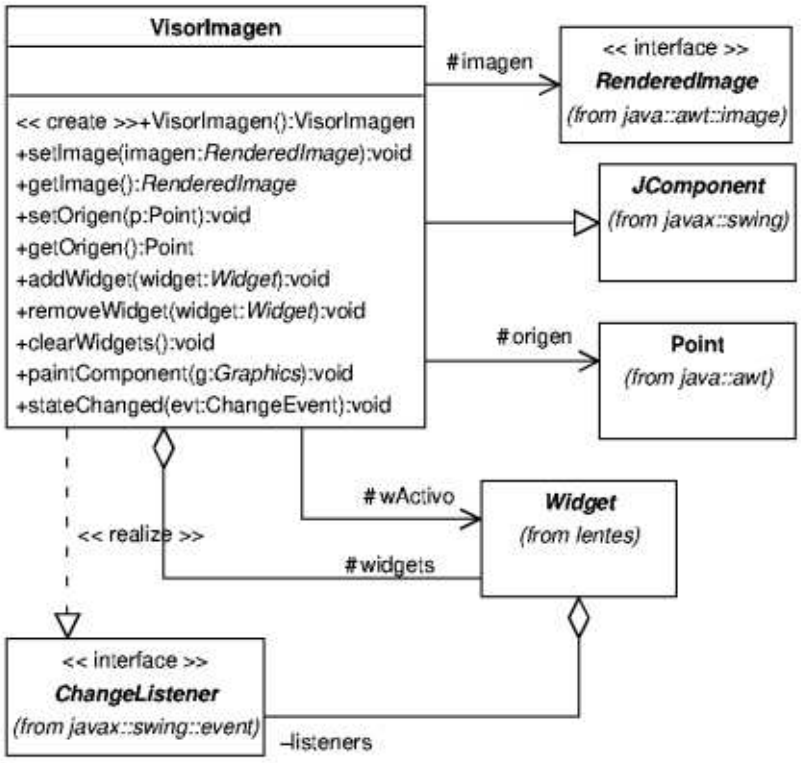


Figura 5.18: Clase VisorImagen

Clase VisorGrafica Componente gráfico que extiende a javax.swing.JComponente. En el se mostrará el intervalo [0, 1] de una función $f(x)$.

Esta clase tiene una serie de propiedades que configurán el estilo de la gráfica.

- El color del fondo
- El color del eje
- El color de la función
- La escala utilizada para pintar la función.

Ver figura 5.19.

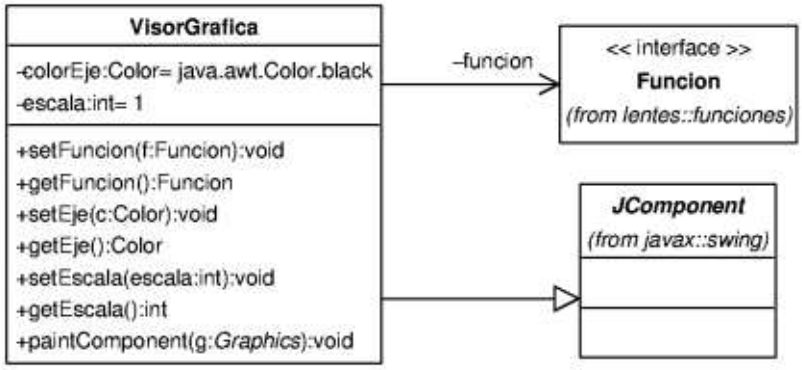


Figura 5.19: Clase VisorGrafica

Clase Coordinador Componente que extiende a javax.swing.JPanel. Es un panel con un objeto VisorImagen, el cual muestra una imagen escalada ajustada al tamaño del componente.

Esta clase tiene un método para obtener otro VisorImagen que muestra una parte de la imagen sin escalar, haciendo de efecto zoom. Esta clase coordina las iteraciones entre la visión global de la imagen y la visión del detalle. Para realizar esta tarea al visor global se le añade el widget Recuadro, el cual delimita el área de la imagen que se verá en el visor de detalle.

Le añade el widget Recuadro a la visión global de la imagen, delimitando el área de imagen que se verá en la visión de detalle.

Se puede ver el diagrama de clases en la figura 5.20.

Clase Recuadro Esta clase extiende e implementa a la clase abstracta lentes.Widget. Su función es dibujar un recuadro sobre la imagen mostrada. La posición y las dimensiones de este recuadro se pueden variar a través de distintos métodos, pero esta clase no reaccionará directamente a los eventos de ratón lanzados por el usuario.

La clase coordinador lo utiliza para marcar sobre el visor global el área que se

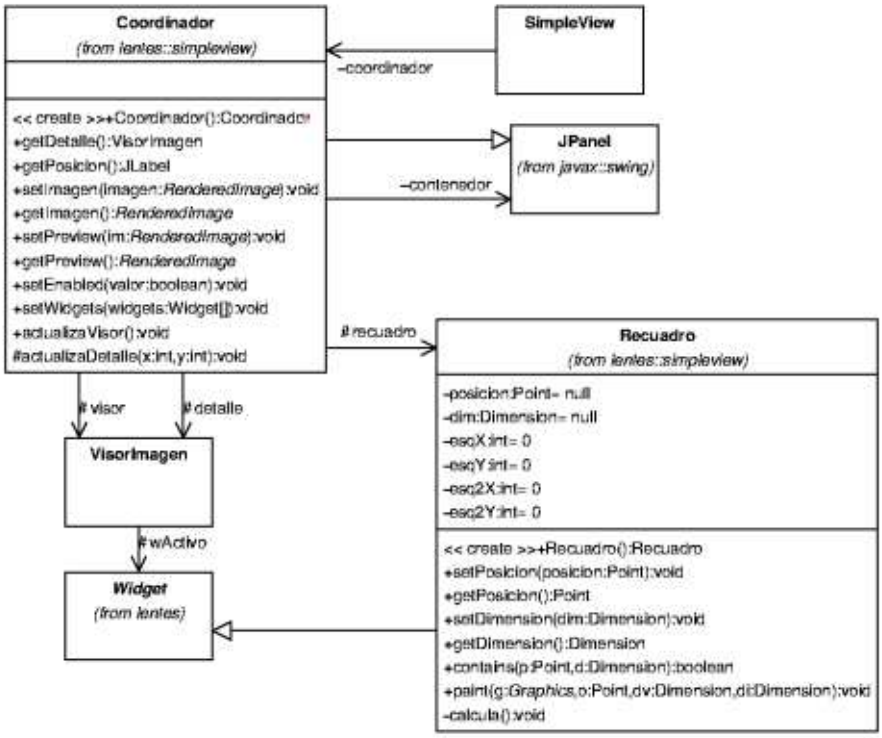


Figura 5.20: Clase Coordinador

muestra en el visor de detalle.

Clase FiltroFichero Esta clase representa un filtro de ficheros. Para decidir que archivos seleccionar se basa en su extensión.

Tiene una propiedad llamada extensiones, que le indica que extensiones de ficheros debe seleccionar. Además siempre cumple las dos reglas siguientes:

1. Seleccionará los directorios.

2. No seleccionará archivos los ocultos.

Si la propiedad extensiones está vacia entonces seleccionará los archivos con cualquier extensión.

Se puede utilizar con un objeto javax.swing.JChooseFile o para sacar un listado de un directorio con un objeto java.io.File.

Ver figura 5.21.

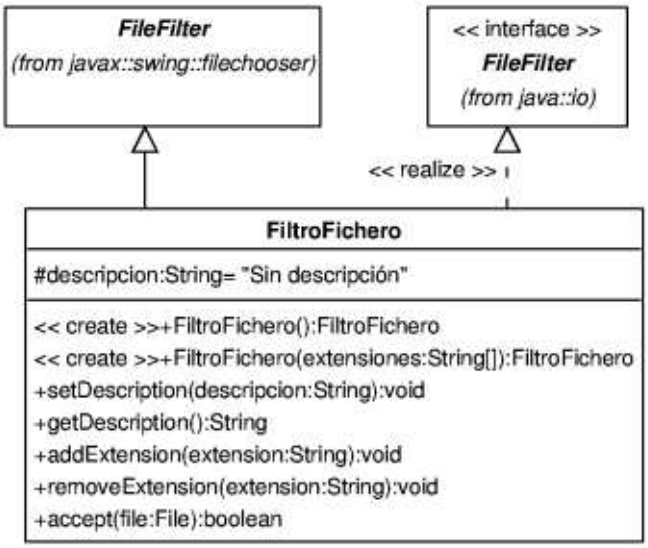


Figura 5.21: Clase FiltroFichero

.

Clase PanelHerramienta Componente donde se muestra el interfaz de usuario de la herramienta activa.

Si la herramienta es un filtro aparecen dos botones que tendrán diferente función dependiendo del modo en el que se encuentre la aplicación.

En modo normal se pueden llevar a cabo las siguientes acciones:

Filtrar Le indica a la aplicación que realice una previsualización.

Deshacer Deshace un cambio hecho previamente.

Y en modo preview:

Aceptar Indica que acepta la previsualización y realiza el filtrado en la imagen completa.

Cancelar Cancela la previsualización.

El diseño de esta clase se puede apreciar en la figura 5.22.

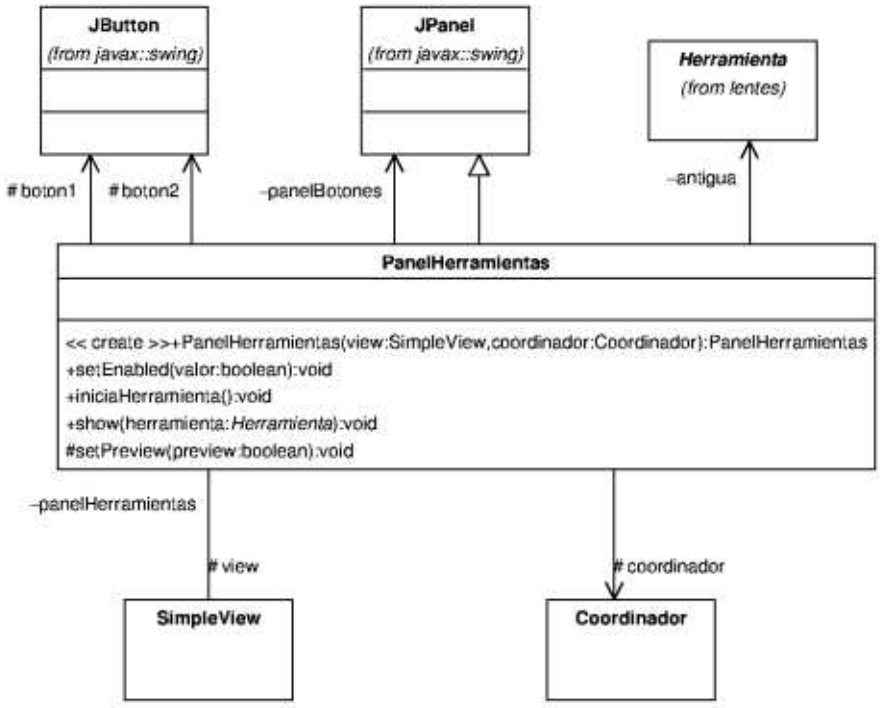


Figura 5.22: Clase PanelHerramienta

El paquete lentes.simpleview.actions

El paquete lentes.simpleview.actions contiene las acciones utilizadas por SimpleView. Ver figura 5.23.

Clase AboutAction Acción que lanza un panel de About sobre la aplicación.

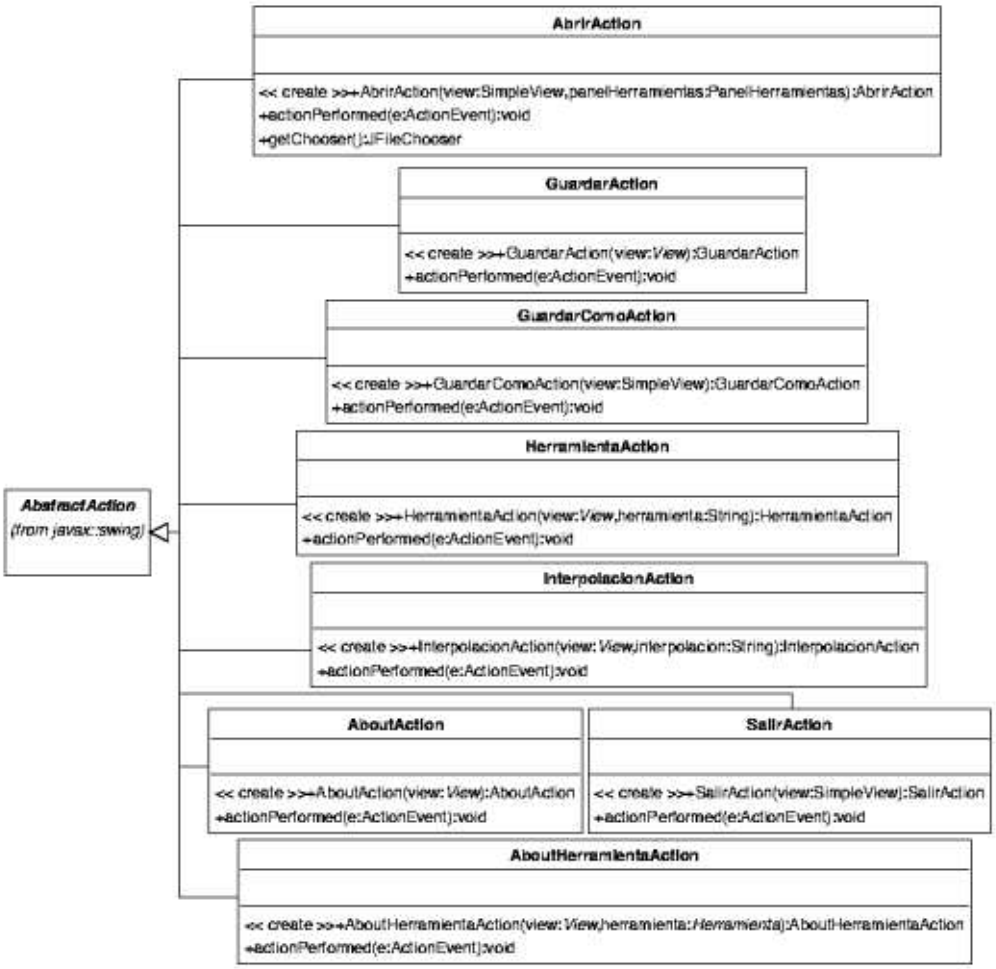


Figura 5.23: Paquete lentes.simpleview.actions

.

Clase AboutHerramientaAction Acción que lanza un panel de About para una herramienta.

Clase AbrirAction Acción que abre una ventana donde seleccionar una imagen, que será abierta por la aplicación.

Clase GuardarAction Acción que guarda una imagen.

Clase GuardarComoAction Acción que abre un ventana donde introducir el nombre de un archivo donde se guardará una imagen.

Clase HerramientaAction Acción que carga una herramienta en la aplicación.

Clase InterpolacionAction Acción que carga un método de interpolación en la aplicación.

Clase SalirAction Acción que hace que salga de la aplicación.

5.2.6. Componente Lentes

Este componente es el lanzador de la aplicación. Utiliza el patrón Singleton, de manera que se puede acceder a el desde cualquier punto de la aplicación, y extiende a Componente.

Debe cargar el archivo de configuración, de donde obtiene, entre otros, la implementación del modelo y la del interfaz que debe cargar. Este archivo tiene las siguientes propiedades básicas:

lentes.ModelClass La clase del componente que implementa el modelo de la aplicación.

lentes.ViewClass La clase del componente que implementa el interfaz de usuario de la aplicación.

model.Herramienta La herramienta cargada por defecto.

model.Interpolacion El método de interpolación por defecto.

view.Path El path en donde el interfaz de usuario busca las imágenes por defecto.

Además cada componente puede añadir sus propias propiedades.

Las propiedades por defecto se guardarán al salir de la aplicación con los valores que haya en ese momento.

Además también carga las herramientas disponibles. Para realizar esta tarea utiliza la clase Plugins.

La clase Plugins Esta clase será la encargada de obtener las herramientas disponibles.

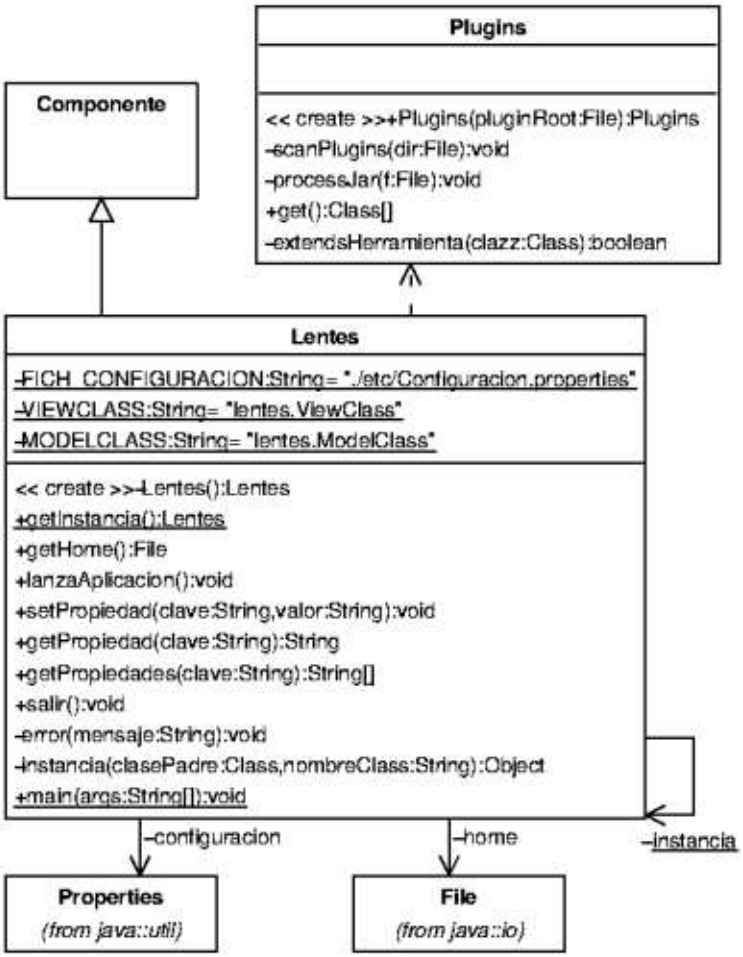


Figura 5.24: Lentes

Para hacer esto recorre los archivos .jar del directorio plugins buscando clases cuyo nombre finalice con Plugin y extiendan a la clase Herramienta.

5.2.7. Herramientas

Para poder aumentar la funcionalidad de la aplicación sin tener que rehacerla se le proporciona la habilidad de utilizar plugins. Una herramienta es un plugin.

Existen dos tipos de herramientas:

1. Las que tienen como finalidad poder calcular la función de error cometido en una fotografía.
2. Las que tienen cualquier otra finalidad.

La clase Herramienta es la encargada de definir la estructura básica de una herramienta. Esta clase es abstracta y extiende componente, de modo que ya tiene la gestión de mensajes implementada.

Al instanciarse esta clase, se cargará automática un fichero de propiedades de donde obtendrá las siguientes:

1. Nombre
2. Autor
3. Versión
4. Fecha
5. Filtro(Si filtra imagenes o no)

El fichero de propiedades deberá llamarse nombreClase.properties, donde nombreClase es el nombre completo de la clase, incluido el paquete, pero sustituyendo los '.' por '/'.

Por ejemplo, la herramienta lentes.herramientas.FiltroFuncion tiene sus propiedades en un fichero llamado lentes/herramientas/FiltroFuncion.properties.

Una herramienta está diseñada para que fácilmente pueda seguir el patrón MVC, model-view-controller. Esta clase, en si misma, desempeña el papel de controlador del componente. Además tiene un método getView() donde conseguir la view, la presentación de la herramienta. Por otra parte, en la implementación de la herramienta se deberían utilizar clases adicionales que definan el modelo.

Esta clase tendrá como propiedad una referencia a la view, de modo que puede acceder y modificar el modelo de la aplicación, además de utilizar los métodos de la view.

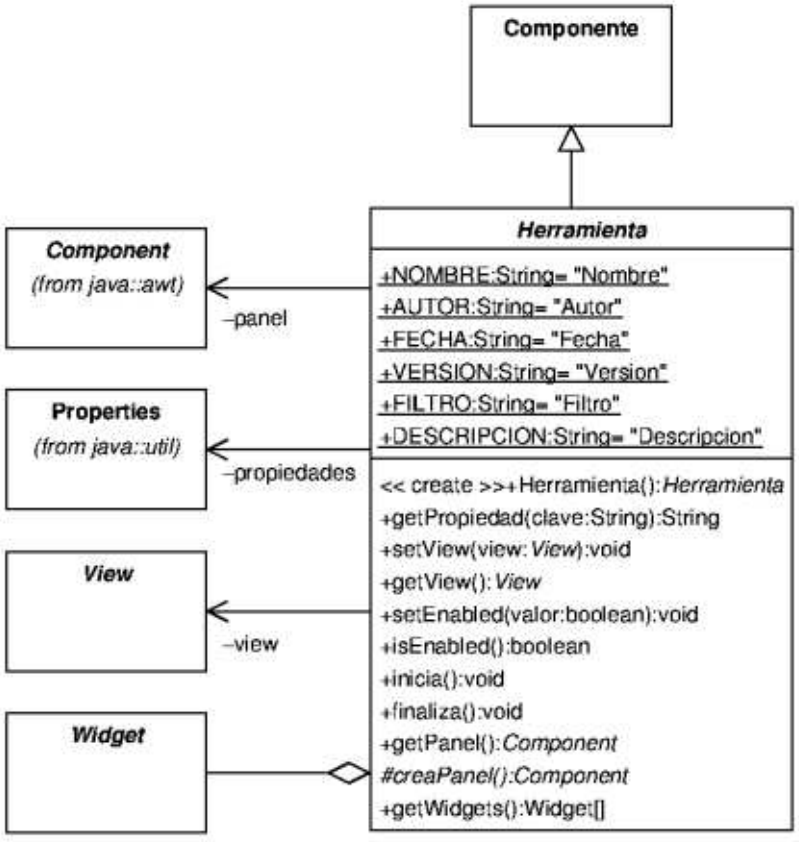


Figura 5.25: Herramienta

5.2.8. Herramienta Filtrofuncion

Ésta es una herramienta muy simple que servirá de ayuda para probar la aplicación.

Está compuesta por una sola clase que se encuentra en el paquete lentes.herramientas. El diagrama de clases se puede apreciar en la figura 5.26.

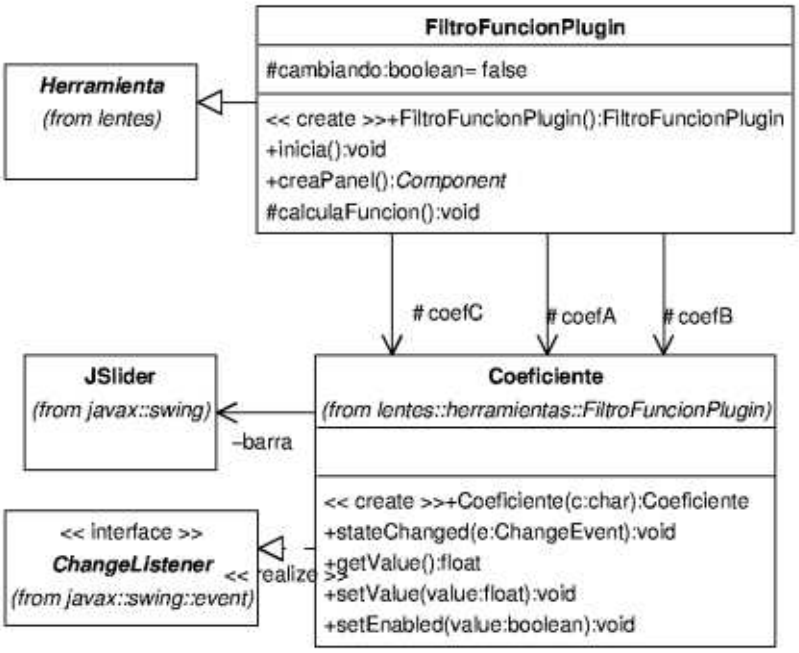


Figura 5.26: Paquete lentes.herramientas

Es una herramienta de tipo filtro, por lo que su finalidad es ayudar al usuario a calcular la función de error cometido en la toma de una fotografía. La ayuda que ofrece esta herramienta es mínima, ya que el usuario deberá introducir manualmente los coeficientes de una función polinómica que aproxime el error buscado.

Según lo visto en la sección 2.1, la función polinómica es del tipo:

$$p(x) = ax^4 + bx^3 + cx^2 + dx$$

Los coeficientes a, b y c son introducidos manualmente, y d es calculado según:

$$d = -a - b - c$$

En la figura 5.27 se puede apreciar el interfaz de usuario de esta herramienta.

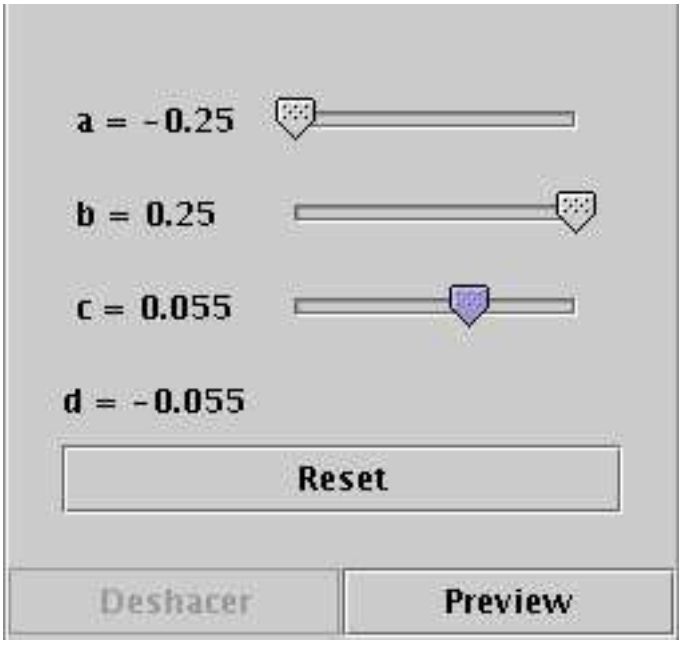


Figura 5.27: Interfaz de FiltroFuncion

El interfaz de usuario esta compuesto por tres barras donde seleccionar los coeficientes a, b y c. El coeficiente d no es modificable, ya que se calcula automáticamente como se ha visto anteriormente.

5.3. Implementación

Para la implementación del sistema se utilizó Java, tal como se ha comentado en la sección 3.4. Éste es un lenguaje de programación orientado a objetos con muchas características importantes. La razón principal para usarlo ha sido su facilidad de uso y que se trata de un lenguaje multiplataforma, no quedando restringida la aplicación al uso de Windows.

Para llevar a cabo la aplicación se utilizó la versión 1.4 del Java Developer Kit, aunque la aplicación debería funcionar con la versión 1.3.

Durante el desarrollo de la aplicación se emplearon las siguientes librerías:

- Commons-logging para escribir archivos log que muestren lo que hace la aplicación y ayuden a depurarla.
- JAI, que es u API para incorporar a las aplicaciones hechas en Java el procesamiento de imágenes con alto rendimiento.

Como entorno de trabajo se ha usado el IDE de Eclipse y como herramienta de construcción se ha utilizado Ant.

Para obtener más información sobre las herramientas puede consultarse el apéndice B.

5.4. Pruebas

5.4.1. Tipos de pruebas

Una vez terminada la implementación se pasa a probar si funciona o no correctamente.

La estrategia de pruebas que se sigue incluye tres tipos de pruebas:

Pruebas de unidad

Las pruebas de unidad tratan de verificar aisladamente cada método implementado en una clase.

De este modo, se prueban todos los métodos de las clases que conforman el sistema, verificando que se obtiene la salida esperada en función de la demanda.

A medida que se van descubriendo los errores, se corrigen, hasta conseguir que todas las operaciones de las clases funcionen correctamente.

Una vez que están todos los métodos probados y todos los errores corregidos, se integrarán progresivamente las clases para ver que, conjuntamente, realizan la misión para la cual fueron diseñadas. Este tipo de pruebas son las pruebas de integración, y son el siguiente tipo de pruebas a realizar.

Pruebas de integración

El objetivo principal de las pruebas de integración es detectar los fallos de interacción entre las distintas clases que componen el sistema.

Este tipo de pruebas se realizan después de las pruebas unitarias, de modo que cada clase probada unitariamente se inserta, de manera progresiva, dentro de la estructura del sistema. Así, las pruebas de integración se convierten en el

mecanismo para comprobar el correcto ensamblaje del sistema completo.

Al realizar la integración de los módulos, es importante centrarse en la búsqueda de defectos tales como aquellos que puedan provocar las excepciones lanzadas por los métodos, el uso de operaciones equivocadas y la invocación equivocada de métodos.

Pruebas de validación de requisitos

Al concluir las pruebas de integración se puede decir que el programa se encuentra completamente ensamblado, y que fueron encontrados y corregidos los errores de interacción entre las clases.

En este punto se comienzan a probar los requisitos del sistema, mediante las pruebas de validación de requisitos. La validación para el software se enfoca a las acciones visibles por el usuario, además de las salidas del sistema que puedan ser reconocidas por el. Estas acciones y salidas engloban las expectativas razonables del usuario, y están definidas en las especificaciones de los requisitos del software. La derivación de las pruebas de validación está basada en los casos de uso contenidos en el modelo de uso de la fase de diseño.

Estas pruebas de requisitos se hacen para dos tipos de requisitos:

- Los funcionales, tomados a partir del modelo de casos de uso.
- Los no funcionales, como pueden ser el tiempo de ejecución, la facilidad de uso, ...

Concluidas estas pruebas y corregidos los errores surgidos, se tiene la certeza de que el sistema cumple perfectamente los requisitos de usuario, teniendo la seguridad de que los cursos normales de los casos de uso se cumplen correctamente.

5.4.2. Realización de las pruebas

Las pruebas de unidad se han ido realizando a medida que se compleron las clases, de modo que, al terminar una clase se probaba si sus métodos funcionaban aisladamente.

Las pruebas de integración se han llevado a cabo a medida que se han ido obteniendo clases completas, de modo que se comprobaba que las clases que tenían implementadas en cada momento realizaban perfectamente su trabajo integradas con las demás.

Por último, las pruebas de validación de requisitos fueron relizadas una vez que el ciclo estaba termindo, probando así que el sistema cumplió los requisitos del usuario.

Capítulo 6

Segundo Ciclo

En este segundo ciclo se desarrollará una nueva herramienta para la aplicación. Esta herramienta es más completa y práctica que la anterior, FiltroFuncion. Tiene como meta calcular el error cometido en la fotografía después de que el usuario le indique líneas rectas en la realidad, que apecen curvadas en la imagen.

6.1. Análisis

6.1.1. Casos de uso expandidos

En este ciclo sólo es necesario realizar un único caso de uso, tal como se puede apreciar en la figura 6.1.

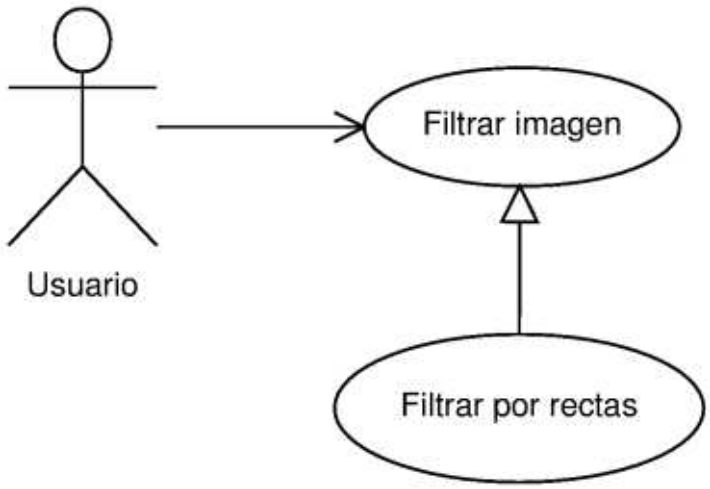


Figura 6.1: Casos de uso

Filtrar por rectas

Actores Usuario

Descripción El usuario introduce una o dos rectas que aparecen curvadas en la imagen y la aplicación construye una función que describe el error cometido en la fotografía.

Precondiciones Hay una imagen abierta.

Postcondiciones Se muestra la imagen corregida.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Indica 1 o 2 líneas rectas que aparecen curvadas	2. Realiza una previsualización
3. Acepta la previsualización	4. Filtra la imagen, guardando el estado anterior

Cursos alternativos

Línea 3 No acepta la previsualización. El sistema vuelve a mostrar la imagen sin filtrar.

6.2. Diseño

Se diseña el paquete lentes.herramientas.rectas para agrupar las clases necesarias para la herramienta Rectas.

Como cualquier herramienta de filtrado, su función es calcular la función de error cometido por un objetivos fotográfico sobre una fotografía.

Para su realización debe ayudarse de unos componentes gráficos que el usuario pueda modificar (widgets) y que le indiquen líneas curvas que deberían ser rectas. Estos componentes pueden verse en la figura 6.2.

Figura 6.2: Herramienta gráfica de Rectas

El interfaz de usuario de la herramienta permite escoger si se desea utilizar una o dos líneas para calcular el error. Con dos líneas se conseguirá una mejor aproximación, ya que la herramienta cuenta con más datos. El interfaz de usuario se puede apreciar en la figura 6.3.

Figura 6.3: Interfaz de usuario de Rectas

Clase RectasPlugin Esta clase concretiza la clase abstracta Herramienta. Representa una herramienta de filtrado para la aplicación.

La estructura de esta clase y sus relaciones se pueden apreciar en la figura 6.4.

Esta clase es la encargada de generar el interfaz de usuario y de controlar el proceso de cálculo de la función de error. Hace dos referencias a elementos de la clase Segmentos, que son log widgets que interactuarán con el usuario. Además, tiene una referencia a un objeto que implementa el interfaz método. Esta clase será la encargada de calcular la función a partir de uno o dos objetos de la clase Segmentos.

Con esta estructa sería muy fácil cambiar el método para calcular la función de error, ya que bastaría con cambiar la implementación del interfaz Metodo.

Interfaz Metodo Este interfaz define como será una clase que se encarga de calcular la función de error cometido en una fotografía.

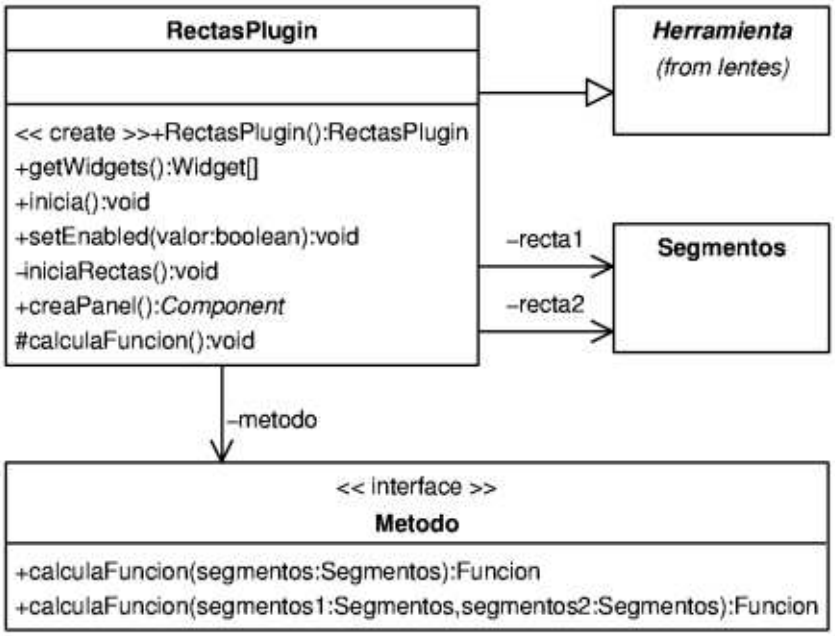


Figura 6.4: Clase RectasPlugin

En él se declaran dos métodos, uno para calcular la función de error a partir de un objeto Segmentos y otro para calcularla a partir de dos de estos objetos.

Clase MetodoBasico Implementación de la clase Metodo. Se basa en los calculos realizados en la sección 2.3.

En la figura 6.5 se puede observar que esta clase utiliza a la clase Bicubica. Esto lo hará en el cálculo de la función de error en el caso de que se utilicen dos objetos de la clase Segmentos.

Los calculos realizados contando con la descripción de una línea curva que debería ser recta, son una aproximación bastante simple y puede cometer bastaste error. Si se cuenta con la descripción de dos líneas se consigue una mejor aproximación.

Clase Bicubica Esta clase representa una función bicúbica del tipo:

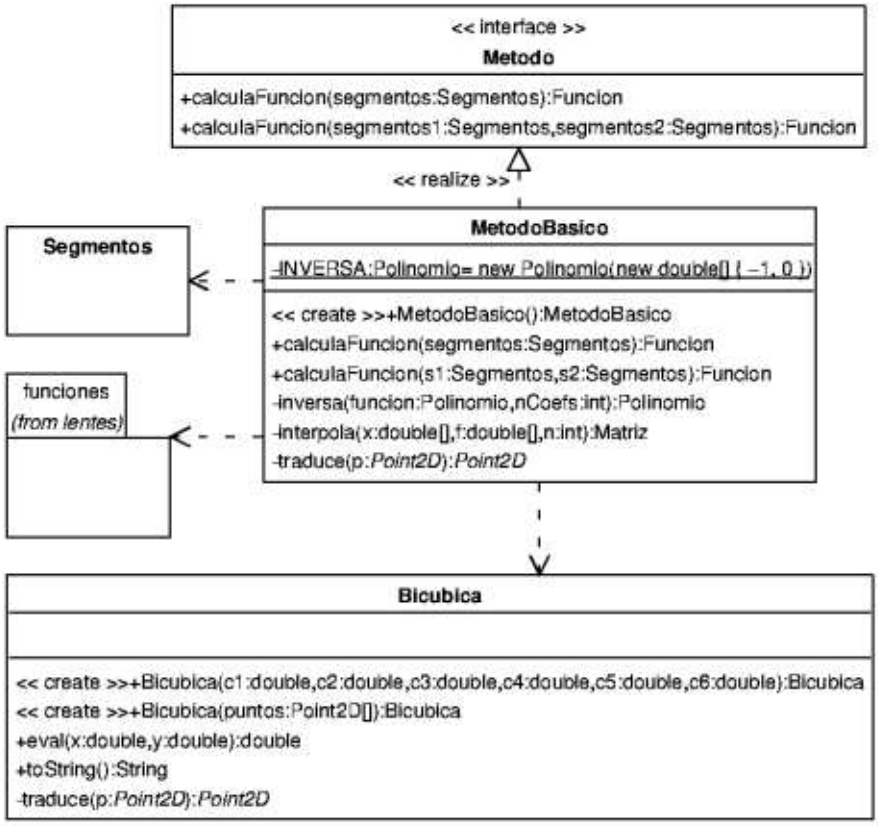


Figura 6.5: Clase MetodoBasico

$$f(x, y) = c_1x^2 + c_2x + c_3xy + c_4y + c_5y^2 + c_6$$

Es una clase muy simple y solamente tiene un método para evaluar la función en un punto, que viene dado por sus coordenadas.

Segmentos Esta clase extiende e implementa a la clase abstracta lentes.Widget.

A partir de un vector de puntos sobre la imagen, se trazan segmentos que unan cada punto con el siguiente. Estos puntos tienen coordenadas relativas (entre 0 y 1) al tamaño de la imagen.

Un usuario puede interatuar con una instancia de este elemento arrastrando

los puntos sobre la imagen a través del ratón.

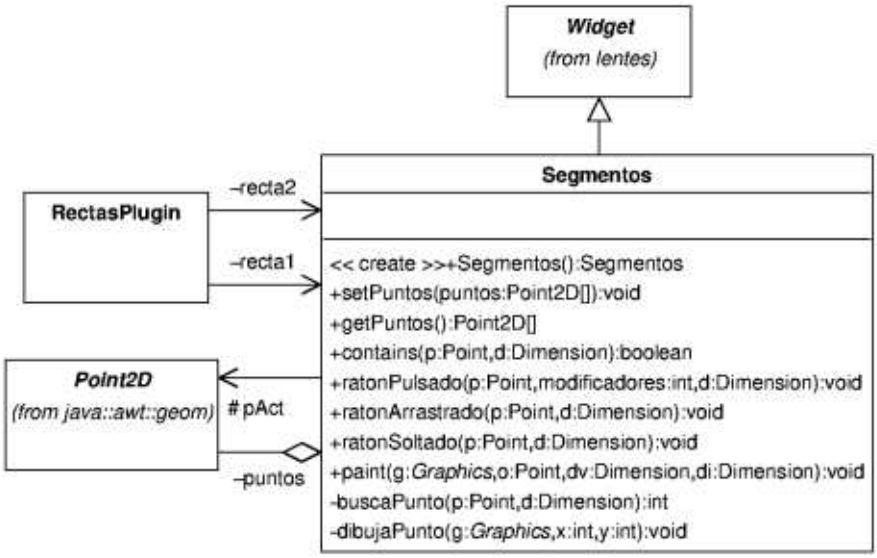


Figura 6.6: Clase Segmentos

El diagrama de clases de esta clase se puede ver en la figura 6.6.

En esta aplicación, un objeto de esta clase describirá, gracias al usuario, una línea curva que debería aparecer recta.

6.3. Implementación y Pruebas

La fase de implementación de este ciclo se ha realizado de acuerdo a las pautas de la sección 5.3.

Para conseguir una buena validación de lo desarrollado en este ciclo, se han llevado a cabo, nuevamente, los tres tipos de pruebas de la sección 5.4:

- Pruebas de unidad
- Pruebas de integración
- Pruebas de validación de requisitos

Capítulo 7

Tercer Ciclo

En este ciclo se desarrollarán dos herramientas para la aplicación. Una para gestionar una base de datos con las funciones de error cometidas por distintos objetivos fotográficos y otra para utilizar estas funciones en el filtrado y corrección de imágenes.

7.1. Análisis

7.1.1. Casos de uso expandidos

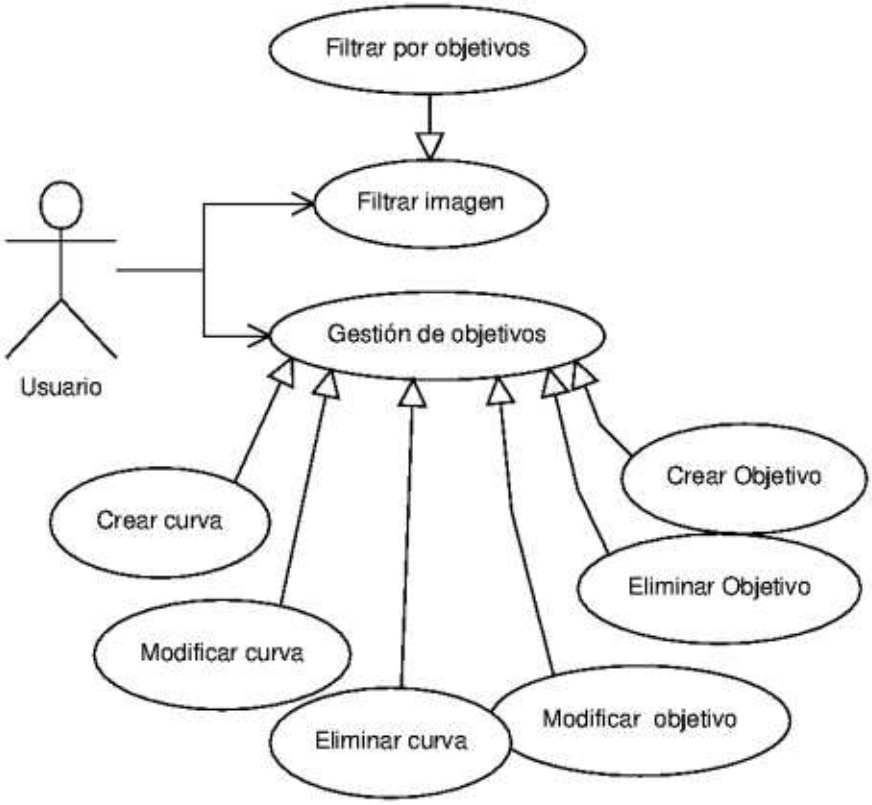


Figura 7.1: Casos de uso

Filtrar por objetivo

Actores Usuario

Descripción El usuario especifica el modelo de objetivo y la distancia focal con los que se tomaron la fotografía y la aplicación calcula la función del error cometido en la imagen.

Precondiciones Hay una imagen abierta.

Postcondiciones Se muestra la imagen corregida.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Especifica el modelo de objetivo	2. Carga el rango disponible para la distancia focal
3. Especifica la distancia focal	4. Realiza una previsualización
5. Acepta la previsualización	6. Filtra la imagen, guardando el estado anterior

Cursos alternativos

Línea 3 No acepta la previsualización. El sistema vuelve a mostrar la imagen sin filtrar.

Crear objetivo

Actores Usuario

Descripción El usuario da de alta una nueva descripción del error cometido por una lente especificando el nombre. Es necesario comprobar que el nombre sea único.

Precondiciones NA

Postcondiciones Se crea una nueva descripción de un objetivo.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Introduce el nombre del objetivo	2. Comprueba que no exista el nombre. 3. Crea el objetivo.

Cursos alternativos

Línea 3 El nombre no es único. Vuelve a pedir el nombre.

Eliminar objetivo

Actores Usuario

Descripción El usuario elimina una descripción del error cometido por una lente. Necesita confirmación.

Precondiciones Objetivo seleccionado.

Postcondiciones Objetivo eliminado.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Pide eliminar el objetivo seleccionado	2. Pide confirmación.
3. Confirma la operación.	4. Elimina el objetivo.

Modificar objetivo

Actores Usuario

Descripción El usuario modifica el nombre del objetivo.

Precondiciones Objetivo seleccionado.

Postcondiciones Se produce el cambio de nombre.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Pide modificar el nombre	2. Muestra un cuadro con el nombre antiguo donde poder modificarlo.
3. Cambia el nombre	4. Comprueba que no exista el nombre. 5. Modifica el nombre del objetivo.

Cursos alternativos

Línea 4 El nombre ya existe. Vuelve a pedir el nombre.

Crear curva

Actores Usuario

Descripción El usuario calcula el error cometido por un objetivo para una determinada distancia focal y lo almacena en la descripción de objetivo.

Precondiciones Objetivo seleccionada.

Postcondiciones Curva guardada.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Indica que quiere crear una nueva curva.	2. Muestra una herramienta para que el usuario pueda medir el error.
3. Mide el error	4. Pide que se introduzca la distancia focal.
5. Introduce la distancia focal.	5. Comprueba que no exista la distancia focal.
	6. Almacena el error

Cursos alternativos

Línea 4 La distancia focal ya existe. Vuelve a pedir la distancia focal.

Eliminar curva

Actores Usuario

Descripción El usuario elimina la curva seleccionada.

Precondiciones Curva seleccionada.

Postcondiciones Curva eliminada.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Indica que quiere eliminar una curva	2. Solicita confirmación
3. Confirma	4. Elimina la curva

Modificar curva

Actores Usuario

Descripción El usuario modifica la distancia focal asociada a la curva.

Precondiciones Curva seleccionada.

Postcondiciones Distancia focal almacenada.

Curso típico de eventos

Acción del Actor	Respuesta del Sistema
1. Solicita modificar la curva	2. Pide que se introduzca la distancia focal.
3. Introduce la distancia focal	4. Comprueba que no exista la distancia focal.
	5. Guarda el cambio.

Cursos alternativos

Línea 4 La distancia focal ya existe. Vuelve a pedir la distancia focal.

7.1.2. Modelo conceptual

Se puede apreciar, según el diagrama 7.2, como los conceptos del dominio aparecidos en la análisis anterior son: Objetivo y Curva.

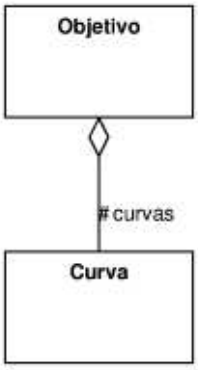


Figura 7.2: Modelo conceptual objetivos

Un objetivo es un objeto que describe el error cometido por un determinado objetivo fotográfico. Este error varií según la distancia focal a la cual se tome la fotografía. De esta manera, una Curva es un objeto que representa el error cometido por un objetivo para una determinada distancia focal.

De esta forma, un objetivo está relacionado con n curvas indexadas por la distancia focal.

7.2. Diseño

En este ciclo de desarrollo se gestarán dos herramientas para la aplicación Lentes: FiltroPlugin y MantemientoPlugin.

Como las dos herramientas están muy relacionadas entre sí, ambas compartirán el mismo paquete: lentes.objetivos.

7.2.1. El modelo

El diagrama del clases del modelo se puede apreciar en la figura 7.3 y está compuesto por la clase Objetivo y la clase Curva.

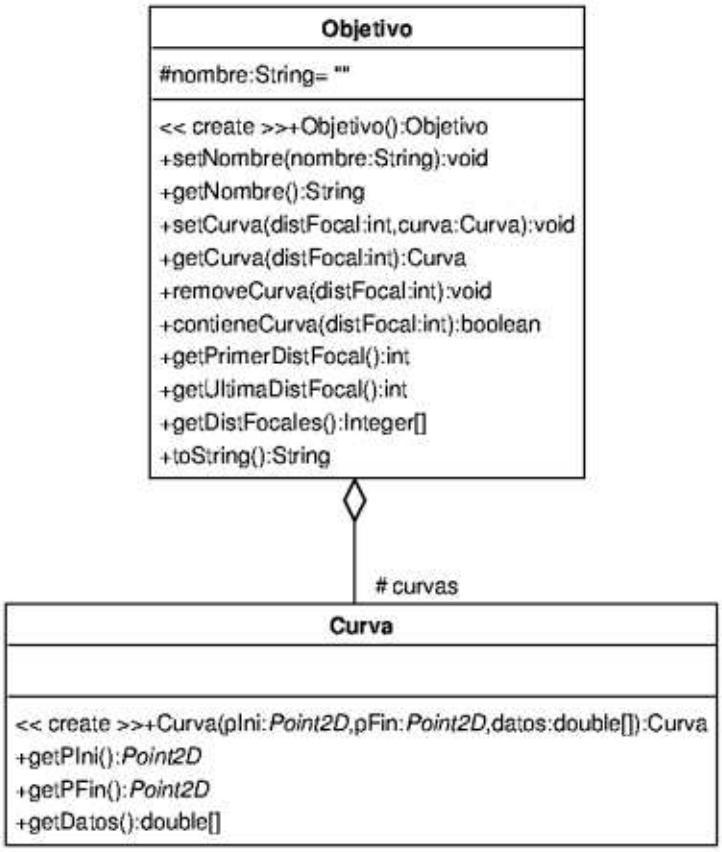


Figura 7.3: El modelo

Clase Objetivo Bean que representa un objetivo fotográfico. Esta clase tendrá las siguientes propiedades:

nombre El nombre del objetivo.

curvas Descripciones de la funciónes de error del objetivo fotográfico (curvas) indexados por la distancia focal.

Las curvas serán almacenadas en un objeto de la clase java.util.TreeMap, por lo que ya se guardaran ordenadas.

Esta clase también aporta métodos para obtener la menor y la mayor distancia focal registradas, de manera que se puede obtener el rango de uso del objetivo.

Clase Curva Clase que almacena la descripción de la función de error de un cometivo por un objetivo fotográfico para una determinada distancia focal.

Esta descripción representa la medición realizada en la sección 1.2.1. Almacena los siguientes valores:

- El punto inicial
- El punto final
- Una lista del valores entre [0, 1] que se corresponde con el muestreo realizado entre los extremos de la recta de medición.

Las propiedades de esta clase no son modificables. De esta manera se evitan posibles errores.

7.2.2. Los Generadores

Un vez obtenido el modelo, es necesario un objeto que pueda interpretar los datos almacenados y calcule la función de error que comete un determinado objetivo en un determinado contexto.

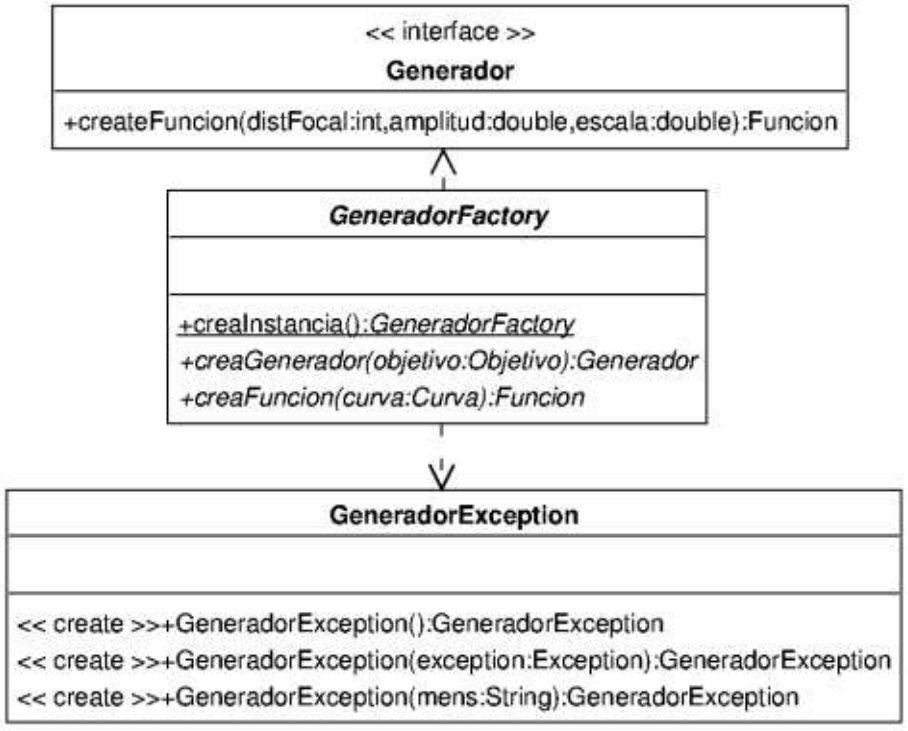


Figura 7.4: Generadores

Interfaz Generador Este interfaz tiene un método que se encarga de calcular una función a partir de una distancia focal, una amplitud y una escala.

Está diseñada para que la implementación envuelva un objeto de la clase Objetivo, y de esta forma poder obtener la función de error cometida por un objetivo fotográfico para una determinada distancia focal. Esta función puede ver escalada tanto en el eje x (escala) como en el y (amplitud).

Un escalamiento en el eje x se calcula mediante las operaciones de la sección

2.4.1. El cambio de escala en el eje y es trivial.

Clase GeneradorException Excepción lanzada por una clase que implemente al interfaz Generador cuando tenga algún problema.

Clase GeneradorFactory Factoría abstracta de la clase Generador con un método para crea un generador a partir de un objeto de la clase objetivo y otro para crear una función a partir de un objeto de la clase curva.

Tiene un método estático para obtener una instancia; por defecto instancia la clase lentes.objetivos.implementacion.FactoryPolinomio.

7.2.3. La ficha

La construcción del objeto generador asociado a un objetivo puede ser un proceso relativamente costoso, por lo cual se crea el concepto ficha, que será un objeto que asocie un objetivo a un generador, de manera que sólo sea necesario instanciarlo una vez. Además este objeto será el encargado de gestionar la permanencia de un objeto Objetivo.

Clase Ficha Bean encargado de relacionar un objetivo a un generador y de gestionar la permanencia de un objeto Objetivo.

Tiene las siguientes propiedades:

objetivo Un objeto Objetivo con la descripción de un objetivo fotográfico.

fichero El nombre del fichero que almacena la descripción del objetivo.

generador Generador de la función de error de un objetivo fotográfico.

modificada Si el bean objetivo ha sido modificado.

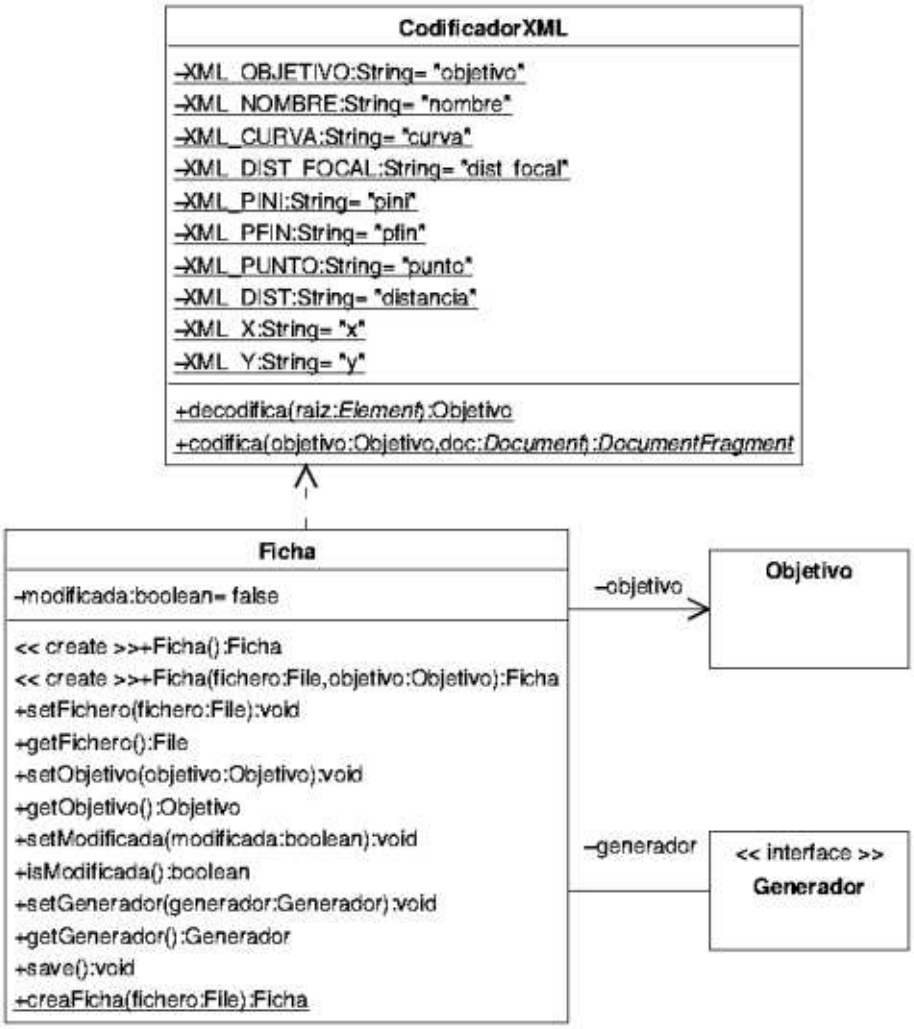


Figura 7.5: Ficha

Además, tiene un método para guardar una descripción de un objetivo fotográfico en el archivo asociado, y otro para cargar una descripción a partir de un nombre de fichero.

Para almacenar los bean Objetivo se utilizarán archivos XML. Para realizar la codificación y decodificación se utilizará la clase CodificadorXML.

Clase CodificadorXML Esta clase es utilizada por la clase Ficha para codificar a XML y decodificar de XML un objecto de la clase Objetivo. Utiliza el API DOM

para realizar estas tareas.

Que es XML

Antes de nada conviene repasar su historia y precedentes. La versión 1.0 del lenguaje XML es una recomendación del W3C desde Febrero de 1998. Está basado en el anterior estándar SGML (Standard Gene ralized Markup Language, ISO 8879), que data de 1986, y a su vez basado en el GML creado por IBM en 1969. Esto significa que aunque XML pueda parecer moderno, sus conceptos están más que asentados y aceptados de forma amplia.

SGML proporciona un modo consistente y preciso para aplicar etiquetas para describir las partes que componen un documento, permitiendo además el intercambio de documentos entre diferentes plataformas. Sin embargo, el problema que se atribuye a SGML es su excesiva dificultad; baste con pensar que la recomendación ocupa unas 400 páginas.

Así que, manteniendo su misma filosofía, de él se derivó XML como subconjunto simplificado, eliminando las partes más engorrosas y menos útiles. Como su padre, XML es un metalenguaje: es un lenguaje para definir lenguajes. Los elementos que lo componen pueden dar información sobre lo que contienen, no necesariamente sobre su estructura física o presentación, como ocurre en HTML.

7.2.4. Las herramientas

Como se ha dejado claro anteriormente, en este ciclo se desarrollarán dos herramientas para la aplicación: Una que gestionará la base de datos de Objetivos y otra que utilizará esa base de datos para corregir las fotografías.

Ya que las dos herramientas se basan en la misma base de datos, se va a crear una clase a la que extiendan ambas herramientas y que defina métodos comunes.

Clase Herramienta En la figura 7.6 se puede apreciar como se desarrolla la clase Herramienta, que extiende a la clase lentes.Herramienta.

Será la clase padre de las herramientas que se quieren implementar y, además, de definir métodos comunes, aporta una lista estática de fichas de objetivos mapeadas por el nombre de objetivo. De esta forma, la clases que extiendan a ésta podrán compartir los mismos datos.

La herramienta FiltroPlugin

Esta herramienta le permitirá al usuario escoger el objetivo fotográfico y la distancia focal con la que se tomó una determinada fotografía. Con estos datos la herramienta consultará la base de datos y obtendrá la función del error cometido en la toma de la fotografía, de manera que se podrá corregir inmediatamente.

El interfaz de usuario de la herramienta se puede ver en la figura 7.7.

Según lo explicado en la sección 2.2.1, puede ser necesario realizar un cambio de escala, por lo que la herramienta también permitirá introducir el factor por el que se escalará la función.

Además, como las mediciones pueden no ser perfectas, la herramienta también debe permitir modificar la amplitud de la función calculada.

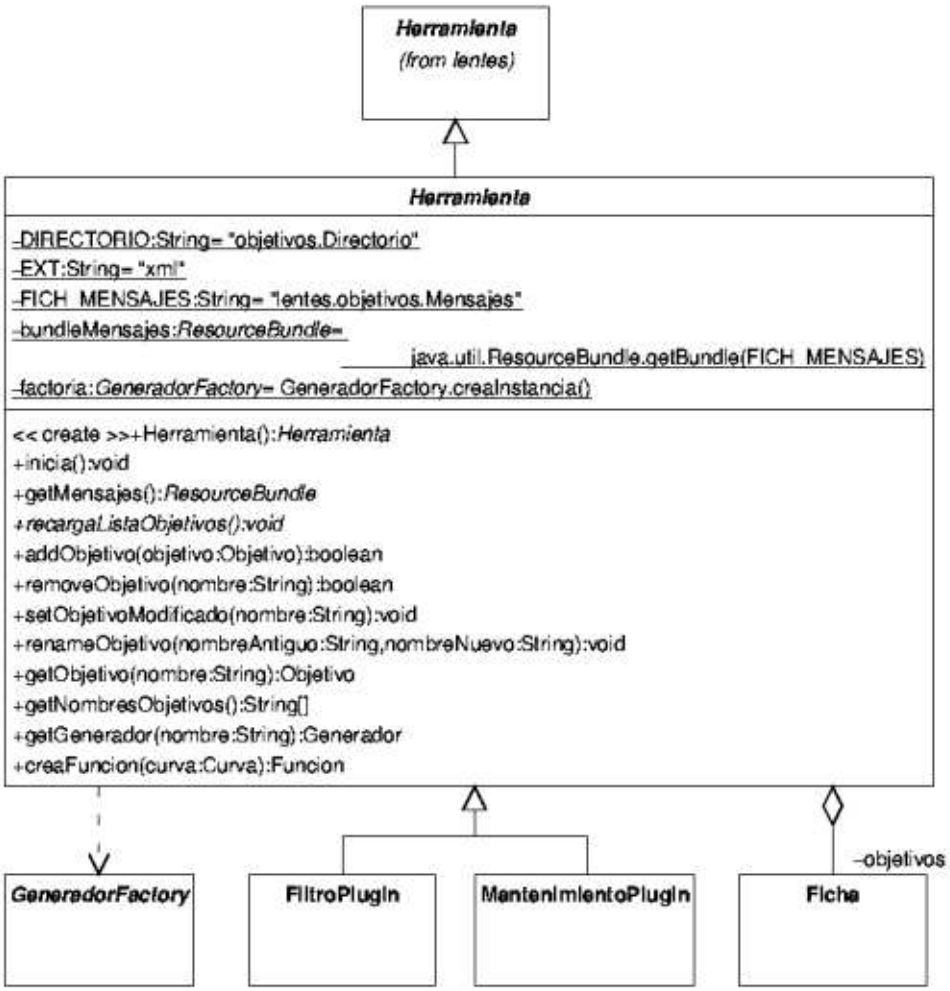


Figura 7.6: Clase Herramienta

Clase FiltroPlugin Esta clase extiende e implementa a la clase Herramienta. Su diseño se puede ver en la figura 7.8.

Es la encargada de generar el interfaz de usuario, que tiene un objeto javax.swing.JComboBox que permite escoger el objetivo deseado y tres objetos de la clase javax.swing.JSlider que permiten fijar la distancia focal, la escala y la amplitud.

Esta herramienta es un filtro en el que la View de la aplicación de añadirá botones para filtrar la imagen.

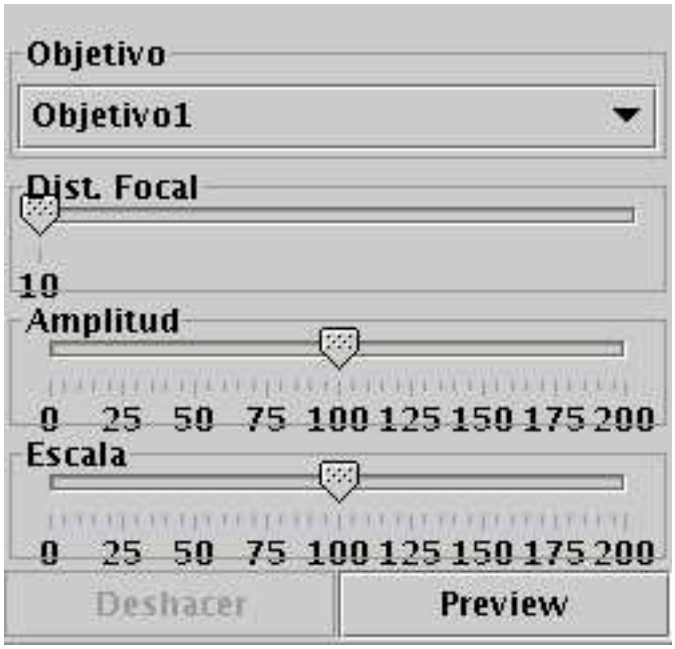


Figura 7.7: Interfaz de FiltroObjetivo

La herramienta MantenimientoPlugin

Esta herramienta no tiene por finalidad corregir una imagen, sino mantener la base de datos de descriptores de objetivos fotográficos utilizada por la herramienta FiltroPlugin.

El interfaz de usuario de la aplicación aparece en la figura 7.9. Como puede apreciarse esta herramienta debe permitir crear, eliminar y modificar objetivos y crear, eliminar y modificar curvas. Como esta herramienta no será un filtro a su interfaz no se le aportarán los botones para filtrar.

Para crear una curva es necesario realizar el experimento de la sección 1.2.1

Clase MantenimientoPlugin La clase MantenimientoPlugin extiende e implementa a la clase Herramienta. Esta clase constituye por sí misma la herramienta de mantenimiento de descriptores de objetivos.

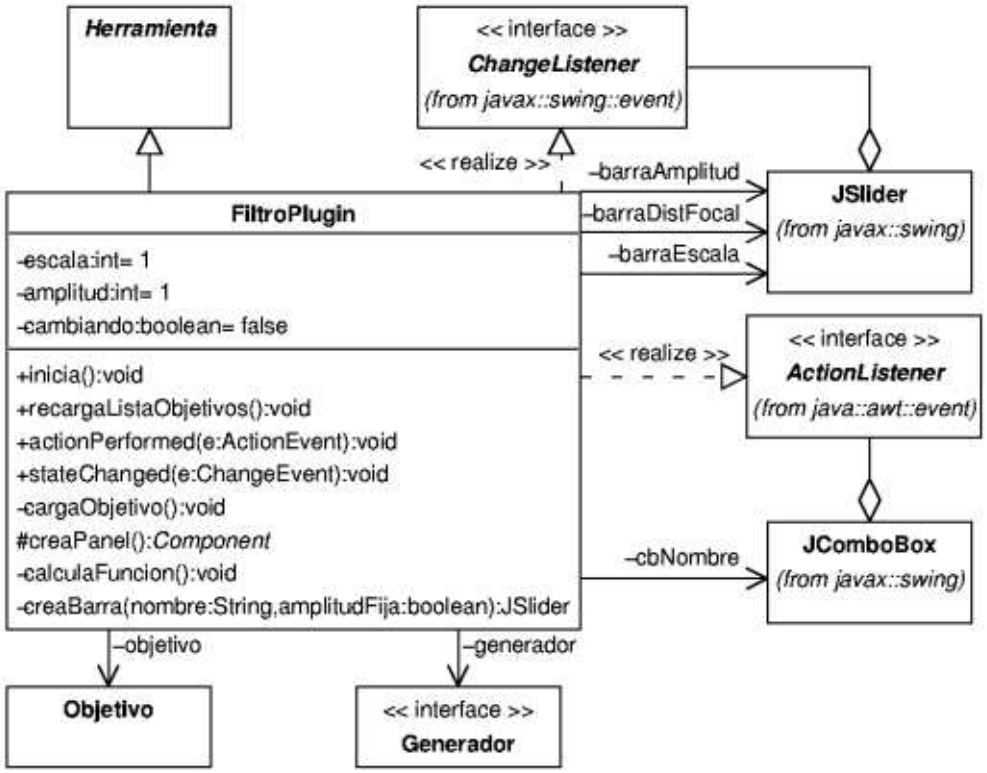


Figura 7.8: Clase FiltroPlugin

El diagrama de clases con las relaciones de esta clase puede verse en la figura 7.10.

Como puede apreciarse en la figura, esta clase tiene referencias a dos objetos de la clase javax.swing.JComboBox, que permiten escoger el objetivo y la curva. Al lado de cada combo aparecen tres botones para crear, modificar y eliminar objetivos o curvas.

Para esta herramienta se ha llevado a cabo una estructuración más completa de su diseño, ya que la complejidad que presenta es más alta. Por ello, se seguirá el patrón Model-View-Controller.

El modelo lo constituyen la clase Objetivo y la clase Curva.

El controlador está implementado en el paquete lentes.objetivos.action, a

Figura 7.9: Interfaz de MantenimientoObjetivo

través de una serie de acciones explicadas posteriormente.

Y la vista se genera con la llamada al método creaPanel() de la clase. Para que el usuario pueda interactuar con la herramienta se crean dos widgets: Medidor y Cuadros. El diagrama de clases de estos widgets se puede apreciar en la figura 7.11.

Clase Medidor Clase que extiende a la clase lentes.Widget. Es un componente gráfico para medir distancias. Consiste en una recta que se puede dividir en varios segmentos. Tomando la longitud de la recta como la unidad, los puntos que añadimos a la recta miden distancias relativas (entre 0 y 1).

Pulsando sobre el medidor con el botón izquierdo del ratón se añadirá una división y pulsando con el botón derecho se eliminará. Arrastrando los puntos pueden desplazarse las divisiones en el sentido del medidor y los extremos por toda la imagen.

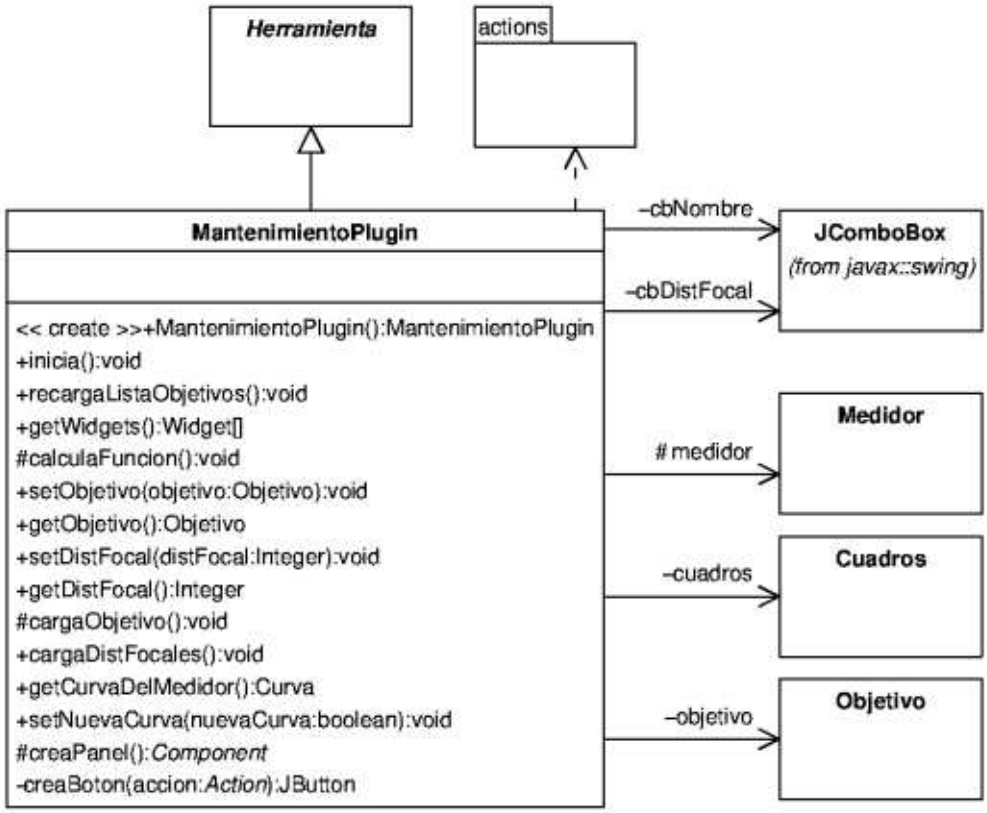


Figura 7.10: Clase MantenimientoPlugin

En la figura 7.12 se puede apreciar como se vería este componente sobre una imagen.

Clase Cuadros Clase que extiende a la clase lentes.Widget. Es un componente gráfico estático, no interactúa con el usuario. Simplemente dibuja dos rectas que dividen una imagen en cuadrantes, de manera que al usuario le sea más facil situarse.

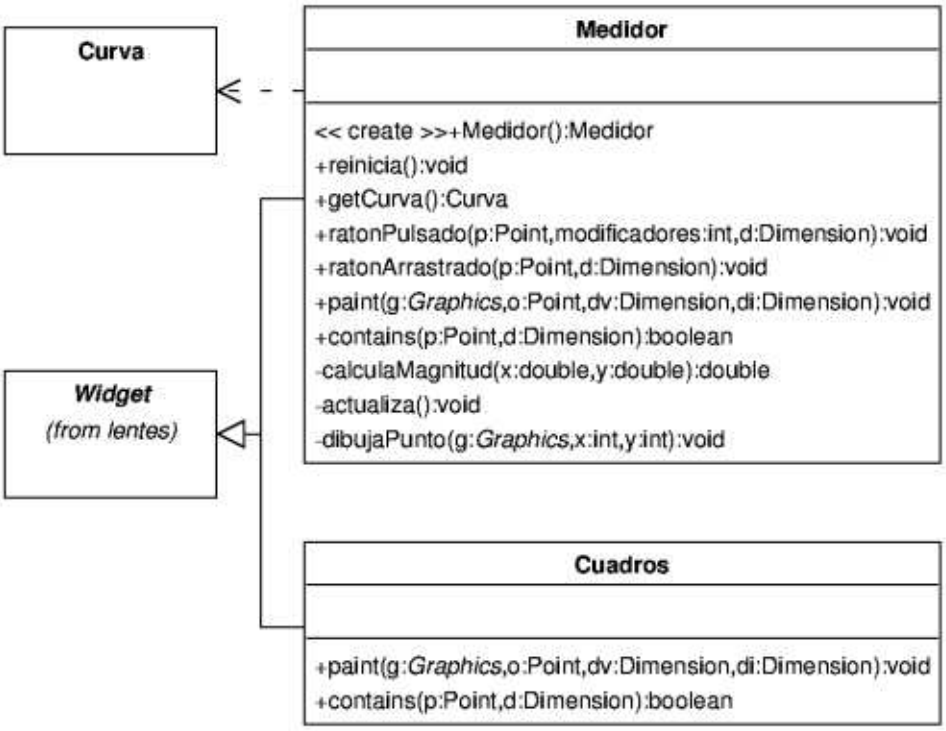


Figura 7.11: Widgets

Paquete lentes.objetivos.implementacion

Este paquete contiene las implementaciones de las clases abstractas e interfaces del paquete lentes.objetivos.

El diagrama de clases de este paquete se puede observar en la figura 7.13.

Clase FactoryPolinomio Esta clase extiende e implementa a la factoría abstracta lentes.objetivos.GeneradorFactory.

Para obtener un objeto Generar para un objeto de la clase Objetivo utiliza a la clase PolinomioGenerador.

Para poder crear un objeto Función a partir de un muestreo representado por un objeto de la clase Curva se basa en lo explicado en la sección 2.2.

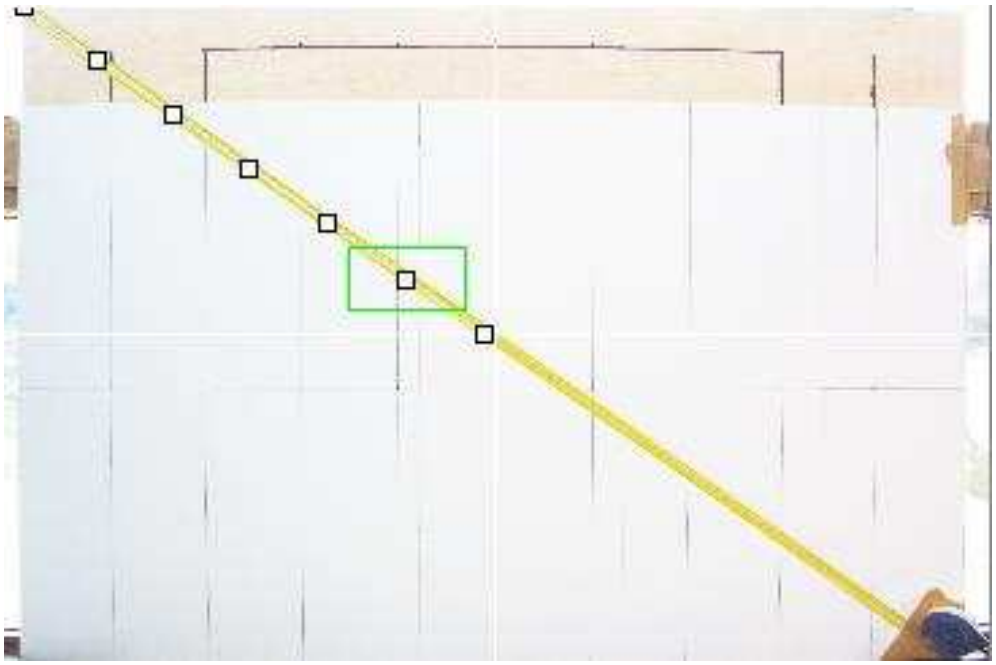


Figura 7.12: Medidor

Clase PolinomioGenerador Implementación del interfaz lentes.objetivos.Generador.

Una instancia de esta clase construye la función de error cometido por un objetivo, dada una distancia focal, una amplitud y una escala.

Ya que para cada distancia focal se construye una función polinómica de grado 4: $p(x) = ax^4 + bx^3 + cx^2 + dx$. Esta clase implementa un método que interpola las funciones para contruir la función buscada. Se puede consultar el método utilizado en la sección 2.2.

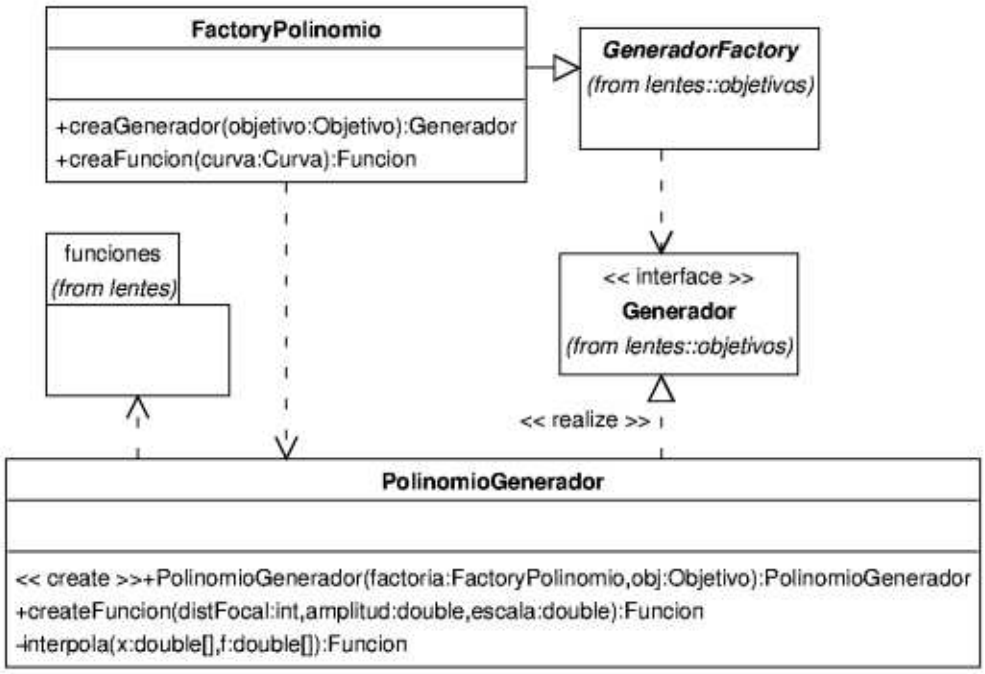


Figura 7.13: Paquete lentes.objetivos.implementacion

Paquete lentes.objetivos.actions

Este paquete contiene las acciones utilizadas por la herramienta MantenimientoPlugin. El diagrama de clases de este paquete puede ver en la figura 7.14.

Este paquete representa el controlador de la herramienta.

Clase BorraCurvaAction Acción para borrar la descripción del error cometido por un objetivo fotográfico para una distancia focal.

Clase BorraObjetivoAction Acción para borrar la descripción de un objetivo fotográfico.

Clase ModificaCurvaAction Acción para modificar la descripción del error cometido por un objetivo fotográfico para una distancia focal. Lanza una ventana

donde introducir la nueva distancia focal, en caso de que se quiera modificar, y coge la descripción de error generada por el medidor.

Clase CancelarNuevaCurvaAction Acción para cancelar la creación de una nueva curva.

Clase GuardarNuevaCurvaAction Acción para guardar una nueva curva.

Clase ModificaObjetivoAction Acción para modificar la descripción de un objetivo fotográfico. Pide el nuevo nombre del objetivo fotográfico.

Clase NuevaCurvaAction Acción para crear una descripción del error cometido por un objetivo fotográfico para una distancia focal. Lanza una ventana donde introducir la nueva distancia focal y muestra el medidor y dos botones, uno para guardar y otro para cancelar.

Clase NuevoObjetivoAction Acción para crear una descripción focal de un objetivo fotográfico. Lanza una ventana donde introducir el nombre del objetivo.

7.3. Implementación y Pruebas

Para este último ciclo la fase de implementación se lleva a cabo siguiendo las pautas de la sección 5.3 y la fase de pruebas de acuerdo a las de la sección 5.4.

Figura 7.14: Paquete lentes.objetivos.actions

.

Capítulo 8

Resultados

Una vez acabado el proyecto resulta evidente el siguiente paso: probarlo, ver las ventajas y los inconvenientes que puede ofrecerle al usuario.

Se les pidió a dos fotográfos profesionales que probaran la aplicación.

Por separado y sin saber nada sobre el proyecto, se les explicó, lo más brevemente posible, en que consistía la aplicación.

A continuación, se les entregó el manual de usuario impreso y se les pidió que intentasen instalar la aplicación e hiciesen uso de ella, probando cada característica presente en la aplicación.

Tras esto, se les dispensó un formulario en el que especificar todos los pros y contras que, a su juicio, había en la aplicación.

8.0.1. Pros

1. Instalación sencilla.
2. Fácil uso.
3. Buenos resultados.

8.0.2. Contras

1. Filtrado un poco lento.

2. El proceso para medir la distorsión introducida por un objetivo fotográfico a partir del experimento es un poco laborioso.

8.1. Experiencia Web

Con el fin de poder hacer llegar la aplicación a un mayor número de usuarios, surgió la idea de desarrollar un sitio web desde donde poder obtener el programa. Además, se podrían ofrecer documentación, noticias, foros, ... De esta forma podría existir una interacción entre usuario y desarrollador, lo cual enriquecería enormemente la aplicación.

Para desarrollar la web recurrí a la aplicación Forrest, que es un entorno de trabajo para la generación de documentación [1]. Esta herramienta permite desarrollar un sitio web con documentación, el estado del proyecto, cambios en versiones, cosas por hacer, ... de manera muy simple.

Una vez contruida la web era necesario conseguir hospedaje para ella. Además, sería muy beneficioso poder contar con herramientas on-line, ya que el sitio generado era estático.

La solución a estos problemas se encuentra en SourceForge.net

8.1.1. Sourceforge.net

SourceForge.net es un servicio web gratuito que proporciona a los desarrolladores de código abierto multitud de herramientas para desarrollar, almacenar y gestionar proyectos de software.

SourceForge aporta un entorno TDE(Team Development Environment), entorno de desarrollo en equipo, que incluye herramientas de administración para cada aspecto del proyecto. Entre otros servicios encontraremos:

- Repositorios de CVS.
- Listas de discusión públicas y privadas.

[1] Ver apéndice B

- Tablones de anuncios y foros.
- Facilidades para hacer copias de seguridad completas.
- Información para elección de licencias.
- Servidor Web Apache preparado para cada proyecto, con soporte PHP3 y base de datos MySQL.
- 100MB de espacio para almacenamiento web y directorio FTP anónimo.
- Servidor de ficheros.
- Gestor de parches.

8.1.2. Resultados en SourceForge

Este proyecto fue desarrolado a título individual y SourceForge está especialmente diseñado para trabajar en equipo. Por ello puede no parecer lo más conveniente, a pesar de lo cual ofrece una serie de características que le aportan mucho valor añadido al proyecto.

Por esta razón, registré el proyecto en Sourceforge. Para ésto tuve que redactar un pequeño resumen en inglés y esperar a que lo aceptasen. Dos días después ya estaba la web en la red.

Durante la vida de desarrollo de este proyecto se crearon tres versiones beta de la aplicación, además de la final, que fueron colgadas de la web para que pudieran ser probadas por los usuarios.

Lamentablemente, aunque me consta que hay muchos usuarios que probaron la aplicación[2], no se produjo ninguna comunicación con los usuarios.

En la figura 8.1 pueden apreciarse las estadísticas del sitio a día 10 de septiembre de 2004.

[2] A día de hoy hay 75 bajadas

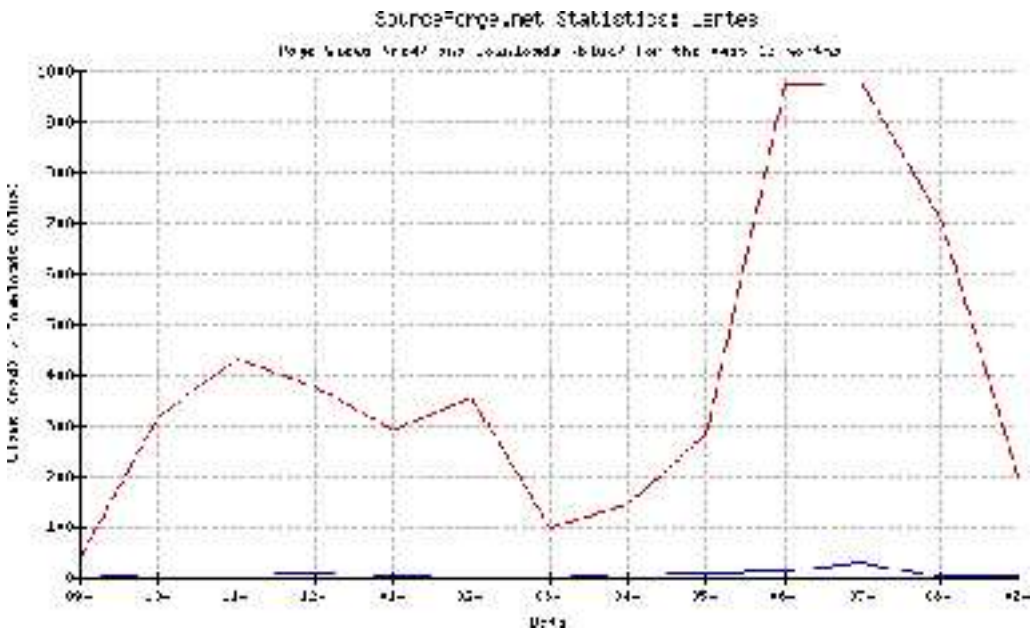


Figura 8.1: Estadísticas

8.1.3. Licencia

Un requisito indispensable para poder hospedar el proyecto en Sourceforge.net fue que la aplicación fuese de código abierto.

De esta manera, la aplicación fue publicada bajo la licencia GPL. Esto significa que el programa se convierte en software libre, con lo que la aplicación se puede distribuir y/o modificar bajo los términos de la GNU General Public License publicados por la Free Software Foundation.

8.2. Conclusión

Tras un breve estudio de los resultados obtenidos en las pruebas, se puede apreciar que la aplicación desarrollada cumple con las especificaciones solicitadas.

Se ha conseguido una aplicación fácil de usar, intuitiva, multiplataforma y que realiza las funciones para las que fue diseñada.

Como handicap importante está el rendimiento; debido a que se usó el lenguaje Java para su desarrollo, se consiguieron peores tiempos de filtrado que los que se podían conseguir usando un lenguaje más eficiente. Como contrapartida, se tiene una aplicación multiplataforma, robusto y seguro.

Apéndice A

Teoría: Interpolación de polinomios

A.1. Problema

Dada una función continua $f : \mathbb{R} \to \mathcal{P}_k(\mathbb{R})$ y una muestra

$$M = (y_1, P_1), (y_2, P_2), \ldots, (y_n, P_n)$$

$$P_i = a_{i,k}x^k + \cdots + a_{i,1}x + a_{i,0} \in \mathcal{P}_k(\mathbb{R}),\ i = 1..n$$

, siendo $y_i \in \mathbb{R}$ y $f(y) = P_y$, con $P_y \in \mathcal{P}_k(\mathbb{R})$.

Nos interesa encontrar una función continua $\widetilde{f} : \mathbb{R} \to \mathcal{P}_k(\mathbb{R})$, que aproxime a f.

Usando que la aplicación

$$\begin{array}{ccc} I : \mathcal{P}_k(\mathbb{R}) & \to^I & \mathbb{R}^{k+1} \\ p = a_k x^k + \cdots + a_1 x + a_0 & \rightsquigarrow & I(p) = (a_0, a_1, \ldots, a_k) \end{array}$$

es un isomorfismo entre los espacios vectoriales $(\mathcal{P}_k(\mathbb{R}), +, *)$ y $(\mathbb{R}^{k+1}, +, *)$, el problema se transforma en:

Encontrar una función $\widetilde{f} : \mathbb{R} \to \mathbb{R}^{k+1}$, que verifique que $\widetilde{f}(y) = (a_0, a_1, .., a_k)$, para cada $y \in \mathbb{R}$.

A.2. Resolución del problema

Definimos $\widetilde{f_j} : \mathbb{R} \to \mathbb{R}$, como el polinomio de Langrange [1] que interpola a la muestra

$$M_j = (y_1, a_{1j}), (y_2, a_{2j}), \ldots, (y_n, a_{nj}), \text{ con } j = 0..k$$

$\widetilde{f} = (\widetilde{f_0}, \ldots, \widetilde{f_k})$ así definida es solución del problema, ya que:

- $\widetilde{f}$ es contínua porque lo es cada una de sus componentes.
- $\widetilde{f}$ interpola a la muestra M.

$$\widetilde{f}(y_i) = (\widetilde{f_0}(y_i), \widetilde{f_1}(y_i), \ldots, \widetilde{f_k}(y_i)) = (a_{i0}, a_{i1}, \ldots, a_{ik}) \text{ para } i = 1, .., n$$

A.3. Cálculo del error

Nos interesa acotar el error cometido al aproximar la función f por $\widetilde{f}$.

A.3.1. Teorema (para funciones de $\mathbb{R} \to \mathbb{R}$)

Sea $f : \mathbb{R} \to \mathbb{R}$ una función $C^{n+1}[a, b]$ y sea p el polinomio de grado menor o igual que n, que interpola a f en n puntos distintos $x, \ldots, x_n \in [a, b]$.

Entonces $\forall x \in [a, b]$, $\exists\ \xi_x \in (a, b)$ tal que

$$f(x) - p(x) = \frac{1}{n!} f'^n(\xi_x) \prod_{i=1}^{n} (x - x_i)$$

[1] Se puede utilizar cualquier otro método de interpolación

A.3.2. Teorema (para funciones de $\mathbb{R} \to \mathbb{R}^{k+1}$)

Sea $f : \mathbb{R} \to \mathbb{R}^k$, tal que $f_i \in C^{n+1}[a,b]$, $i = 1,..,n$ y sea $\widetilde{f}$ la función cuya componente i-esima es el polinomio de interpolación de f_i en n puntos distintos $(y_1,..,y_n) \in [a,b]$.

Entoces $\forall y \in [a,b]$ $\exists$ ξ_y tal que

$$||f(y) - \widetilde{f}(y)|| \leq \frac{\sqrt{k+1}}{n!}||f'^n(\xi_y)|| \prod_{j=1}^{n}(y - y_j)$$

Demostración

Por el teorema de la sección A.3.1, tenemos que dado $y \in [a,b]$ $\exists$ $\xi_y^i \in [a,b]$ tal que

$$f_i(y) - \widetilde{f}_i(y) = \frac{1}{n!} f_i'^n(\xi_y^i) \prod_{j=1}^{n}(y - y_j) \ \forall i = 0,..,k$$

Entonces

$$||f(y) - \widetilde{f}(y)||^2 = \sum_{i=0}^{k}(f_i(y) - \widetilde{f}_i(y))^2 = \sum_{i=0}^{k}(\frac{1}{n!} f_i'^n(\xi_y^i) \prod_{j=1}^{n}(y - y_j))^2$$
$$= (\frac{1}{n!})^2(\prod_{j=1}^{n})(y - y_j))^2 \sum_{i=0}^{k}(f_i'^n(\xi_y^i))^2$$

Por otra parte

$$\sum_{i=0}^{k}(f_i'^n(\xi_y^i))^2 \leq \sum_{i=0}^{k}||f'^n(\xi_y^i)||^2 \leq \sum_{i=0}^{k}||f'^n(\xi_y)||^2 = (k+1)||f'^n(\xi_y)||$$

donde $\xi_y \in [a,b]$ tal que $||f'^n(\xi_y)|| \geq ||f'^n(\xi_y^i)||$, $\forall i = 1,..,k$.

Entonces

$$||f(y) - \widetilde{f}(y)||^2 \leq \frac{1}{n!}^2 (k+1)||f'^n(\xi_y)||^2 \prod_{j=1}^{n}(y - y_j)$$

por tanto

$$||f(y) - \widetilde{f}(y)|| \leq \frac{\sqrt{k+1}}{n!}||f'^n(\xi_y)|| \prod_{j=1}^{n}(y - y_j)$$

Apéndice B

Herramientas utilizadas

JDK14

Kit de desarrollo Java versión 1.4. Contiene, entre otros, el compilador y la máquina virtual de Java, además de las librerías estándar.

JAI

JAI es el acrónimo de Java Advanced Imagen API. Un API para incorporar a las aplicaciones hechas en Java el procesamiento de imágenes con alto rendimiento.

Sus principales ventajas son:

1. Independencia de la plataforma
2. Entorno de trabajo extensible
3. Alto rendimiento
4. Facilidad de programación, con lo que se acorta el tiempo de desarrollo.
5. Centrado el plataformas de red.

6. Soporta imágenes con formatos BMP, GIF, FPX, JPEG, PNG, PNM, TIFF.

7. Más de 100 operaciones de procesamiento de imágenes, la mayoría nativas con lo que se consigue mayor rendimiento.

8. Ejecución en diferido.

9. Tiling, división de imágenes en bloques, de manera que sólo es necesario bajarse o procesar una sección de la imagen, lo que reduce el ancho de banda necesario.

Ant

Apache Ant es una herramienta de construcción basada en Java. En teoría, es una especie de Make, pero sin los inconvenientes de éste.

Éstas son sus ventajas principales:

1. Independiente del sistema operativo.

2. Fácilmente extensible a través de clases Java.

3. Los archivos de configuración están escritos en XML.

Forrest

Forrest es un entorno de trabajo para la generación de la documentación de un proyecto. Los documentos se escriben utilizando XML y la herramienta utiliza Apache Cocoon para transformar las fuentes XML en un sitio web, a través de archivos XSLT.

LaTex

LaTeX es un conjunto de macros de TeX, no una modificación en ningún modo de lo que es TeX. TeX es una aplicación informática (y el lenguaje de marcado que utiliza) destinada a la edición y composición tipográfica de textos. Fue creada por el Dr. Donald E. Knuth de la Universidad de Stanford, Palo Alto (California) en los Estados Unidos.

La idea principal de LaTeX es ayudar a quien escribe un documento a centrarse en el contenido más que en la forma. Es muy utilizado para la composición de tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea. LaTeX fue escrito originariamente en 1984 por Leslie Lamport y es Software Libre bajo licencia LPPL.

El modo en que LaTeX interpreta la "forma"que debe tener el documento es mediante "etiquetas". Puede resultar extraño que hoy en día se siga usando algo que no es WYSIWYG (lo que ves es lo que obtienes) pero las características de LaTeX siguen siendo muchas y muy variadas. También hay varias herramientas (aplicaciones) que ayudan a una persona a escribir estos documentos de una manera más visual (LyX, TeXmacs y otros). A estas herramientas se las podría denominar WYSIWYM (lo que ves es lo que pensaste).

Una de las ventajas de LaTeX es que puede ser exportado muy fácilmente a Portable Document Format (PDF) y a otros formatos como HTML usando latex2html.

Eclipse

Un framework de código abierto independiente de la plataforma, para construir ricas aplicaciones clientes de cualquier tipo. La aplicación más importante que ha

sido realizada con el framework es el altamente afamado IDE Java y compilador, que también son usados para desarrollar el mismo eclipse.

Fue originariamente creado por IBM pero el desarrollo actual es manejado por la Fundación Eclipse, un consorcio sin ánimo de lucro compuesto principalmente por líderes de la industria.

El workbench(IDE) de Eclipse emplea módulos anexos (en inglés plug-in) para proveer toda su funcionalidad, a diferencia de otros IDE's donde ésta generalmente está siempre presente, la necesite el usuario o no. El mecanismo de módulos anexos le permite al workbench dar soporte a diferentes lenguajes además de Java. Por ejemplo, existe un módulo anexo para dar soporte a C/C++. Existen otros módulos anexos para añadir un poco de todo, desde telnet hasta soporte a bases de datos.

Los widgets de Eclipse están basados en SWT un toolkit de tercera generación para JAVA que mejora toolkits de primera y segunda generación de Sun (AWT y Swing, respectivamente). El interfaz del workbench también depende de una capa intermedia GUI llamada JFace, que simplifica la construcción de aplicaciones basadas en SWT.

Gimp

El GIMP es el programa de manipulación de imágenes del proyecto GNU (GNU Image Manipulation Program). Se publica bajo la licencia GNU General Public License.

GIMP fue ideado como una alternativa de software libre al popular programa de retoque fotográfico Photoshop. La primera versión se desarrolló para sistemas Unix y fue pensada especialmente para GNU/Linux, sin embargo actualmente (versión 2.0) existen versiones totalmente funcionales para Windows y para Mac OS X.

Commons-Logging

El paquete Commons-Logging es un puente muy ligero entre las diferentes librerías de logging. De esta manera se puede escribir una aplicación sin dependencias sobre ningún paquete particular de logging.

DIA

Dia es un programa de creación de diagramas basado en gtk+, distribuido bajo la licencia GP.

Dia fue diseñado para parecerse al programa comercial de windows 'Visio'. Puede ser empleado para crear diferentes clases de diagramas. Actualmente tiene objetos especiales para dibujar diagramas de relación, UML, diagramas de flujo, diagramas de red, circuitos simples..... Ofrece la posibilidad de añadir nuevas formas escribiendo simples archivos XML, usando un subconjunto de SVG para dibujar la forma.

Apéndice C

Manual de instalación

C.1. Requisitos

Para poder ejecutar la aplicación es necesario tener instalados los siguientes programas:

Java 2 Platform, Standard Edition Está probado en versiones superiores a la 1.4, pero debería funcionar con versiones anteriores. Puede ser necesario tener fijada la variable de entorno JAVA_HOME.

Java Advanced Imaging Es necesaria la versión 1.1.1. Con otras versiones pueden obtenerse resultados no deseados.

C.2. Instalación

La instalación de la aplicación es muy sencilla. Simplemente es necesario descomprimir el archivo zip y ya estará listo para probarlo.

C.2.1. Estructura de la instalación

Cuando se descomprime el zip se crean los siguientes directorios:

bin Aquí están los scripts para lanzar la aplicación.

etc En este directorio se encuentra el archivo de configuración.

lib Donde está el jar de la aplicación y las librerías utilizadas.

docs Directorio con la documentación de la aplicación.

plugins Directorio con los plugins.

objetivos Directorio donde se almacenan las descripciones del error cometido por distintos objetivos fotográficos.

license Directorio con documentos de licencia.

C.2.2. Instalación de plugins

Para instalar un nuevo plugin es suficiente con dejar el archivo jar en la carpeta plugins.

Ahora simplemente se debe lanzar la aplicación y se podrá disfrutar de la nueva herramienta.

C.3. Ejecución

En la caperta bin se encuentran los scripts que lanzarán la aplicación tanto para Windows como en Unix.

Para ejecutar la aplicación en Windows basta con hacer doble-click sobre el fichero lentes.bat.

En Linux o Unix se debe ejecutar el script lentes.sh. Previamente es necesario que este archvio tenga permiso de ejecución. Para realizar esto se debe ejecutar la instrucción: chmod +x lentes.sh.

Apéndice D

CDROM

El CDROM de este proyecto está estructurado de la siguiente manera:

. En la raíz se encuentra la memoria del proyecto.

web En este directorio se encuentra el sitio web, que contiene la documentación de la aplicación.

imagenes Aquí están las pruebas realizadas en la sección 1.2.1.

lentes-src Código fuente de la aplicación.

memoria-src Código fuente de la memoria.

lentes Aplicación.

zips En este directorio se encuentran los archivos zip de la aplicación.

software En este directorio aparecen versiones para Linux y Windows de los programas necesarios para ejecutar la aplicación.

Apéndice E

Estructura del código

. En la raíz se encuentran los archivos de configuración necesarios para utilizar Ant y Forrest.

build En este directorio se construye la aplicación.

- **build/classes** Directorio donde se almacenan las ficheros .class, creados a partir del código fuente de java.
- **build/dist** En este directorio se crea la distribución de la aplicación.
- **build/site** Aquí se genera la documentación web a través de Forrest.
- **build/temp** Directorio temporal utilizado por Forrest.

lib Directorio donde se almacenan las diferentes librerías que utiliza la aplicación. Actualmente solamente se usa commons-logging de Jakarta.

license Directorio que contiene la licencia de uso de la aplicación y la de commons-logging.

objetivos Aquí se guardan los archivos xml con los descriptores del error cometido por distintos objetivos fotográficos.

src Directorio de los archivos fuente. Está dividido en cuatro carpetas.

src/bin Aquí estan los archivos necesarios para lanzar la aplicación en Windows y en Linux.

src/documentation Directorio con los archivos fuente necesarios para generar la documentación web de la aplicación a través de Forrest.

src/etc Directorio con el archivos de configuración de la aplicación.

src/java Directorio con los archivos necesarios para crear la aplicación. Incluye los ficheros fuente de Java y archivos de propiedades.

Apéndice F

build.xml

La herramienta de construcción Apache Ant utiliza un archivo de configuración que describe como construir la aplicación. Este archivo se debe llamar build.xml y en él se definen una serie de objetivos. En cada objetivo se realiza una determinada tarea y pueden definirse dependencias entre objetivos.

Objetivos

prepare Crea la estructura de directorios necesaria para compilar la aplicación.

prepareDist Crea la estructura de directorios necesaria para construir una versión de distribución de la aplicación.

clean Elimina los directorios utilizados en la construcción de la aplicación.

compile Compila el código fuente de Java.

dist Crea la versión de distribución de la aplicación.

javadoc Crea la documentación javadoc de la aplicación.

all Objetivo por defecto, llama a los objetivos clean y dist.

zipBin Construye un archivo comprimido zip con la versión de distribución de la aplicación, el sitio web y la documentación javadoc.

zipSrc Construye un archivo comprimido zip con el código fuente de la aplicación.

zipSite Construye un archivo comprimido zip con el sitio web.

Apéndice G

Costes

En la realización de este proyecto intervinieron distintos perfiles según las distintas características que tienen las actividades. Así, se contó con un investigador, un analista y un programador.

El precio por hora de los recursos es el siguiente:

Pérfil	Euros/hora
Analista	10
Programador	5

El proyecto se dividió en dos grandes bloques. En el primero de ellos se desarrolló la aplicación, interviniendo los perfiles analista y programador. En el tercer bloque se realizó la formalización de la documentación y participaron también los dos perfiles.

Toda esta información se resume en la siguiente tabla:

Etapa	Trabajo	Coste
Desarrollo	1000 horas	7500
Formalización documentación	350 horas	2000
Total	1350 horas	9500

Bibliografía

[1] LARMAN, C., UML y Patrones. Introducción al análisis y diseño orientado a objetos, Prentice Hall.

[2] MULLER, P.A., Modelado de objetos con UML, Ediciones Gestión 2000, Barcelona, 1997.

[3] PRESSMAN, R.S., Ingeniería del Software. Un enfoque práctico, McGraw Hill, 1998, 4ªedición.

[4] MARÍA DEL CARMEN GUISÁN. Econometría. McGraw-Hill. Madrid 1997.

[5] Gamma E., Helm R., Johnson R., Vlissides J., Patrones de Diseño. Elementos de Software Orientado a Objetos Reutilizable, 1ª edición, Madrid, España, Pearson Educación S.A., 2003.

[6] Burden, R. L. y Faires, J. D. Análisis Numérico. ITP, 1998.

[7] Conde, C. y Winterm, G. Métodos y Algoritmos básicos del Algebra Numérica. Reverté, 1990.

[8] Gerald, C. F. y Wheatley, P. O. Applied Numerical Analysis. Addison-Wesley, 1990.

[9] Kincaid, D. y Cheney, W. Análisis Numérico: las matemáticas del cálculo científico. Addison-Wesley, 1994.

[10] Viaño, J. M. Lecciones de Métodos Numéricos. 2, Resolución de ecuaciones numéricas. Tórculo, 1997.

[11] Arnow, David M.; Weiss, Gerald: Introducción a la programación con JAVA. Un enfoque orientado a objetos. Addison-Wesley, 2000.

[12] Apache Forrest - http://xml.apache.org/forrest

[13] Wikipedia - http://es.wikipedia.org

[14] Jakarta Commons - http://jakarta.apache.org/commons

[15] World Wide Web Consortuim - http://www.w3c.org

[16] Panorama Tools - http://www.path.unimelb.edu.au/ dersch/

[17] Lensdoc - http://www.andromeda.com/info/lensdoc.html

[18] Java - http://wwws.sun.com/software/java/index.html

[19] Java Advanced Imaging - http://java.sun.com/products/java-media/jai/index.jsp

[20] Object Management Group - http://www.omg.org

[21] Eclipse - http://www.eclipse.org

[22] Dia - http://www.gnome.org/projects/dia/