Salvador Castillo Beltrán
Alberto Aguilera Juárez

# Práctia 2 - Nuevas Tecnologías de la Programación

## 1. Enunciado del problema

La práctica consiste en la implementación utilizando la librería Qt (versión 4.x) de un editor gráfico de dibujo vectorial al estilo del famoso programa xfig de linux, aunque bastante más simple. Para diseñar el interfaz gráfico puede usarse si se desea Qt Designer, Qt Creator o QDevelop, aunque es recomendable programar el interfaz gráfico manualmente.

**Requerimientos mínimos**

Los requerimientos mínimos que se piden al programa son los siguientes:

**1)** La zona del tablero (widget central) se implementará creando una subclase de QWidget, donde deben controlarse los eventos de ratón y eventos paint para que no se pierdan los contenidos del tablero, con este evento.

**2)** Incluir una barra de menús, que contenga obligatoriamente el menú **Fichero** (el resto de menús son opcionales) con al menos las siguientes opciones:
- **Nuevo:** Borra el dibujo actual del tablero.
- **Salir:** Para salir del programa.

**3)** El programa debe permitir introducir al menos objetos línea y objetos rectángulo. Para ello podría usarse por ejemplo el siguiente método:
- **Líneas**: Pinchar con el botón izquierdo del ratón en la posición inicial y posteriormente en la posición final.
- **Rectángulos:** Pinchar con el botón izquierdo del ratón en la posición del vértice superior izquierdo del rectángulo y luego en la posición del vértice inferior derecho.

**4)** Habrá botones (línea y rectángulo) para seleccionar el modo actual. Si por ejemplo pinchamos en el de **líneas** el editor pasa a modo *dibujar líneas* y a partir de ahora se comienzan a introducir líneas al ir pinchando con el ratón en la zona de dibujo.

Opcionalmente se pueden incluir en el programa otras mejoras, explicándolas en la documentación del programa. Podéis mirar por ejemplo el programa *xfig* para tomar ideas de qué mejoras introducir. Por ejemplo algunas mejoras podrían ser las siguientes:

- Dibujar los distintos objetos mediante el uso de líneas elásticas en lugar de con el método explicado en el punto 3 de los requisitos mínimos. Después de pinchar con el ratón en el primer punto del objeto se iría dibujando al arrastrar el ratón, y quedará fijo al soltar el botón del ratón.

- Permitir introducir otros objetos como círculos, elipses, polilíneas, etc.

- Uso de barras de utilidades (toolbars), por ejemplo para incluir los botones de tipo de objeto a dibujar.

- Permitir introducir texto.

- Mover, borrar y copiar objetos.

- Poder seleccionar un vértice de un objeto para escalarlo o bien moverlo.

- Opciones de deshacer (Undo) y rehacer (Redo).

- Poder seleccionar ciertas características de la línea como si es continua o discontinua, grosor, color, etc.

- Edición de objetos (cambio de sus características).

- Salvar y cargar el dibujo en un fichero.

- Incluir el menú Opciones, con opciones como selección del color para los objetos que se dibujen a continuación.

- Incluir un menú de Ayuda.

- Uso de diálogos para cargar o salvar (en caso de que se implemente la mejora).

- Zoom.

# 2. Análisis de requerimientos

## 2.1. Requerimientos funcionales

De todos los **requisitos mínimos**, las mejoras planteadas y las mejoras que se nos han ocurrido a nosotros, pasamos a listar los requerimientos que hemos seleccionado para nuestro programa Paint, para a partir de aquí, tenerlos en cuenta en el diseño e implementación:

**- Req. 1:** La zona de dibujo (Lienzo) será una subclase de QWidget que gestionará los eventos del ratón y de paint.

**- Req. 2:** Incluir una barra de menús que contenga fichero. Lo ampliamos en un requerimiento posterior.

**- Req. 3:** Se debe poder dibujar líneas y rectángulos.

Los requerimientos funcionales que añadimos, tomados de la **parte opcional** son:

**- Req. 4:**"Dibujo elástico" para que se vaya viendo en todo momento cómo va a quedar la figura cuando se suelte el botón del ratón.

**- Req. 5:** Se podrá dibujar Elipses, Polilíneas y Texto (son dos puntos opcionales, pero los englobamos en uno).

**- Req. 6:** Se podrá deshacar y rehacer.

**- Req. 7:** Se podrán seleccionar ciertas características, no sólo de la línea, sino de todas las figuras que se pueden dibujar. Esto lo explicamos un poco más adelante.

**- Req. 8:** Guardar y cargar en un fichero.

**- Req. 9:** Se incluirá en el menú Opciones, opciones de dibujo, como el color.

**- Req. 10:** Usaremos diálogos para guardar y cargar ficheros.

Los requerimientos funcionales **que se nos han ocurrido** y que vamos a tener en cuenta para el diseño e implementación son:

**- Req. 11:** A parte de guardar y cargar, se añadirá la opción de crear nuevo dibujo.

**- Req. 12:** Para cada figura se podrán elegir unas opciones u otras, por lo que especificamos como requerimiento qué opciones se podrán elegir para cada una:

Para la **línea**:

1) Color del trazo.
2) Tipo del trazo (punto, guiones, guión-punto, guión-punto-punto).
3) Grosor del trazo.

Para la **elipse**:

1) Color del borde.
2) Tipo del borde (punto, guiones, guión-punto, guión-punto-punto).
3) Grosor del borde.
4) Color de relleno.

Para la **polilínea**:

1) Color del trazo.
2) Tipo del trazo (punto, guiones, guión-punto, guión-punto-punto).
3) Grosor del trazo.

Para el **rectángulo**:

1) Color del borde.
2) Tipo del borde (punto, guiones, guión-punto, guión-punto-punto).
3) Grosor del borde.
4) Color de relleno.

Para el **texto**:

1) Color del texto.

**- Req. 13:** Uso de diálogos de Qt para selección de color.

**- Req. 14:** Uso de toolbars para las opciones Color de relleno, Color de borde, Tipo de trazo, Grosor de trazo, Deshacer, Rehacer, Abrir, Guardar, y Nuevo.

**- Req. 15:** Tener en cuenta si los últimos cambios en el dibujo fueron guardados cuando se

vaya a seleccionar "Nuevo dibujo", "Cargar dibujo" y "Salir". Utilizar **ventanas de confirmación** cuando sea necesario.

**- Req. 16:** Uso de atajos de teclado.

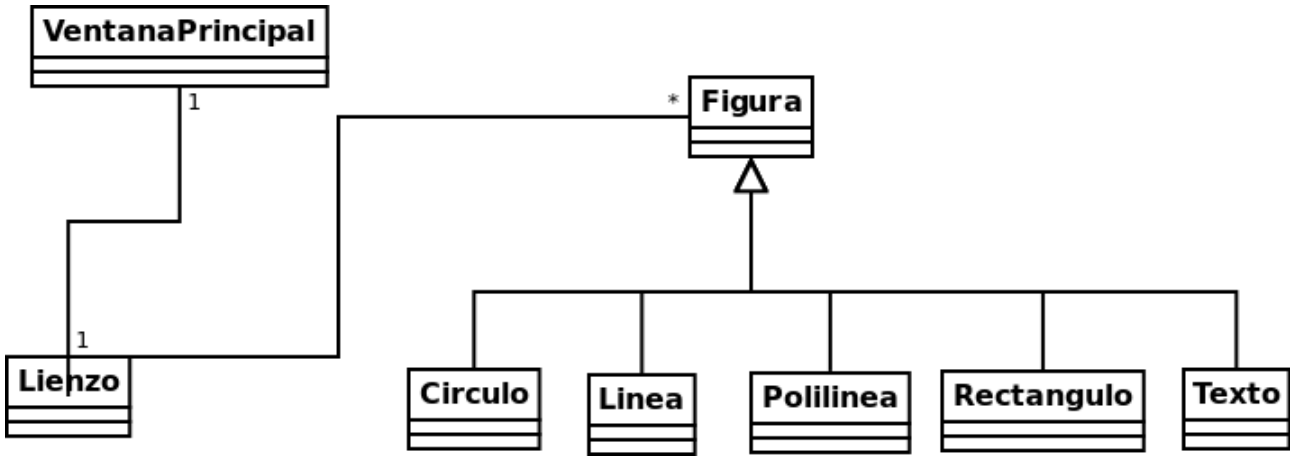
**- Req. 17:** Añadir un "Acerca de" que suele estar en todos los programas, usando una **ventana de información**.

### 2.2. Requerimientos no funcionales

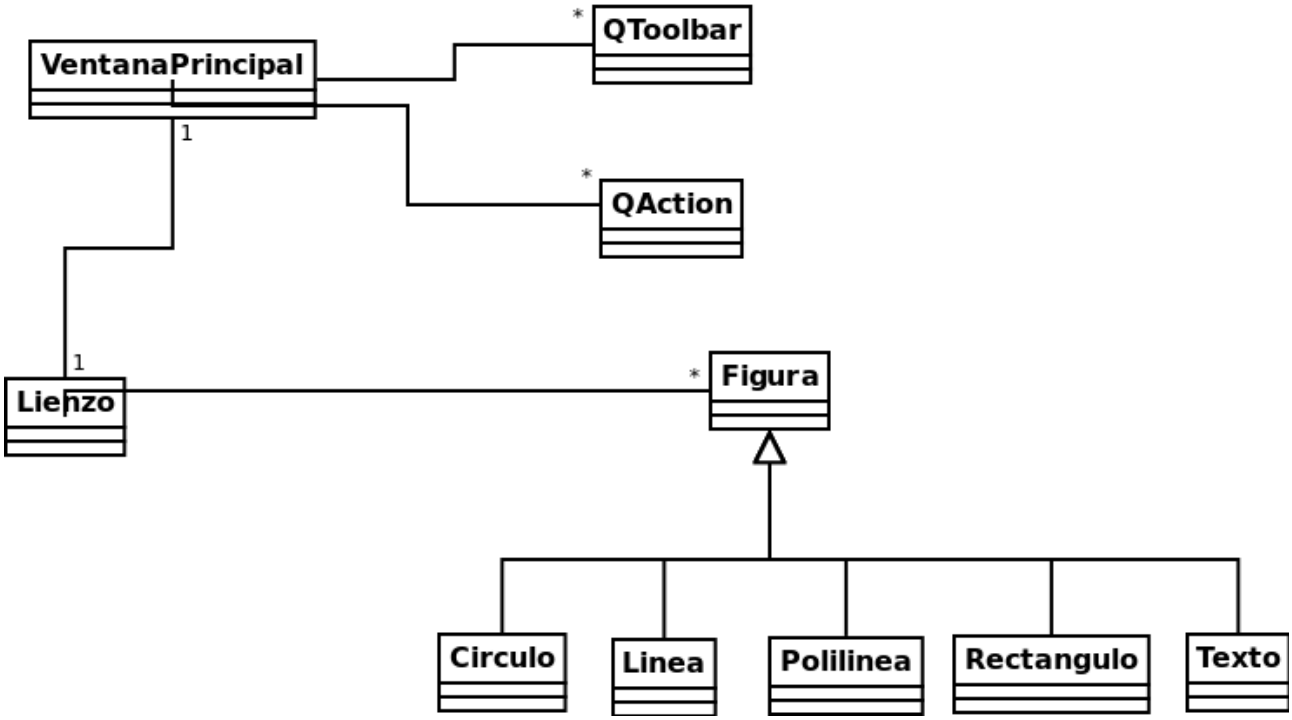
**- Req. 1:** Las figuras deberán tener independencia de la interfaz, es decir, que deberán tener la autonomía suficiente para poder pintarse en distintos widgets. Así, en el caso de que quiera pintarse la misma figura en varios widgets y que se modifiquen por igual, deberá ser posible sin tener que modificar el código de las figuras.

## 3. Diagrama de clases conceptual

Partiendo de los requerimientos, y sacando los conceptos que van a tener una entidad propia en nuestro diseño, realizamos un primer diagrama de clases conceptual:



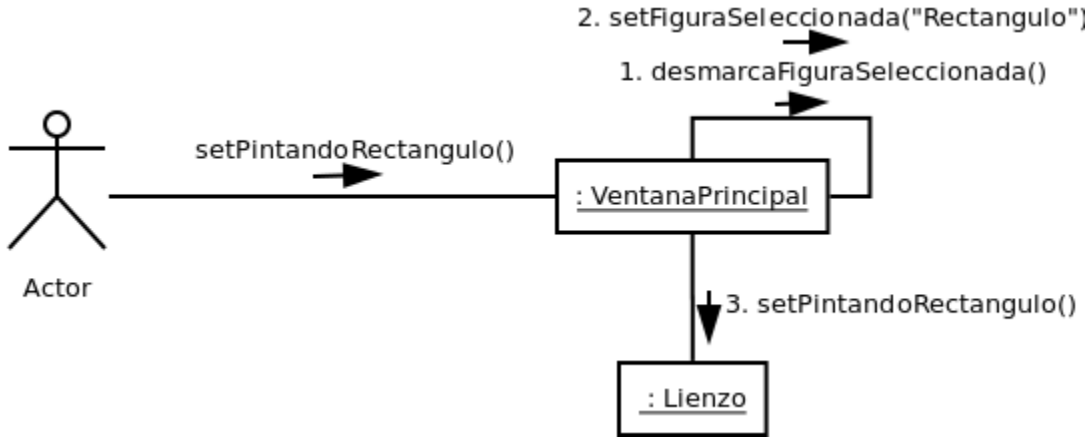

Una vez planteado este primer diagrama conceptual, vemos que los elementos de la interfaz estarán asociados a VentanaPrincipal, por lo que si añadimos esto al diagrama para dejarlo un poco más representativo tenemos:




## 4. Mecanismo de los métodos más importantes

En la ventana principal sólo se gestionarán los evento relacionados con los botones de los menú y de los toolbars, mientras que la parte importante estará en el lienzo. Será ahí donde deberemos gestionar los eventos pressed, move y released.

Para exponer cómo funcionarían un par de métodos. Por un lado, un método de la ventana principal será el que se invoque cuando se active un botón de dibujo distinto distinto. Así, si se produce la señal triggered del botón de dibujo **rectángulo**, se llamará a un slot de la ventana principal que sea **setPintandoRectangulo()**. En el diagrama de colaboración se ve qué hará este método:

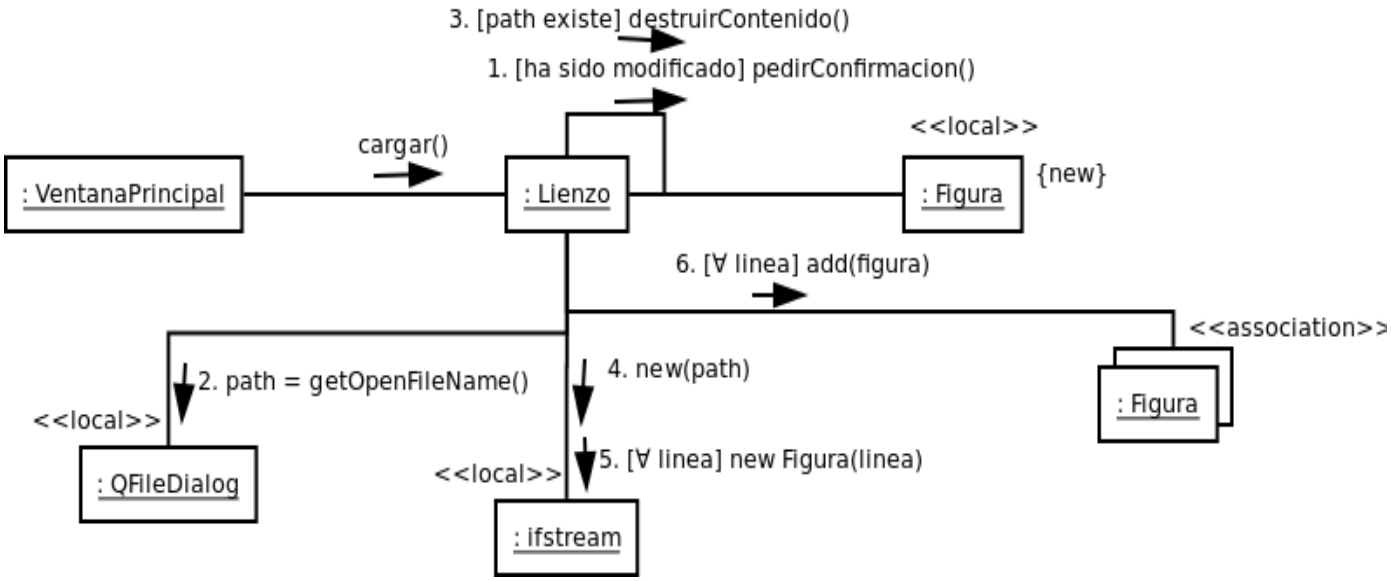



Como nota aclaratoria, veremos más adelante que el lienzo necesita de alguna forma conocer

qué opción de dibujo está activada en los botones, ya que de eso dependerá que gestione los eventos de una forma u otra. Como según el diseño que vamos a aplicar, éstos están fuera del lienzo, y es la ventana principal la que tiene acceso a ellos, realizamos el tercer paso de indicarle que eso ha cambiado, y en este caso concreto, debe pasar a gestionar los eventos del ratón teniendo en cuenta que se tiene seleccionada la opción d epintar un rectángulo.

Por otro lado, los botones para **guardar, abrir y crear un nuevo dibujo**, no estarán asociados con ningun método de la ventana principal, aunque sea ésta la que esté asociada con dichos botones. La razón es que, si lo conectáramos con métodos de la ventana principal, éstos no disponen de la información que se encuentro dentro del lienzo, y tendrían que utilizar métodos del lienzo para desarrollar la tarea. Para realizar un código más limpio y compacto, nos saltamos ese paso, y conectamos estos botones directamente con los métodos del lienzo que realicen dicha acción.

Veamos cómo funcionaría, por ejemplo, el método **cargar()** del lienzo:




Nótese que por simplificar, se ha indicado que se construye un objeto Figura con una linea (string) como parámetro. Para tal efecto, habría que comprobar el formato del string que representa esa figura, y crear un objeto de alguna de sus subclases. Esto se conseguiría realizar quitando solapamiento si utilizáramos un "Factory Pattern". Como al revisar los requerimientos, vemos que este es el único lugar en el que deberemos crear figuras de esta forma, lo traduciremos en un simple if en su lugar.

Comentaremos más adelante cómo hemos definido estos strings representativos de figuras.

Otros métodos interesantes a comentar son los gestores de eventos PressEvent, MoveEvent y ReleaseEvent. El mecansimo que seguiremos se divide en varios pasos:

**1) PressEvent:** dependiendo del tipo de dibujo seleccionado en los botones crearemos una figura u otra con la información del event. Esto lo guardamos en un atributo que represente la figura que se está pintando actualmente.

**2) MoveEvent:** dependiendo del tipo de dibujo seleccionado en los botones, modificamos la figura pintándose actualmente con la nueva información proporcionada por el event.

**3) ReleaseEvent:** hacemos lo mismo que en el MoveEvent, y añadimos la figura que hemos pintado al vector de figuras que componen el dibujo. También hay que preparar donde vaya a ir la siguiente figura para que no haya ningún error de ejecución.

Como se puede ver, este procedimiento es simple, y hace que el método PaintEvent() no se componga sólo de un bucle que pinte el vector de figuras en el orden correspondiente, sino que también, en el caso de estar pintándose una figura nueva, habrá que pintarla, ya que ésta no se encontrará en el vector hasta que no se produzca el ReleaseEvent.

*Cadenas representantes de figuras*

Con esto queda justificado el diseño de la implementación que procederemos a realizar, pero en última instancia, debemos definir un formato de cadena que represente unívocamente a una figura. Esto lo haremos de la siguiente forma.

Líneas: '**L'.X1.Y1.X2.Y2.R.G.B.grosor.estiloTrazo**
Rectángulos: '**R'.X1.Y1.X2.Y2.R.G.B.grosor.estiloBorde.(* | (Rrelleno.Grelleno.BRelleno))**
Elipses: '**E'.centroX.centroY.radioX.radioY.R.G.B.grosor.estiloBorde.(* | (Rrelleno.Grelleno.BRelleno))**
Polilíneas: **'P'.R.G.B.grosor.estiloTrazo.{X.Y}(.X.Y)***
Texto: **'T'.X.Y.R.G.B.(?)***

Como se puede apreciar, con el primer carácter se identifica de qué tipo de figura se trata, y el resto de campos se separan por puntos. Además, en las figuras que pueden o no tener color de relleno, la ausencia de éste se indica con un asterisco.
En las polilíneas, al haber un número indefinido de puntos que la representan, éstos se dejan al final de todo.
En el texto sucede algo similar, ya que la longitud de la cadena es indefinida.

## 5. Procedimiento de desarrollo con QT Designer

Para diseñar este software, sólo vamos a utilizar QT Designer como ayuda para el diseño de la interfaz de las ventanas que necesitemos usar. El resto del código, lo realizaremos a mano usando un editor de texto simple (gedit).

*Así pues, con QT Designer, generamos ficheros con extensión .ui, un por cada ventana que necesitemos usar. Éstos contendrá código xml que describe la interfaz de la ventana correspondiente, y con la ayuda de la herramienta uic* los convertiremos al código correspondiente en C++. Aún así, esto se realiza internamente, de forma que *qmake* ya hace uso de uic para hacer esa conversión y a la vez genera el Makefile necesario para la compilación.

## 6. Compilación y ejecución del programa y notas adicionales

Para facilitar la compilación (aunque son tres comandos), hemos facilitado un script en bash con nombre compilar.sh. Para compilar y ejecutar, hará falta únicamente seguir dos pasos:

1) Ejecutar el script de compilación: **./compilar.sh**
2) Ejecutar el binario resultante: **./practica2**

En este documento hemos recogido los puntos más importantes en lo que respecta a la planficiación que seguimos en el desarrollo, pero como documentación adicional, hemos generado el directorio **html/** con la herramienta doxygen. Para consultar dicha documentación, es necesario abrir el archivo **index.html** que se encuentra dentro con una navegador como Firefox, Opera o Chrome.

## 7. Manual de usuario

A continuación aclaraciones sobre los botones que aparecen en la interfaz, siguiendo los números de la siguiente captura.


**1:** Crear un dibujo nuevo.
**2:** Guardar el dibujo actual.
**3:** Abrir un dibujo guardado anteriormente. Sólo se admiten dibujos guardados con este programa. Los formatos habituales jpg, png, gif, etc no se pueden abrir.
**4:** Deshacer el último cambio aplicado al dibujo. Esta acción se puede realizar tantas veces se desee hasta lelgar al principio. Incluso habiendo abierto un dibujo que ya estuviese guardado de antes, se podrá aplicar esta operación de deshacer.
**5:** Rehacer el último cambio deshecho. Esto se podrá hacer siempre que queden movimientos deshechos. Si de deshacen modificaciones, y se dibuja algo, no se podrá rehacer lo anterior.

**6 - 9:** Tipos de figuras a dibujar. Antes de dibujar las figuras hay que seleccionar las opciones que se deseen (color, trazo, etc), ya que no se podrán editar una vez dibujadas.
**6:** Pintar líneas rectas.
**7:** Pintar rectángulos.
**8:** Pintar elipses.
**9:** Pintar polilíneas.
**10:** Pintar texto.

**11:** Color del borde en las figuras, y del texto.
**12:** Color del relleno, en el caso de que se pinte alguna figura que tenga relleno (rectángulos o elipses).
**13:** Este botón sirve para eliminar el color del relleno que pudiera haber. No lo pone en blanco, sino que lo elimina.

**14:** Grosor del borde en las figuras o del texto.

**15 - 19:** Tipo de trazo que se aplicará en las figuras. En el texto no será aplicado.

**15:** Línea sólida.
**16:** Línea de puntos.
**17:** Línea de barras o guiones.
**18:** Línea con el patrón guión-punto.
**19:** Línea con el patrón guión-punto-punto.