

Interfaces gráficas de usuario y Qt

Dr. J.B. Hayet

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS

Octubre 2009



Outline

1 Interfaces gráficas de usuario

2 La librería Qt



Outline

1 Interfaces gráficas de usuario

2 La librería Qt



GUIs

Las **Interfaces Gráficas de Usuario** (IGU o GUI) son interfaces para usuarios dentro de programas informáticos basadas en elementos gráficos básicos (los *widgets*) que permiten al usuario expresar peticiones, manipular datos, controlar el funcionamiento del programa... en general **interactuar con el programa**.

Se pueden ver como una forma particular de **Interfaz Hombre Maquina** (que puede incluir formas de interacción más diversas: sonido, sensores hápticos...)



GUIs

- + Elemento fundamental en la mayoría de las aplicaciones hoy para **mejorar la productividad** al usar programas informáticos
 - Facilidad de uso, ejecución acelerada.
 - Aprendizaje.
 - Riesgos de errores disminuidos.
 - Visibilidad y claridad de las funcionalidades.
- **Inversión de tiempo** elevada para algo no tan fundamental.



GUIs

Widgets típicos:

- ventanas,
- menús,
- botones, botones “radio”, *check boxes*,
- iconos,
- zonas de texto, listas.

Se usa generalmente el acrónimo **WIMP** (Windows, Icons, Menus, Pointing devices).



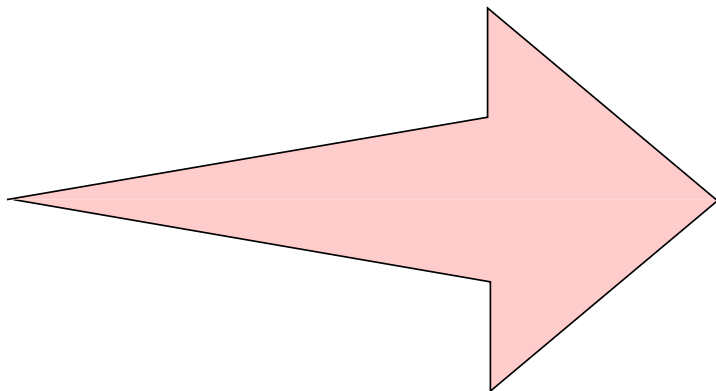
GUIs

A la frontera de:

- **programación**: necesita un buen diseño de la arquitectura del software,
- **psicología y ergonomía**: un factor muy importante para que el usuario lo use fácilmente y sin ambigüedad,
 - Caso extremo: *cockpits* de los aviones.
 - Caso de la catástrofe del Mont Saint-Odile (ergonomía del modo bajada).
- **diseño gráfico**: artes gráficos, tipografía. . .



Interacción



Shells

Ventanas

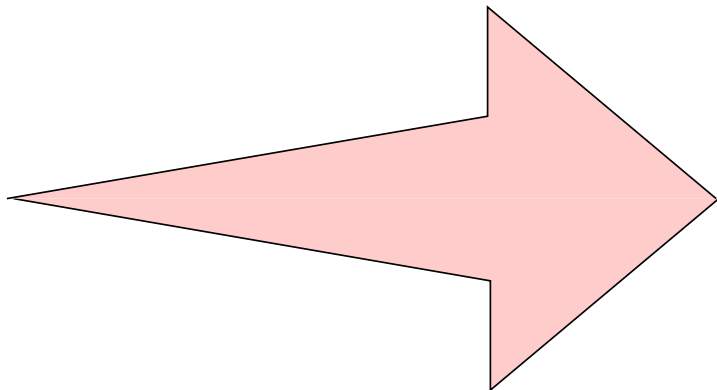
WIMP

Realidad virtual
o aumentada

Shells: requiere un nivel de “experto”...



Interacción



Shells

Ventanas

WIMP

Realidad virtual
o aumentada

Shells: requiere un nivel de “experto”...



Interacción

- Manipulación **indirecta**: menús, formularios, botones de tipo radio. . .
 - interacción limitada, control de datos a posteriori.
- Manipulación **directa**: los objetos de datos están manipulados directamente por sus representaciones gráficas (noción de metáfora)
 - “desktop”
 - WYSIWYG



Diseño de GUIs

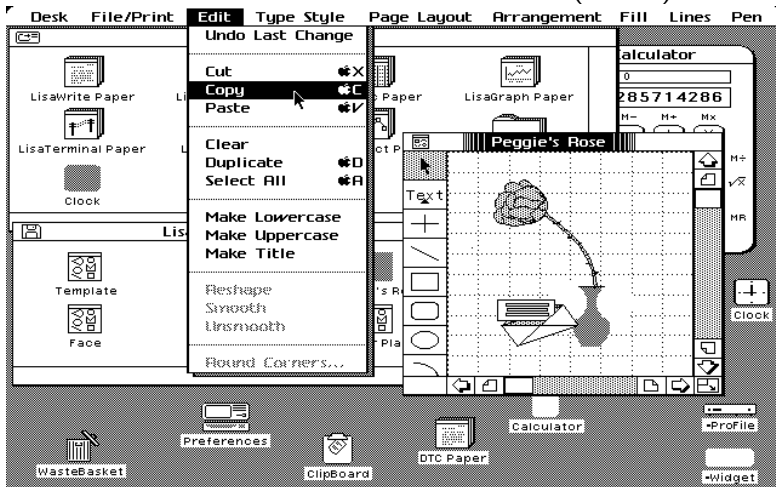
Widget toolkits, unos ejemplos:

- Java: Swing
- C++: Qt, Borland, wxWidgets
- C: Motif, GTK
- Tk



GUIs

WIMP: esquema ya bastante viejo sin evoluciones profundas desde los primeros sistemas de GUI desde los años 70 (PARC)



GUIs

Ahora, nuevos conceptos:

- Interacción con otros sensores y otras pantallas.
- **ZUI** (Zoomable User Interface): sin ventanas, sino con varios niveles de zoom.
- Interfaces “**tridimensionales**”: Compiz en Linux, Spaces en MacOSX. . .
- Interfaces “**realistas**” con motores físicos: **BumpTop**



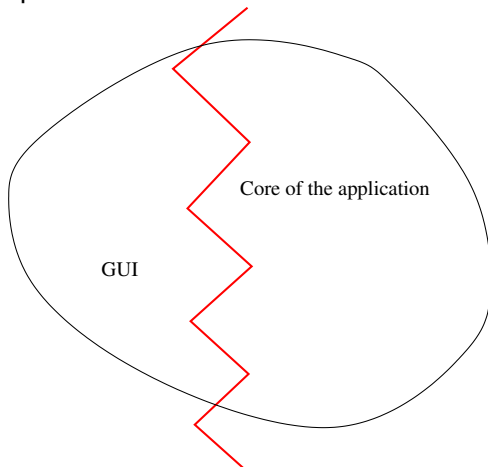
Diseño de GUIs

- Concepción un poco diferente de lo usual: el **comportamiento del usuario** está primordial.
- **Difícil de integrar** los factores humanos.
- Método clásico poco adaptado:
análisis → concepción → implementación → tests
- En general, la concepción de GUI es **iterativa**, con una **evaluación (feedback) del usuario** para re-empezar el análisis del problema.



Diseño de GUIs

El punto más importante:



Separar, si posible, la parte GUI de la parte principal de la aplicación



Diseño de GUIs: modelo MVC

Modelo propuesto en el PARC en 1979: **Model, View, Controller**:

- **Model**: los datos, y las herramientas para manipularlas.
- **View**: la manera de desplegar esos datos.
- **Controller**: el manejo de los inputs del usuario y la manera de que la interfaz reacciona a esos inputs.

Modelo adoptado en Swing (Java) y posible que usar en Qt4.



Diseño de GUIs

Por naturaleza, la parte GUI del programa debe de reaccionar a **eventos asíncronos**, que pueden afectar varias componentes del sistema:

- sistema de comunicación por mensajes,
- particularmente adaptado a lenguajes OO (Java, C++, C#),
- mensajes por llamada de método,
- organización modular y jerarquizada,
- arboles de *widgets*.



Diseño de GUIs

Típicamente, **programación por eventos**, donde el programa se pone a la “escucha” del usuario:

```
while (true) {  
    event = getNextEvent();  
    processEvent(event);  
    ...  
}
```

- A cada acción del usuario corresponde un evento enviado a la aplicación (ratón, teclado, manipulación de ventana...)
- Evento = Tipo de evento + datos (ex: posición del cursor)



Diseño de GUIs

El cuerpo de una GUI:

- 1 Crear, posicionar, dibujar los **widgets principales**
- 2 Lanzar el **ciclo de manejo de los eventos**

El ciclo recupera los eventos y llama las funciones adecuadas del programa, con dos implementaciones:

- protocolo no embarcado,
- protocolo embarcado.



Diseño de GUIs: protocolo no embarcado

El programador escribe el ciclo de escucha:

- **switch múltiples** sobre los tipos de eventos
- pero bien complicado: puede haber combinaciones de eventos implicando varios *widgets*
- muy complejo



Diseño de GUIs: protocolo embarcado

- control de la interfaz **embarcado dentro de los *widgets***
- ciclo de escucha proveído,
- **paradigma OO**: objeto gráfico reaccionando de una manera autónoma,
- funciones ***callback*** asociadas a los eventos en cada objeto,
- así está en la mayoría de los *Toolkits* actuales.



Diseño de GUIs

Cuidado que a priori el ciclo “infinito” es secuencial, y que las funciones de callback no pueden monopolizar toda la CPU (se necesita en particular **refrescar muy regularmente la apariencia gráfica de los *widgets***).

Caso en que son muy útiles **mecanismos de paralelización**: procesos o *threads* (pero con problemas de sincronización; de soporte, de notificación de la aplicación a la GUI).

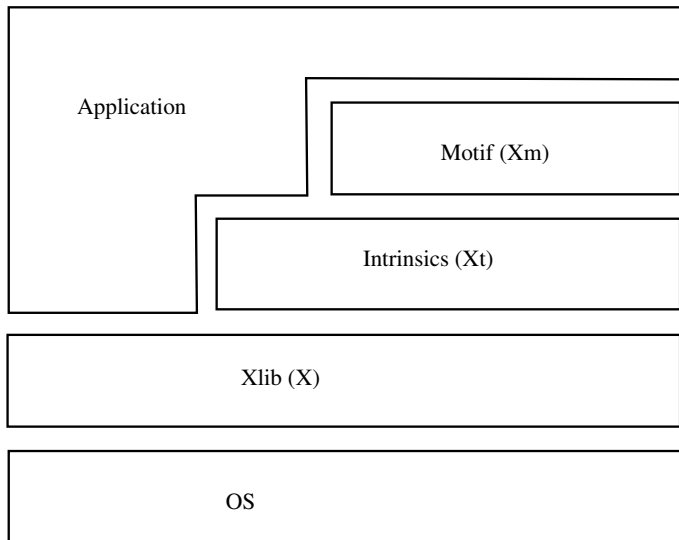


Diseño de GUIs: Motif

- **X**: sistema de manejo de ventanas.
- API para aplicaciones: Xlib pero de muy bajo nivel.
- Toolkits de mas alto nivel: Xt y Xm (**Motif**).
- Ya viejito, con **apariencia anticuada**.



Diseño de GUIs: Motif



Diseño de GUIs: Motif

El main:

```
Widget top_wid , button ;
XtAppContext app ;
void pushed_fn () ;
top_wid = XtVaAppInitialize(&app , "Push" , NULL , 0 ,
                           &argc , argv , NULL , NULL ) ;
button = XmCreatePushButton (top_wid , "Push_me" ,
                             NULL , 0) ;

/* Manage button */
XtManageChild (button) ;
/* Attach a callback */
XtAddCallback (button , XmNactivateCallback ,
               activateCB , NULL) ;
/* Display the main window and all its children */
XtRealizeWidget (top_wid) ;
/* Main loop */
XtAppMainLoop (app) ;
```



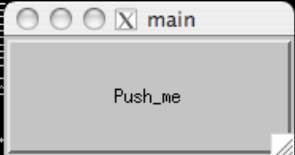
Diseño de GUIs: Motif

Función de *callback*:

```
void activateCB (Widget w, XtPointer client_data ,  
                  XtPointer cbs) {  
    printf ("Don't Push Me!!\n" );  
}
```



Ejemplo Motif



```

init.csh;xfig
xterm
macaye[~][20:07]>cd opt/CLASE-PROGRAMACION/
CLASE1/  CLASE13/  CLASE17/  CLASE22/  CLASE4/  CLASE9/
CLASE10/  CLASE14/  CLASE18/  CLASE23/  CLASE5/  EXAM/
CLASE11/  CLASE15/  CLASE19/  CLASE24/  CLASE7/  PREPROJET/
CLASE12/  CLASE16/  CLASE2/   CLASE3/   CLASE8/  PROJET/
macaye[~][20:07]>cd opt/CLASE-PROGRAMACION/CL
CLASE1/  CLASE12/  CLASE15/  CLASE18/  CLASE
CLASE10/  CLASE13/  CLASE16/  CLASE19/  CLASE
CLASE11/  CLASE14/  CLASE17/  CLASE2/   CLASE
macaye[~][20:07]>cd opt/CLASE-PROGRAMACION/CL
/Users/jbhayet/opt/CLASE-PROGRAMACION/CL
macaye[CLASE24][20:07]>ls
CLASE24.app      Makefile        example1.
CLASE24.pro      Motif           main.cpp
macaye[CLASE24][20:07]>cd Motif/
/Users/jbhayet/opt/CLASE-PROGRAMACION/CLASE24/Motif
macaye[Motif][20:07]>./main
Don't Push Me!!
Don't Push Me!!
Don't Push Me!!
Don't Push Me!!
Don't Push Me!!

```



Unos criterios para buenos GUIs

- **Guiar** el usuario:
 - **incitar** a hacer unas acciones y no otras: menús desactivados, desplegar formato
 - **localización espacial**: agrupamiento espaciales o de estilo
 - **feedback**: que el usuario vea lo que esta haciendo (*passwords*)
 - legibilidad
- **Aliviar** la cantidad de datos
 - Brevedad, concisión global
 - Aliviar densidad (percepción)
- **Controlar**: validación de las acciones, proponer seguir/abortar operaciones.



Unos criterios para buenos GUIs

- **Flexibilizar**: que el usuario pueda eventualmente cambiar la apariencia del GUI.
- **Prever usuarios experimentados**: *shortcuts*.
- **Proteger** contra errores (con confirmaciones).
- **Desplegar mensajes** de error claros.
- **Proponer mecanismos para corregir** errores (ex: anular).
- **Homogeneizar** la organización dentro de un contexto dado.
- **Usar semántica** (metáforas en iconos. . .)



Outline

1 Interfaces gráficas de usuario

2 La librería Qt



Qt

- *Toolkit* gráfico en C++.
- Muchas extensiones y herramientas de desarrollo (*Designer*).
- **Gratuito** (con licencia GPL) si uso no comercial, disponible en muchas plataformas (Linux, MacOSX, Windows).
- Base de KDE.
- Apoyado por (comprado. . .) Nokia.
- Versión 4.5
- Aplicaciones con Qt: Opera, Google Earth, Skype



Qt

Características:

- “Look and feel” variable, para parecerse a estilos ya existentes (Mac, Windows, Motif. . .)
- Internacionalización: QString y Unicode
- OpenGL multiplataformas.
- Parser XML integrado.
- Maneja bases de datos SQL.
- Herramienta de generación automática Qt Designer.



Qt: librerías

- **QtCore**: Clases básicas, no GUI, usadas por todos los módulos.
- **QtGui**: Componentes GUI (lo útil para gráficos)
- **QtNetwork**: Clases para programación de redes
- **QtOpenGL**: Clases para soporte OpenGL
- **QtSql**: Clases para integración de bases de datos SQL
- **QtSvg**: Clases para integración de dibujos vectoriales SVG



Qt: librerías

- **QtXml**: Clases para manejar XML.
- **QtDesigner**: API para Qt Designer.
- **QtUiTools**: Clases para manejar los formularios .ui del Designer.
- **QtAssistant**: Soporte para línea online.
- **Qt3Support**: Clases para compatibilidad Qt3.



Qt: clases

Muchos de los objetos heredan de la clase `QObject`, en particular:

- `QWidget`: clase de todos los *widgets*, de botones a sliders. . .
- `QCoreApplication` y `QApplication`, que manejan el ciclo de escucha de los eventos, provee la inicialización y terminación de la aplicación, y todos los reglajes hechos al nivel de la aplicación.

```
QApplication::setStyle(new QWindowsStyle);
```

- `QAction`: que permite especificar una abstracción de acción dada (a un menú o a una toolbar).



Qt: clases

Ejemplo de QAction:

```
openAct = new QAction(QIcon(":/images/open.png"),  
                      tr("&Open..."), this);  
openAct->setShortcut(tr("Ctrl+O"));  
openAct->setStatusTip(tr("Open an existing file"));  
connect(openAct, SIGNAL(triggered()),  
        this, SLOT(open()));  
fileMenu->addAction(openAct);  
fileToolBar->addAction(openAct);
```



Una aplicación

Ejemplo básico:

```
#include <QApplication>
#include <QLabel>

int main( int argc , char **argv ) {
    QApplication* app = new QApplication(argc , argv);
    QLabel* hello =
        new QLabel("<font_color=blue>Hello_Qt</font>" ,
                    0);
    hello->show();
    return app->exec();
}
```



Qt: hijos

Constructores sobrecargados para los QObject:

```
QAction ( QObject * parent );  
QAction ( const QString & text , QObject * parent );  
QAction ( const QIcon & icon , const QString & text ,  
          QObject * parent );
```

Se especifica el **padre en la jerarquía gráfica** como ultimo argumento. Si NULL o 0, se trata de un *widget* de nivel mas arriba (estructura de árbol).



Qt: hijos

```
int main(int argc, char *argv[]) {  
    QApplication app(argc, argv);  
    QWidget window;  
    window.resize(200, 120);  
  
    QPushButton quit(" Quit", &window);  
    quit.setFont(QFont(" Times", 18, QFont::Bold));  
    quit.setGeometry(10, 40, 180, 40);  
    QObject::connect(&quit, SIGNAL(clicked()),  
                    &app, SLOT(quit()));  
  
    window.show();  
    return app.exec();  
}
```

Una aplicación con una ventana y un botón adentro.



Qt: hijos

- Un *widget* hijo está desplegado dentro del objeto padre.
- Eventualmente está recortado, ubicado por default a la esquina **top-left** del padre.
- Se modifica eventualmente la ubicación pero en **coordenadas relativas** (x_{tl}, y_{tl}, w, h) .

```
quit.setGeometry(10, 40, 180, 40);
```

- Cuando un objeto padre está desplegado (por `show()`), todos los hijos lo están también.

```
window.show();
```

- Se puede **negar explícitamente de desplegar** con el método `hide()`.



Qt: creación de su propio widget

Usar los **mecanismos de herencia** de C++:

```
class MyWidget : public QWidget {  
    public:  
        MyWidget(QWidget *parent = 0);  
};  
  
MyWidget::MyWidget(QWidget *parent) : QWidget(parent)  
    setFixedSize(200, 120);  
    QPushButton *quit =  
        new QPushButton(tr("Quit"), this);  
    quit->setGeometry(62, 40, 75, 30);  
    quit->setFont(QFont("Times", 18, QFont::Bold));  
    connect(quit, SIGNAL(clicked()),  
        qApp, SLOT(quit()));  
}
```



Qt: creación de su propio widget

Luego usar ese nuevo *widget* como los otros:

```
int main(int argc, char *argv[]) {  
    QApplication app(argc, argv);  
    MyWidget widget;  
    widget.show();  
    return app.exec();  
}
```



Qt: creación de su propio widget

- El boton quit es un hijo construido por aloca  n dinamica; no se guarda huella de el explicitamente pero Qt s  .
- En particular, se va a encargar de **liberar la memoria asociada a este objeto** cuando el objeto papa (widget) estar   destruido.
- Consecuencia: no es necesario hacer destructores que se encarguen de la destrucci  n de los *widgets* locales



Qt: creación de su propio widget

Una propiedad muy interesante: al usar cadenas de caracteres (como para el botón), se puede usar cadenas normales o cadenas modificadas por la función estática `QWidget::tr()`

El papel de esta función: prever que esta cadena puede estar luego traducida automáticamente (en el momento de la ejecución) en otra lengua, con un archivo de traducción.



Qt: señales y slots

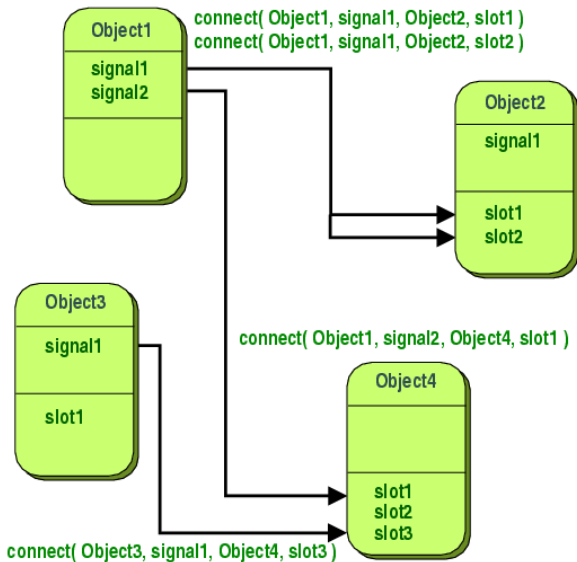
En cada objeto heredando de `QObject`, se puede especificar que el objeto puede emitir/recibir señales (además de los señales que pueden manejar por default):

- **señales**: métodos especiales que corresponden a la llamada del señal,
- **slots**: funciones miembros llamadas automáticamente si “conectada” a una señal emitida,

```
connect( button , SIGNAL(clicked()) ,  
        app , SLOT(quit()) );
```



Qt: señales y slots



Qt: señales y slots

```
#include <QApplication>
```

```
#include <QPushButton>
```

```
#include <QFont>
```

```
int main( int argc , char **argv ) {  
    QApplication a(argc , argv);  
    QPushButton quit("Quit" , 0);  
    quit.resize(75 , 30);  
    quit.setFont(QFont("Times" , 18 , QFont::Bold));  
    QObject::connect(&quit , SIGNAL(clicked()) ,  
                    &a , SLOT(quit()));  
    quit.show();  
    return a.exec();  
}
```



Qt: señales y slots

Leer en la documentación que tipo, ejemplo con QAbstractSlider:

Public Slots

- * **void** setOrientation (Qt::Orientation)
- * **void** setValue (**int**)
- * 17 public slots inherited from QWidget
- * 1 public slot inherited from QObject

Signals

- * **void** actionTriggered (**int** action)
- * **void** rangeChanged (**int** min, **int** max)
- * **void** sliderMoved (**int** value)
- * **void** sliderPressed ()
- * **void** sliderReleased ()
- * **void** valueChanged (**int** value)
- * 1 signal inherited from QWidget
- * 1 signal inherited from QObject



Qt: señales y slots

- mecanismo propio (y emblemático) de Qt,
- en otros Toolkits: mecanismos de callbacks que muchas veces son 1 por objeto,
- SLOT y SIGNAL son macros del preprocesador,
- el mecanismo esta proveído gracias a una precompilación con **moc**,
- ...que genera archivos de código temporales:

```
macaye[qtClient][7:32]>ll *.cpp
-rw-r--r--  1 jbhayet  jbhayet   3798 Mar  5  2007 qtClientTrackThread.cpp
-rw-r--r--  1 jbhayet  jbhayet  33506 Mar  5  2007 qtClient.cpp
-rw-r--r--  1 jbhayet  jbhayet   6194 Mar  6  2007 moc_qtClient.cpp
-rw-r--r--  1 jbhayet  jbhayet   2123 Mar  6  2007 moc_qtClientTrackThread.cpp
```



Qt: señales y slots

Para compilar, se puede **usar qmake**:

```
qmake -project // Crea archivo dir.pro
qmake          // Crea Makefile
make           // Crea archivos moc_* y binarios
```



Qt: señales y slots

Crear señales y slots, en el .h:

```
class Person : public QObject {
    Q_OBJECT
private:
    int currentAge;
public:
    Person(unsigned int age) { currentAge = age; }
    int getAge() const { return currentAge; }
    void shareAge();

public slots:
    void setAge(int age) {currentAge=age;};
signals:
    void ageChanged(int newAge);
};
```



Qt: señales y slots

- El objeto tiene que heredar (directamente o indirectamente) de QObject.
- La palabra llave Q_OBJECT es necesaria para el moc.
- Los **señales no son implementados** (lo hace Qt).
- Los slots **son métodos** y tienen que ser implementados.



Qt: señales y slots

Crear señales y slots, en el .cpp:

```
void Person::shareAge() {  
    emit ageChanged(currentAge);  
};  
  
int main( int argc , char **argv ) {  
    QApplication a(argc , argv);  
    Person p1(20),p2(30);  
    QWidget::connect(&p1,SIGNAL(ageChanged(int)),  
                     &p2,SLOT(setAge(int)));  
  
    p1.shareAge();  
    std::cout << p2.getAge() << std::endl;  
    return a.exec();  
}
```

Preferir QApplication para aplicaciones sin GUI.



Qt: señales y slots

- **Palabra llave emit** para activar una señal: similitud con excepciones.
- Cuidado a ciclos infinitos (por ejemplo si el emit esta en el `setChange` !).
- El objeto no tiene que preocuparse de quien recibirá la señal que emite.
- Esta señal puede estar conectada a varios *slots*.
- La precompilación se encarga de resolver los problemas de nombres.



Qt: señales y slots

- Tipos de los señales y de los slots **deben de ser compatibles** (verificación en la ejecución)
- Pero un slot puede tener menos parametros

```
public slots:
    void setAge(int age) {currentAge=age;};
    void doSomething() { std::cout << "Do something"
                        << std::endl;};
```

y:

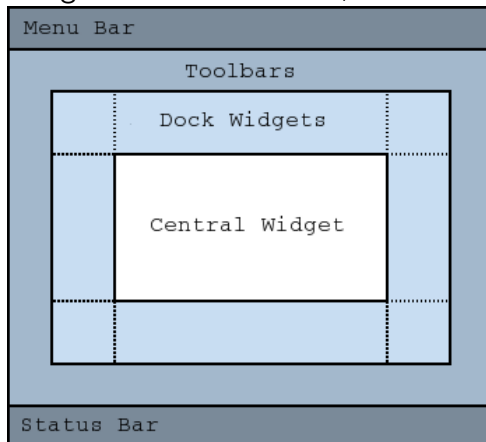
```
QWidget::connect(&p1, SIGNAL(ageChanged(int)),
                &p2, SLOT(doSomething()));
```

esta OK



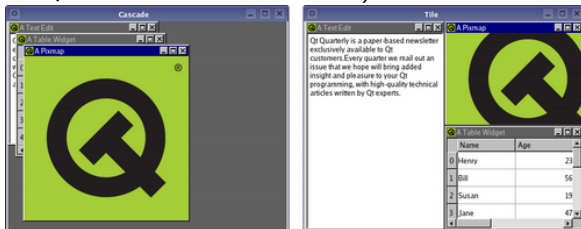
Qt: unos widgets

- QMainWindow **clase básica para manipular una ventana principal de aplicación**, con barra de menú, barras de herramientas, dock widgets barra de estatuto, con un widget central

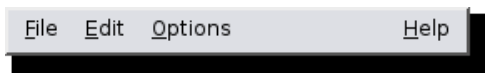


Qt: unos widgets

- QMdiArea: **espacio de trabajo** en que se puede haber varias ventanas (Multiple Document Interfaces)



- QMenuBar: **barra de menú**



Qt: unos widgets

- QToolBar: **barra de herramientas**, contenedor con un conjunto de widgets (botones, ...) o de acciones (QAction)



- QToolButton: **botones típicamente para toolbar**, que no enseña un texto sino un icono, y que puede desarrollar un menú
- QStatusBar: barra abajo de la aplicación, que puede **enseñar información sobre el estado corriente**
- QToolTip: **globos de ayuda** que aparecen para dar información sobre un widget
- QWhatsThis: descripción mas detallada (por ejemplo para el What's this de Windows)

```
newAct = new QAction(tr("&New"), this);
newAct->setStatusTip(tr("Create a new file"));
newAct->setWhatsThis(tr("Click this option to create a new file"));
```



Qt: unos widgets

- QAction: objetos compartidos por QMenuBar y QToolBar, especifican una **acción como elemento de un menú o botón** de un QToolBar.
 - Pueden estar creadas independientemente o añadidas con funciones especiales:

```
QAction* addAction(const QString & text);  
QAction* addAction(const QIcon & icon ,  
                  const QString & text);
```
 - Puede contener texto, icono, status...
 - Hacerlas hijos de la ventana en que viven (QMainWindow).



Qt: unos widgets

- QLabel: **despliega texto, sin posibilidad de interacción** con el usuario; puede incluir imágenes, videos
- QLineEdit: **edición de texto sobre una linea** (tamaño reducido)
- QTextEdit: *widget* de tipo **edición de texto**, con características avanzadas (soporte HTML)



Qt: unos widgets

Dialogos: QMessageBox, QFileDialog, QProgressDialog, usadas para **comunicaciones o tareas de corta duración con el usuario**; están todos top-level (especificarle un “padre” solo le centra sobre el “padre”)

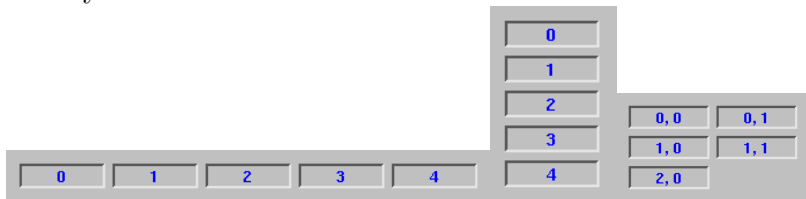
```
fileName = QFileDialog::getOpenFileName( this ,  
    tr( "Open_Image" ) , " /home/jbhayet" ,  
    tr( "Image_Files_( *.png_*.jpg_*.bmp )" ) );
```



Qt: layout

La organización espacial de los *widgets* dentro de otro *widget* puede estar **manejada automáticamente** por un **objeto de tipo QLayout**.

Clases derivadas de este son QHBoxLayout, QVBoxLayout, QGridLayout :



En este caso, no se necesita hacer explícitas las coordenadas relativas de los objetos; la aplicación les ubica correctamente; además, los objetos hijos están redimensionados si el objeto padre esta redimensionado.



Qt: layout

```
class MyWidget : public QWidget {  
    public:  
        MyWidget(QWidget *parent = 0);  
};  
MyWidget::MyWidget(QWidget *parent):QWidget(parent) {  
    QPushButton *quit = new QPushButton(tr("Quit"));  
    quit->setFont(QFont("Times", 18, QFont::Bold));  
  
    QLCDNumber *lcd = new QLCDNumber(2);  
    lcd->setSegmentStyle(QLCDNumber::Filled);  
  
    QSlider *slider = new QSlider(Qt::Horizontal);  
    slider->setRange(0, 99);  
    slider->setValue(0);  
    ...  
}
```



Qt: layout

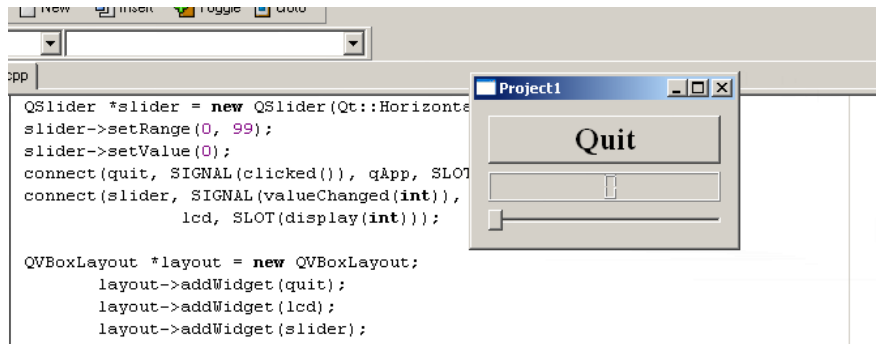
```
...
connect(quit , SIGNAL(clicked()), qApp, SLOT(quit()))
connect(slider , SIGNAL(valueChanged(int)),
        lcd , SLOT(display(int)));

QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(quit);
    layout->addWidget(lcd);
    layout->addWidget(slider);
    setLayout(layout);
}
```

Los *widgets* añadidos al *layout* se hacen automáticamente hijos del *widget* que llama el `setLayout`



Qt: layout



Qt: estilo

Qt **emula la apariencia de widgets de otros toolkits** (Motif, Windows)



El comportamiento se decide con:

```
QApplication::setStyle( new QMotifStyle );
```



Qt: gráficos

Manipulación de imágenes:

- Iconos:

```
QPushButton *quit = new QPushButton(tr("Quit"));  
quit->setIcon( QIcon("Winter.jpg") );
```

- Zona de dibujo: QPainter

```
QPainter painter( this );  
painter.setPen( black );  
painter.drawLine( 0, 0, 0, 99 );
```

- Dibujo OpenGL: OpenGL



Qt

Leer:

- Sitio de Qt: <http://qt.nokia.com/>
- Sitio de KDE: <http://www.kde.org/>
- Documentación:
<http://qt.nokia.com/doc/4.5/index.html>
- En Windows, compila bien con Dev-cpp,
nicolasj.developpez.com/articles/qt4/ (francés)
<http://vcg.isti.cnr.it/~ponchio/computergraphics/setup.html>
<http://darkhack.googlepages.com/qttutorial>

