FIEC

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

PROGRAMACIÓN APLICADA A LA AUTOMATIZACIÓN

M.Sc. Alexander Prieto León

Conferencia X

Unidad III Programación visual y orientada a eventos para la creación de HMI.

3.2 Manejo de eventos (continuación) y uso de elementos de ventana.

OBJETIVOS

Objetivos:

- Explicar los conceptos relativos al mecanismo Signal & Slot.
- Implementar el mecanismo Signal & Slot.
- Utilizar los elementos propios de ventana.
- Utilizar diálogos estándares.

BIBLIOGRAFÍA

Bibliografía:

- Zhi Eng, Lee; Rischpater, Ray. Application Development with Qt Creator: Build cross-platform applications and GUIs using Qt 5 and C++, 3rd Edition. (2020). Packt Publishing. ISBN-10: 1789951755, ISBN-13: 978-1789951752.
- http://www.qtrac.eu/C++-GUI-Programming-with-Qt-4-1st-ed.zip
- Qt 5.14.2 Reference Documentation. Qt Creator Help.
- https://doc.qt.io/qtcreator/index.html
- https://doc.qt.io/qt-5/qtdesigner-manual.html

- **En la clase anterior:**
  - **¿Qué son los eventos?**
  - **¿Cómo se nombra en Qt a la función manejadora de un evento?**

Los Eventos en Qt.

En Qt, los eventos son objetos, obtenidos de la clase abstracta **QEvent**, que representan las cosas que han ocurrido dentro de una aplicación o una actividad exterior de la que la aplicación tiene que estar al tanto. Los eventos pueden ser recibidos y manejados por cualquier instancia de una subclase de **QObject**, y estos son especialmente relevantes para los widget.

Función o Propiedad	Uso
objectName : QString	Para indicar el nombre del objeto.
bool **connect** (constQObject * sender, const char * signal, const char * method, Qt::ConnectionType type = Qt::AutoConnection) const	Para conectar una señal y un slot al objeto.
bool **disconnect** (const char * signal = 0, constQObject * receiver = 0, const char * method = 0)	Para desconectar una señal y un slot del objeto.
virtual void **timerEvent** (QTimerEvent * event)	Para manejar los eventos de tiempo.
Int **startTimer** (int interval)	Comienza el contador de tiempo de tiempo (Timer) para que se ejecute el evento del timer cada *interval* milisegundos.
virtual bool **event** (QEvent * e)	Para recibir los eventos de un objeto, devuelve *true* si el evento es reconocido y procesado.

Signal & Slot.

- Las señales y los slots son usadas para la comunicación entre los objetos. El mecanismo de señales y slots es la característica central de Qt y probablemente la que lo difiere de los restantes frameworks.

- **Signal**: es una notificación que un objeto emite cuándo cambia su estado de manera que podría interesarle a otros objetos.

- **Slot:** es una función que se ejecuta cuándo una señal se emite.

Signal & Slot.

- Los slots son generalmente idénticos a cualquier función miembro de C++. Ellos pueden ser virtuales; se pueden sobrecargar; pueden ser públicos, protegidos o privados, pueden ser directamente invocados como cualquier función de C++ y sus parámetros pueden ser de cualquier tipo.

- La diferencia es que un slot puede también ser conectado a una señal, caso en el cual será automáticamente llamado cada vez que la señal se emita.

Signal & Slot.

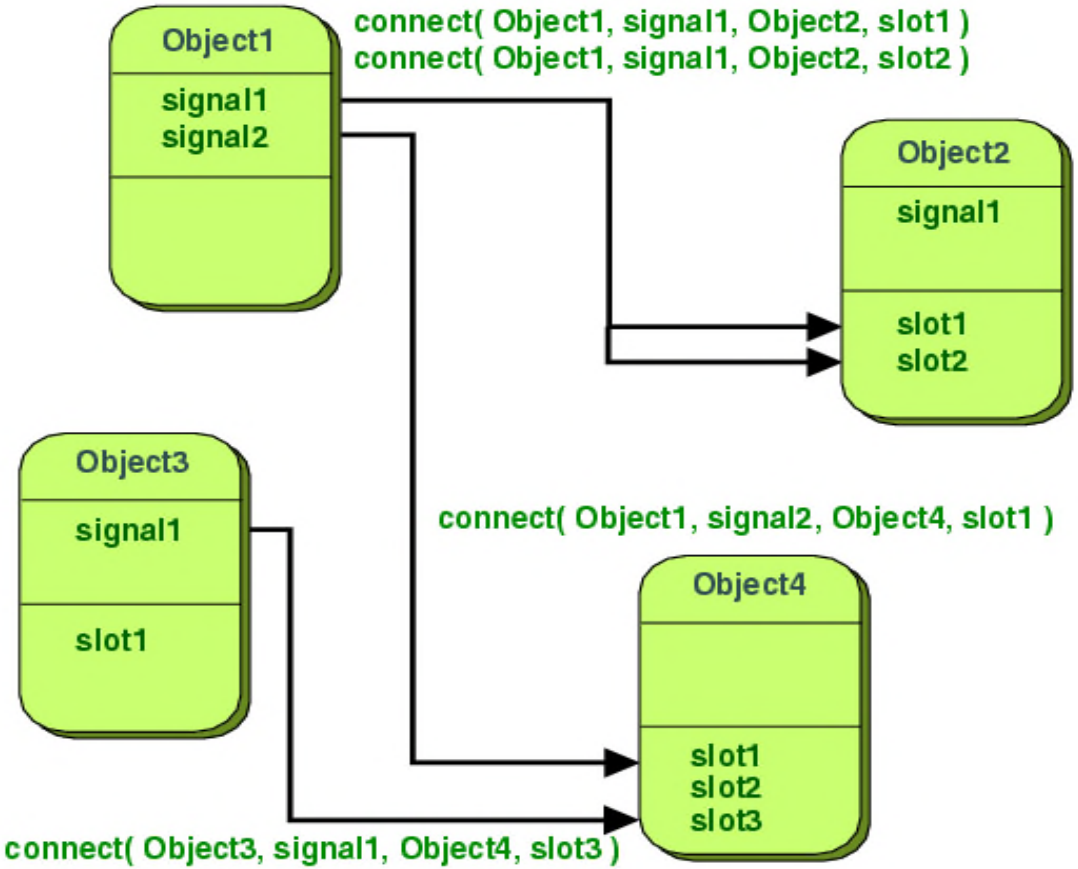
- Cualquier objeto en el que deseemos implementar señales y slots debe de heredar de la clase **Qobject.**

- Las señales y los slots que serán conectados entre sí deben tener la misma forma, es decir, el mismo tipo de retorno (siempre **void**) y los mismos argumentos a recibir.

- Se pueden conectar tantas señales como se quiera a un único slot, así como una señal pueda ser conectada a varios slot. Incluso es posible conectar una señal a otra y en este caso la segunda señal sería emitida inmediatamente después de la primera.

Signal & Slot.

- Comunicación entre objetos mediante el mecanismo signal & slot.



Implementando el mecanismo Signal & Slot.

- Ejemplo 1: Declaremos una pequeña clase de C++ que encapsula el comportamiento de una variable.

```
class Variable
{
public:
    Variable() { m_value = 0; }
    int value() const { return m_value; }
    void setValue(int value);
private:
    int m_value;
};
```

Implementando el mecanismo Signal & Slot.

- Ejemplo 1: implementemos la misma clase pero basada en la clase **QObject** de tal manera que se pueda usar el mecanismo de señales y slots.

```
#include <QObject>
class Variable : public QObject
{
    Q_OBJECT
public:
    Variable() { m_value = 0; }
    int value() const { return m_value; }
public slots:
    void setValue(int value);
signals:
    void valueChanged(int newValue);
private:
    int m_value;
};
```

```
void Variable::setValue(int value)
{
    if (value != m_value)
    {
            m_value = value;
            emit valueChanged(value);
    }
}
```

Implementando el mecanismo Signal & Slot.

- Ejemplo 1: Observaciones.
  - Esta última versión tiene la misma variable interna y provee métodos públicos para acceder a dicha variable, pero en adicción esta soporta el mecanismo de señales y slots. Esta clase le puede decir al mundo exterior que su estado ha cambiado emitiendo una señal, **valueChanged(),** y además tiene un slot al cual otros objetos pueden enviarle señales.
  - Todas las clases que quieran implementar el uso de las señales y los slots deben poner la declaración **Q_OBJECT** al inicio de su definición. Además de heredar directa o indirectamente de **QObject**.
  - El método **setValue()** es método de acceso y es slot a la vez, por lo que puede ser llamado no sólo por su objeto sino también por **signals** de otros objetos que se encuentren conectadas. En este caso la palabra **emit** emite la señal **valueChanged()** desde el objeto con el nuevo valor como argumento.

Conectando las señales y los slots.

- En el siguiente segmento de código se crean dos objetos de tipo Variable y se conecta la señal **valueChanged()** del primero con el slot **setValue()** del segundo haciendo uso de la función **QObject::connect().**

```
01 Variable a, b;
02 QObject::connect(&a, SIGNAL(valueChanged(int)),
03                         &b, SLOT(setValue(int)));
04 a.setValue(12);     // a.value() == 12, b.value() == 12
05 b.setValue(48);     // a.value() == 12, b.value() == 48
```

Conectando las señales y los slots.

- En la línea 2 se hace el llamado a la función **QObject::connect()** la cual se encarga de establecer la conexión entre la señal y el slot ahí especificado.

- Al ejecutarse la línea 4 se realiza el llamado de la función **a.setValue(12)** la cual hace emitir la señal **valueChanged(12).** Esta a su ves es recibida en el slot **setValue()** del objeto b, es decir se hace el llamado **b.setValue(12)**. Entonces b emite la misma señal **valueChanged()** pero como esta señal (la del objeto b) no ha sido conectada a ningún slot entonces es ignorada.

- En la línea 5 se vuelve hacer el llamado de la función **setValue()** del objeto b, emitiéndose la señal **valueChanged()** la cual también es ignorada porque no ha sido conectada a ningún slot.

Conectando las señales y los slots.

- Una señal y un slot se conectan mediante la función connect() cuyo prototipo es el siguiente:

connect(sender, SIGNAL(signal), receiver, SLOT(slot));

- donde sender y receiver son punteros a **QObject**, signal y slot son funciones que tienen los mismos tipos de parámetros y **SIGNAL()** y **SLOT()** macros que convierten sus argumentos en cadenas de caracteres.

- **QObject::sender()**: Esta función retorna un puntero al objeto que envía la señal, si es llamada en el Slot activado por la señal; de otra manera retorna 0. Este puntero es válido sólo durante la ejecución del Slot. Este puntero se invalida si el objeto **sender** se destruye o si el Slot es desconectado de la Signal del **sender**.

Conectando las señales y los slots.

- Ejemplos de conexiones:

Una señal puede ser conectada a varios slots.

```
connect(slider, SIGNAL(valueChanged(int)), spinBox, SLOT(setValue(int)));

connect(slider, SIGNAL(valueChanged(int)),this,SLOT(updateStatusBarIndicator(int)));
```

Cuando la señal es emitida los slots son llamados uno después del otro sin un orden específico.

Conectando las señales y los slots.

- Ejemplos de conexiones:

Varias señales pueden ser conectadas a un mismo slot.

```
connect(lineEdit, SIGNAL(textChanged(const QString &)),
                  this, SIGNAL(updateRecord(const QString &)));
```

Cuando la primera señal es emitida, la segunda también es emitida. A parte de eso, las conexiones señales-señales no se diferencias de las señales-slot.

Las señales también pueden ser desconectadas, para ello se utiliza la función **disconect()** como se muestra a continuación:

```
disconnect(lcd, SIGNAL(overflow()), this, SLOT(handleMathError()));
```

Esto es raramente necesario porque Qt remueve automáticamente todas las conexiones que envuelven a un objeto cuando este es eliminado.

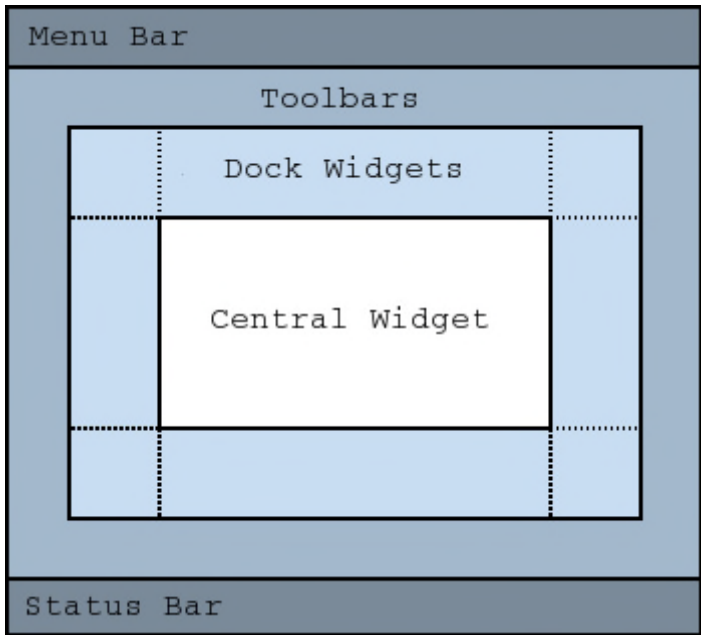
QMainWindow. La ventana principal de una aplicación.

- la ventana principal de una aplicación generalmente es una subclase de **QMainWindow**.

- La razón por la cual se trabaja con una subclase de **QMainWindow** como ventana principal es para, además de utilizar las funcionalidades generales que brinda **QMainWindow**, poder definir nuevas o incluso redefinir algunas existentes en la clase base. Esto se logra creando nuevos slots o manejando los eventos que sean oportunos, aspectos que no se pueden hacer directamente sobre **QMainWindow**.

QMainWindow. La ventana principal de una aplicación.

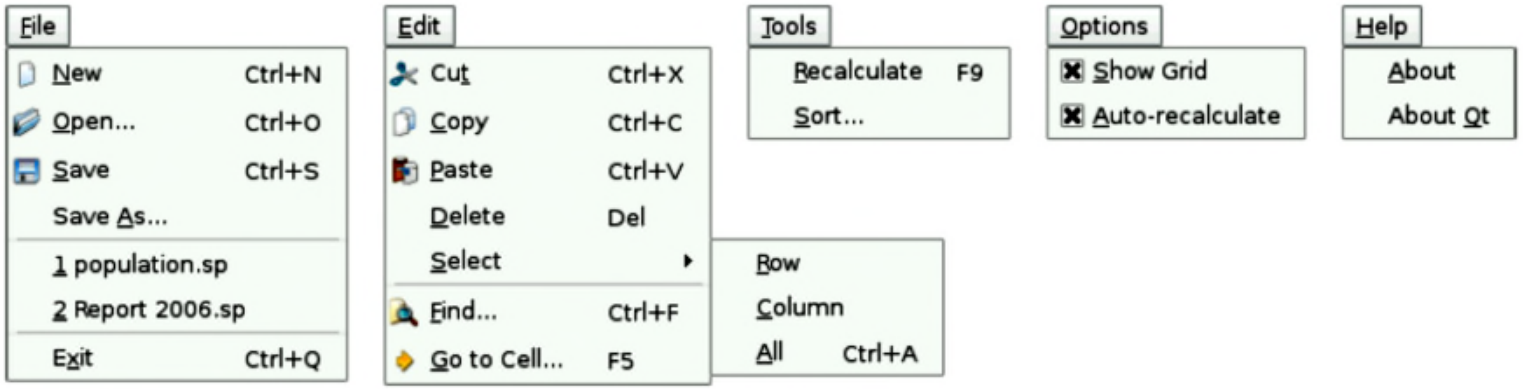
- **QMainWindow** proporciona un área de interfaz con los siguientes componentes: barra de menú (menu bar), barra de herramientas (toolbars), barra deslizante (dock widget), área central (central widget) y barra de estado (status bar).



- Cada uno de estos componentes de la Interfaz puede o no estar presentes a excepción del área central.

QMainWindow. La ventana principal de una aplicación.

- Es necesario destacar que un menú no es más que una lista de acciones, es por ello que la clase **QMenu** se usa casi siempre con la clase **QAction**.



- **QAction**: La clase QAction proporciona una acción de interfaz de usuario abstracta que se puede insertar en widgets.
- **QMenu:** proporciona un widget de menú para usar en barras de menú, menús contextuales y otros menús emergentes.

- **QMainWindow. La ventana principal de una aplicación.**

- **Ejemplo 2: Se desea realizar una aplicación visual que utilice Menú y Acciones para cambiar el texto de una etiqueta.**

  Utilice Signal &Slot.

  Ejemplo:

```
connect(ui->action1,SIGNAL(triggered()),this,SLOT(poner1()));
```

Ventanas de diálogos, comunicación con el usuario.

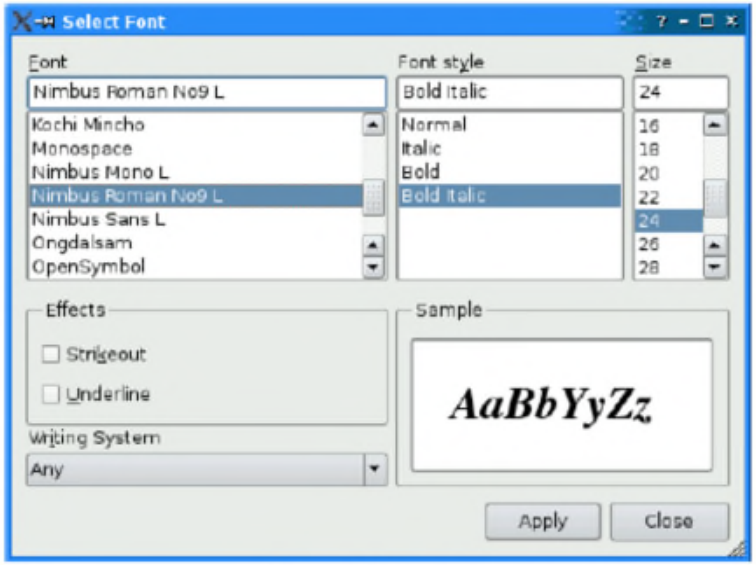
- Los diálogos son útiles en ocasiones en las que requerimos la confirmación del usuario antes de ejecutar una acción, como al guardar o eliminar un archivo o al salir de una aplicación. También suelen utilizarse para obtener pequeñas cantidades de información, como la palabra a buscar en un diálogo de buscar y reemplazar o el nombre que se desea dar a un marcador/favorito en un navegador web.

Diálogos estándares

- Ahora veremos la segunda forma de utilizar diálogos en una aplicación de Qt, esto es, utilizando las clases de diálogos estándar que nos proporciona Qt, estas clases son:
  - **QMessageBox:** Permite mostrar mensajes de notificación al usuario.
  - **QFileDialog:** Permite mostrar un diálogo que sirve para seleccionar un archivo o carpeta del sistema de archivos.
  - **QInputDialog:** Permite mostrar un diálogo con un widget para que el usuario puede introducir información.
  - **QColorDialog:** Muestra un diálogo para que el usuario pueda seleccionar un color.
  - **QPrintDialog:** Muestra un diálogo para especificar configuraciones de la impresora.
  - **QFontDialog:** Permite seleccionar un color.

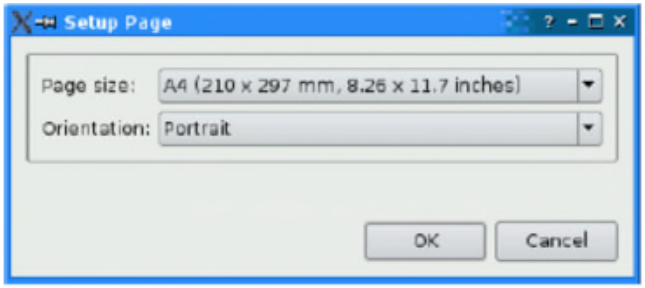
- **Ventanas de diálogos, comunicación con el usuario.**

QColorDialog

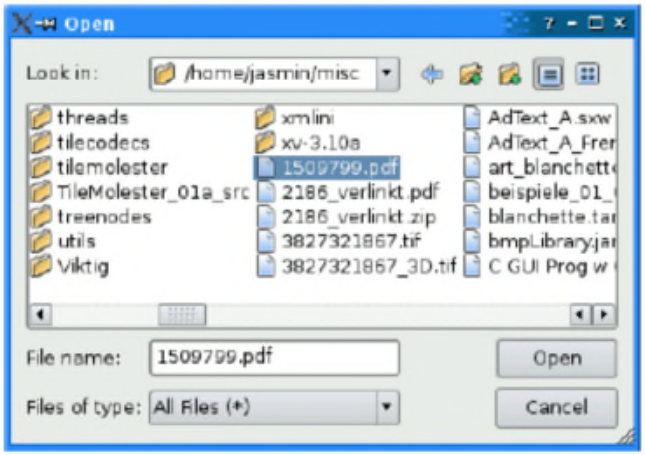


QFontDialog

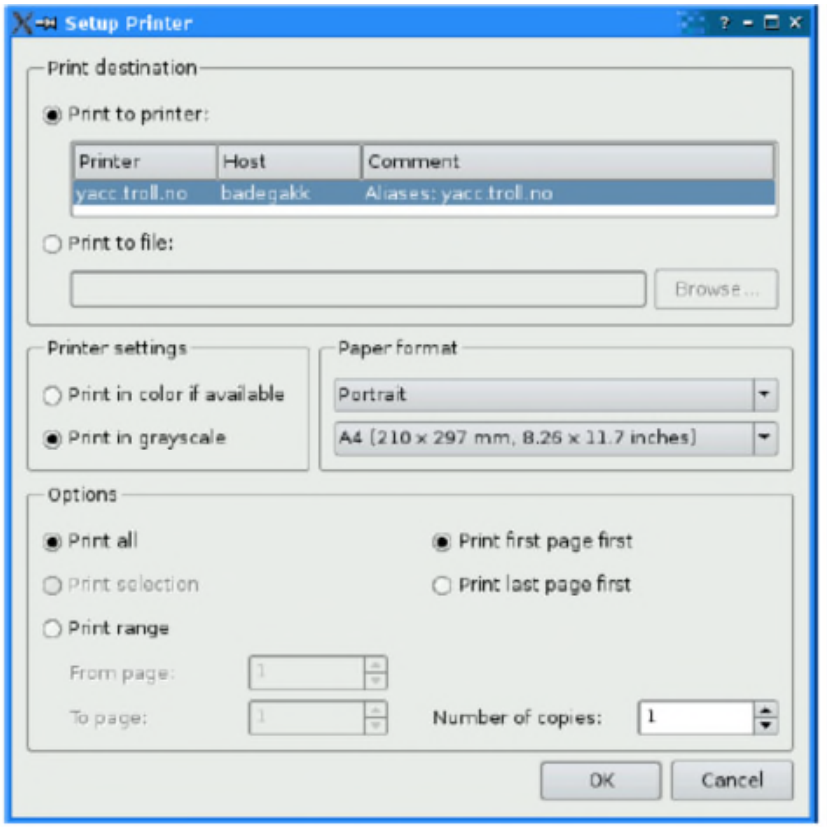
➢ Ventanas de diálogos, comunicación con el usuario.



QPageSetupDialog



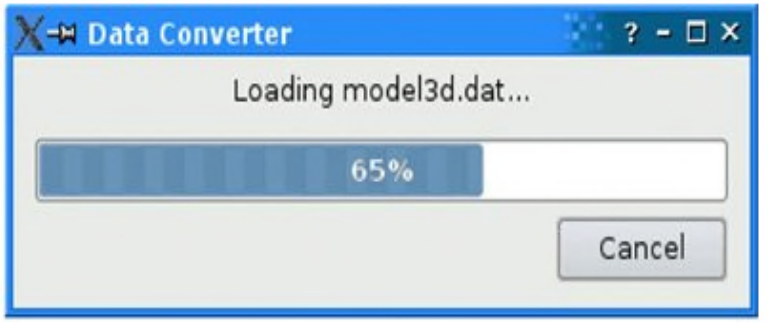
QFileDialog



QPrintDialog

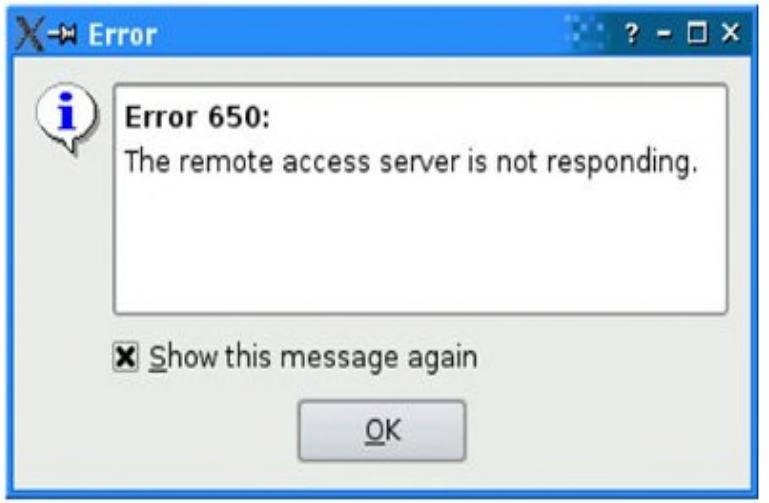
➢ Ventanas de diálogos, comunicación con el usuario.

QInputDialog



QProgressDialog

QMessageBox



QErrorMessage

- **Un ejemplo de cómo utilizarlas es el siguiente:**

```
void MainWindow::on_pushButton_clicked()
{
    QMessageBox::StandardButton reply;
    reply = QMessageBox::information(this, tr("QMessageBox::information()"),
                                     "MESSAGE", QMessageBox::Ok|QMessageBox::Cancel);
        if (reply == QMessageBox::Ok)
            ui->label->setText(tr("OK"));
        else if (reply == QMessageBox::Cancel)
            ui->label->setText(tr("Cancel"));
        else  ui->label->setText(tr("Escape"));
}
```

En este caso se ha utilizado el diálogo estándar QMessageBox para mostrar una ventana de información. Al salir de la ventana de dialogo esta devuelve información acerca de que botón fue presionado para salir.

CONCLUSIONES