Introducción al desarrollo de GUI's mediante Qt

Eric Orellana-Romero, Cristian Duran-Faundez
Departamento de Ingeniería Eléctrica y Electrónica, Facultad de Ingeniería
Universidad del Bío-Bío, Concepción, Chile
Email: eorellan@alumnos.ubiobio.cl, crduran@ubiobio.cl

Document Version: 1.0.0 – Oct-2012

Resumen

Un importante tópico referente al desarrollo de software, al que no siempre se da la importancia que merece, es la creación de interfaces de usuario amigables. Según dicta la experiencia, un proyecto con una interfaz complicada resulta difícil de aprender a utilizar, influyendo negativamente en el éxito de este, a pesar de que cumpla con los requerimientos impuestos. Dado lo anterior, en este trabajo se entregan las bases para el desarrollo de interfaces de usuario (GUI - graphical user interface), utilizando las herramientas disponibles en la biblioteca Qt para su desarrollo y el lenguaje C++ para implementar la lógica requerida. Durante el desarrollo del artículo se muestran ejemplos básicos para comprender los conceptos relacionados con Qt, concluyendo con dos aplicaciones en la que se utilizan en conjunto y se agrega mayor complejidad.

1. Introducción

Uno de los factores que causa mayor impacto en los usuarios al evaluar un software es su interfaz. Si bien es posible cumplir con todos los requerimientos exigidos por el cliente utilizando como nexo humano-máquina un intérprete de comandos (consola o terminal), un importante porcentaje de usuarios no se relaciona comúnmente con este método, por lo que se genera un aprendizaje lento y poco intuitivo, lo que puede derivar en la no utilización de la aplicación desarrollada, llevando al fracaso al proyecto [1, sección 1]. Producto de lo anterior, los diseñadores de software buscan generar interfaces de usuario (UI) sencillas e intuitivas, siendo importante considerar su presentación, la interacción con el usuario y la semántica (significado) asociada a ella. Si bien en primera instancia, un usuario evalúa la presentación de la interfaz (la que no debe sobrecargarse), posteriormente cobra relevancia su facilidad de uso (interacción), por lo que, con el propósito de mejorar la adaptación del usuario al software, se agrega semántica a los objetos que la componen. Un ejemplo de esto es la metáfora ligada al espacio principal de trabajo en sistemas operativos (Linux, Mac OS, Windows), el que es asociado a un *escritorio* [2, sección 1]. Dada la importancia de contar con un buen diseño al implementar la UI, se han generado principios para guiar su desarrollo y métricas para evaluar su funcionamiento [2, secciones 2 y 3], permitiendo crear interfaces cada vez más "amigables" al usuario, siendo ampliamente utilizadas las UI's gráficas (GUI's).

Puesto que el diseño de una interfaz de fácil aprendizaje (intuitiva, fácil interacción, etc.), buena apariencia y funcional forma parte importante del éxito de una aplicación, se han implementado diversas utilidades para desarrollar de manera sencilla GUI's que cumplan con estas características, reduciendo el tiempo y el esfuerzo por parte del grupo desarrollador. Una de ellas es la estudiada en este trabajo, el framework Qt, el que es introducido en la sección 2. Su instalación es explicada en la sección 3, mostrando como proceder en sistemas Linux y Windows. Como una forma de aclarar los conceptos asociados a su utilización, en el apartado 4 se explica su funcionamiento a través de código, para luego, en el apartado 5, utilizar las herramientas gráficas disponibles para agilizar el desarrollo, todo mediante ejemplos. Posterior a esto, en la sección 6 se crean dos aplicaciones que incluyen a los ejemplos explicados previamente y agregan una mayor complejidad, concluyendo con la sección 7, en la que se entrega la síntesis del trabajo realizado. Además, se adjunta a este trabajo el código para implementar la última aplicación de ejemplo.

2. Framework Qt para el desarrollo de GUI's

Qt es un completo framework de trabajo que entrega herramientas para la creación multiplataforma de interfaces gráficas. Fue desarrollado por la empresa Quasar Technologies[1], que posteriormente cambia su nombre a Trolltech, con el fin de contar con una herramienta que permitiera generar interfaces multiplataforma para proyectos escritos en C++. De esta forma, en 1994 fue lanzada la primera versión de Qt, `Q` por el primer nombre de la empresa que inició su desarrollo (Quasar) y `t` por la palabra `toolkit`, contando con soporte para los sistemas Linux y Windows. Producto de esto, en 2001 se añade soporte para MAC OS X, transformándose en una de las herramientas más utilizadas para la creación de entornos gráficos. En 2008 Trolltech es adquirido por Nokia, extendiéndose su aplicabilidad a sistemas móviles. En cuanto a licencias de uso, Qt es distribuido bajo 3 diferentes tipos, GPL, LGPL y comercial. El primer tipo, GPL, está orientado al desarrollo de aplicaciones libres, por lo que los códigos de ellas y cualquier cambio realizado al código fuente de Qt deben estar disponibles a la comunidad. El segundo tipo, LGPL, permite la creación de aplicaciones cerradas, pero cualquier cambio realizado a los códigos del framework deben ser compartidos. Por último, la licencia comercial permite realizar aplicaciones completamente cerradas. De esta forma, existen versiones de Qt gratuitas y de paga, dependiendo de la licencia bajo la que se rige el desarrollo [1], [3]. Así, de ser una herramienta utilizada en unos pocos proyectos, ha pasado a ser una de las más utilizadas, siendo tal vez los proyectos más conocido el gestor de ventanas utilizado en el proyecto KDE[2], Photoshop y Google Earth. En la actualidad se encuentra disponibles en la versión 4.8.3 para plataformas Linux, Mac OS y Windows.

3. Instalación del framwork Qt

Para comenzar a utilizar este framework es preciso agregar las librerías y programas (`Qt Creator`, `Qt Designer`, entre otros) asociados a él. En los próximos apartados se muestra este proceso para sistemas Linux y Windows.

3.1. Instalación bajo sistemas Linux

Linux es un sistema operativo libre que en la mayoría de sus distribuciones cuenta con un repositorio en el que se almacena gran cantidad de software gratuito, siendo administrado por la comunidad desarrolladora del proyecto utilizado (Debian, Ubuntu, Fedora, etc.). Mediante este repositorio es posible descargar e instalar aplicaciones automáticamente, utilizando para ello gestores disponibles para esta tarea. En este trabajo se utiliza el sistema operativo `GNU/Linux Debian Squeeze, release 2.6.32-5-686`, y como gestor de descarga e instalación de programas desde los repositorios, `aptitude`. El primer paso para agregar el framework Qt a Linux, es dar a conocer al gestor antes mencionado la dirección FTP del repositorio donde se encuentra contenido Qt, modificando[3] para ello el fichero *sources.list*, como se muestra a continuación[4].

1. Abrir el fichero de configuración *sources.list*, accediendo mediante consola como super-usuario:

```
# gedit /etc/apt/sources.list
```

2. Agregar la dirección FTP del servidor espejo en Chile del repositorio oficial de Linux Debian:

```
1  deb http://ftp.cl.debian.org/debian/ squeeze main contrib non-free
2  deb-src http://ftp.cl.debian.org/debian/ squeeze main contrib non-free
```

3. Guardar los cambios y cerrar.

[1] Los creadores del framework Qt son los ingenieros Haavard Nord y Eirik Chanble-Eng.
[2] Página oficial KDE, `http://www.kde.org/`
[3] Se utiliza el procesador de texto *gedit*.
[4] Se asume que la máquina cuenta con una conexión a Internet

Una vez realizados estos pasos, mediante las instrucciones mostradas abajo, en primer lugar se actualiza la lista de paquetes disponibles para ser instalados por *aptitude*, luego se proceden a instalar las librerías de C/C++ y los compiladores (gcc y g++) contenidos en el paquete *build-essential*, siguiendo con el constructor de proyectos *make* y el framework Qt. Es recomendable la utilización de gestores de instalación, pues manejan las dependencias y conflictos que pudieran generarse.

```
# aptitude -u
# aptitude install build-essential
# aptitude install make
# aptitude install qtcreator
```

Al finalizar el proceso, Qt está completamente instalado en nuestro sistema.

3.2. Instalación bajo sistemas Windows

El proceso de instalación de Qt para sistemas Windows es similar al explicado anteriormente, exceptuando el hecho de que no se cuenta con repositorios desde donde descargar e instalar los paquetes necesarios. Dado esto, se debe obtener el instalador del framework desde la página oficial del proyecto[5], disponible en la sección *Product, Download Qt*. Siguiendo estos pasos, se descarga la versión 4.8.3 bajo licencia comercial, por lo que su utilización gratuita es válida durante un periodo de prueba de 30 días. Si es preciso utilizar Qt bajo licencia GPL, es posible obtener el instalador mediante el link *Qt Project site*[6] mostrado en la misma página, a través del cual se accede al sitio open sources del proyecto. En nuestro caso utilizamos la versión 1.2.1 de Qt SDK para Windows, la que contiene la versión 4.8.1 de las librerías y los programas necesarios para agilizar el proceso de desarrollo. Así, una vez descargado el instalador, se siguen los pasos dados por el wizard de instalación, procediendo como con cualquier aplicación Windows. Se recomienda instalar todas las aplicaciones disponibles para tener una versión completa del framework.

4. Qt mediante código

Qt fue desarrollado como un framework multiplataforma utilizando el paradigma de programación orientado a objetos, por lo que cuenta con clases (atributos públicos y métodos) dedicadas a la generación de `objetos` gráficos y al control de la interacción entre ellos. A modo de introducción, se muestra un ejemplo que ejecuta el clásico mensaje *Hola Mundo*, utilizando algunas clases de Qt y el lenguaje de programación C++.

Cuadro 1: Implementación *Hola Mundo* utilizando Qt

```
1 #include <QApplication>
2 #include <QLabel>
3 int main(int argc, char *argv[])
4 {
5      QApplication app(argc, argv);
6      QLabel *label = new QLabel("Hola Mundo");
7      label->show();
8      return app.exec();
9 }
```

Mediante el código mostrado arriba se genera la ventana de la figura 1. En detalle, a través de las líneas 1 y 2 se incluyen las clases que implementan ventanas de aplicación y etiquetas, respectivamente. En una ventana de aplicación, típicamente se agregan barras de menú (`QMenuBar`), barras de herramientas (`QToolBar`), barras de estado (`QStatusBar`) y otros componentes, llamados widgets. En las líneas 5 y 6 se genera, en primer lugar, un objeto de `QApplication` llamado *app*, el que es inicializando a través de

[5] `http://qt.nokia.com/`
[6] `http://qt-project.org/`

su constructor con las variables *argc* y *argv*, y posteriormente, un puntero de tipo `QLabel` llamado *label*, mediante el cual se crea un objeto inicializado con el string *Hola Mundo*. En la línea 7, se hace visible la etiqueta creada y, para finalizar, en la línea 8, se entrega el control de la aplicación a la ventana *app*, comenzando el bucle de eventos asociada a ella. Dado que la aplicación es extremadamente sencilla, la definición de la interfaz se programa directamente en la función `main`, no siendo una práctica habitual. Es recomendable crear los widgets de forma "oculta", para de esta forma evitar el "parpadeo" de la ventana.

Figura 1: Ejecución código cuadro 1

Para conseguir un archivo ejecutable de la ventana en la figura 1, se debe compilar el código del cuadro 1. Para ello Qt cuenta con herramientas que facilitan esta tarea. Los pasos a seguir para esto son mostrados abajo, donde la primera instrucción genera un archivo intermedio, con extensión *.pro*, que identifica al proyecto, la segunda genera un archivo `Makefile`, en el que se declaran reglas de compilación de la aplicación, y la última ejecuta las reglas antes mencionadas, generando una archivo binario en base a los códigos del proyecto.

```
$ qmake-qt4 -project -o salida.pro
$ qmake salida.pro
$ make
```

Una vez concluida la compilación, como se menciona arriba, se genera una fichero binario ejecutable con el nombre dado al archivo *.pro*, en este caso *salida*, el que puede ser arrancado mediante la instrucción

```
$ ./salida
```

El ejemplo mostrado en el cuadro 1 no es de gran utilidad por sí solo, ya que no se generan interacciones con el usuario. Para esto, Qt implementa `señales entre widgets`, las que son utilizadas para controlar ciertas acciones de interés. En el código a continuación se agrega funcionalidad al ejemplo anterior para explicar este concepto.

Cuadro 2: Implementación interacción básica entre widgets

```
1 #include <QApplication>
2 #include <QPushButton>
3 int main(int argc, char *argv[])
4 {
5      QApplication app(argc, argv);
6      QPushButton *button = new QPushButton ("Quit");
7      QObject::connect(button, SIGNAL(clicked()),&app, SLOT(quit()));
8      button->show();
9      return app.exec();
10 }
```

Así, mediante el programa en el cuadro 2 se genera la ventana mostrada en la figura 2, implementando un botón `QPushButton` que al ser presionado cierra la ventana `QApplication` que lo contiene. Las diferencias con el ejemplo anterior se encuentran en las líneas 2, 6 y 7. Así, en la línea 2 se incluye la cabecera de la clase `QPushButton`, en 6 se genera un puntero a la clase y se crea el objeto *button*,

inicializandolo con el stream *Quit*, y en 7 se implementa la funcionalidad de `QPushButton`, utilizando para ello el método `connect` presente en el namespace `QObject` del framework Qt. En detalle, en la línea 7, se relaciona a *button* con *app* ejecutando la función *SLOT(quit())* sobre *app* al emitirse la señal *clicked()* una vez presionado *button*. En general este es el método utilizado para generar interacción entre widgets por Qt, `SIGNAL-SLOT`. En base a las textitseñales (`SIGNAL's`) emitidas por un objeto, se ejecutan determinados `SLOT's` (en este caso *quit()*) de otros, realizando acciones previamente establecida.

Figura 2: Ejecución código cuadro 2

Los ejemplos mostrados en este apartado explican el proceso básico de creación de las aplicaciones gráficas Qt, sin utilizar herramientas de ayuda para el programador, con el fin de comprender a nivel de código el procedimiento efectuado durante el desarrollo, permitiendo un mejor entendimiento una vez que se utilice QDesigner.

5. Qt mediante herramientas gráficas

Como se menciona anteriormente, existen diferentes programas que facilitan el desarrollo de una aplicación. Puesto que construir entornos gráficos utilizando la metodología explicada en el apartado anterior resulta engorroso y costoso (tiempo), en este trabajo se adoptan dos herramientas incluidas al instalar Qt para facilitar el proceso, `Qt Creator` y `Qt Designer`. De este modo, `Qt Creator` se utiliza como IDE de programación y `Qt Designer` como "editor gráfico de widgets". La principal ventaja de utilizar `Qt Designer` es que permite realizar el diseño de la aplicación de forma gráfica, generando la cabecera *ui_nombreProyecto.h*, que implementa el código necesario para construirla. A modo de explicar el funcionamiento de estas herramientas, se re-implementa el ejemplo mostrado en la figura 2.

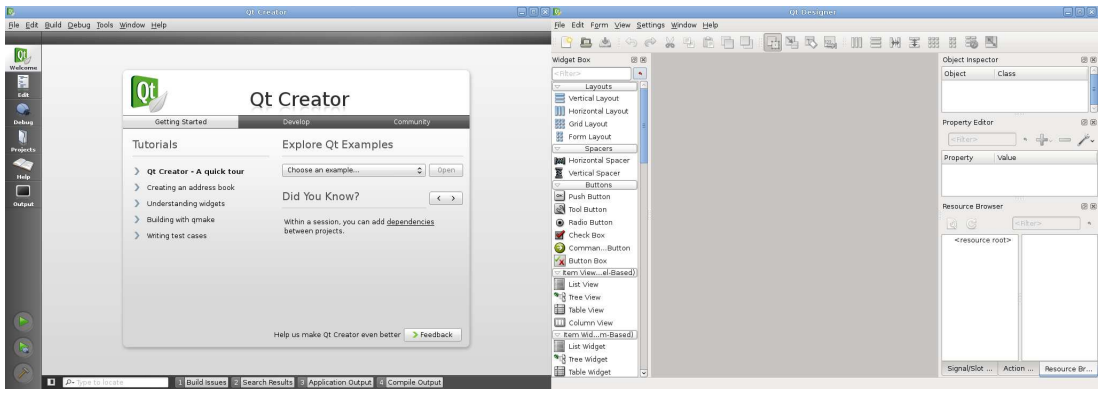

Figura 3: Interfaz `Qt Creator` (izquierda) y `Qt Designer` (derecha)

Así, para comenzar con el ejemplo se debe crear un proyecto desde el menú *File*, opción *New File or Project* de `Qt Creator` (figura 3) en el que se selecciona *Qt4 Gui Application* para generar una "plantilla"

de una GUI. En este punto se debe elegir el nombre y la dirección del directorio raíz del proyecto, que en nuestro caso será llamado *ventanaSalida*, y el nombre de los archivos que serán generados, donde modificaremos el nombre de la clase por *ventanaSalida*. De este modo, son generados los archivos *main.cpp, ventanaSalida.cpp, ventanaSalida.h, ventanaSalida.ui, ventanaSalida.pro*. Los ficheros *main.cpp* y *ventanaSalida.pro* tienen la misma función que en el apartado anterior. Sin embargo, la implementación se crea en dos archivos, *ventanaSalida.h* para la definición de atributos y métodos, y *ventanaSalida.cpp* para escribir la funcionalidad de la interfaz. Una vez creado el proyecto, abriremos el archivo *ventanaSalida.ui*, desde el directorio elegido como raíz. Los ficheros *.ui* son asociados con `Qt Designer`, por lo que este se abrirá de forma integrada en `Qt Creator`, permitiendonos crear la interfaz mostrada en la figura 2. Al abrir el archivo, al centro del espacio de trabajo se encuentra la visualización de la ventana en construcción, la que en el ejemplo cuenta con tres widgets previamente agregados, `QMenuBar`, `QMainToolBar` y `QStatusBar`, los que son eliminados (click derecho sobre el widget a eliminar, *remove ...*), ya que son innecesarios en nuestro ejemplo. De esta forma, se agregan los widgets necesarios, arrastrandolos a la interfaz en desarrollo desde la barra a la izquierda del espacio de trabajo. Para el ejemplo, se incluye un botón `QPushButton`, modificando el nombre del objeto (objectName) por *button* y la etiqueta (text) del botón (debe estar seleccionado) por *Quit*, utilizando para ello la tabla de propiedades de la esquina inferior derecha. Para finalizar, se ubicará el botón en la esquina superior derecha y se ajustará el área de la ventana en construcción (cuadrado plomo), "clickeando" sobre ella y arrastrando el cuadro azul de la esquina inferior derecha hasta conseguir que se vea como en la figura 2. De esta forma, luego de compilar, presionando la flecha verde en el borde izquierdo de `Qt Creator` (similar al símbolo de play), se mostrará la ventana creada y se generarán varios archivos adicionales, entre ellos *ui_ventanaSalida.h*. En este archivo se implementa, al compilar, el código equivalente al diseño de la interfaz creada gráficamente con `Qt Designer`. Además, se modifica el archivo *salidaVentana.cpp*, agregando la cabecera #include "ui_ventanaSalida.h".

Cuadro 3: Extracto archivo *ui_ventanaSalida.h*

```
1 ...
2 class Ui_ventanaSalida
3 {
4 public:
5     QWidget *centralWidget;
6     QPushButton *button;
7
8     void setupUi(QMainWindow *ventanaSalida)
9     { ... }
10    void retranslateUi(QMainWindow *ventanaSalida)
11    { ... }
12 };
13
14 namespace Ui {
15     class ventanaSalida: public Ui_ventanaSalida {};
16 }
17 ...
```

Para agregar la acción de cierre al objeto *button* de la clase `QPushButton` (declarado en la línea 6 del cuadro 3), se debe proceder de forma similar al ejemplo anterior, incluyendo en el constructor de la clase la conexión entre la señal emitida por él y la función *close* del objeto de *ventanaSalida*, como se muestra en el código del cuadro 4. Así, en la línea 2 se agrega la cabecera que implementa la interfaz, y se incluye la línea 9 para agregar la función de cierre. Cabe destacar que se referencia a *button* mediante *ui→button* ya que el objeto de `ventanaSalida` es llamado *ui* en *ventanaSalida.h (ventanaSalida *ui;)* y que es necesario indicar que el método *connect* es implementado en el namespace *QObject*. Luego de realizar esta modificación y compilar el proyecto la interfaz es completamente funcional.

Cuadro 4: Extracto archivo *ventanaSalida.cpp*

```
#include "ventanasalida.h"
#include "ui_ventanasalida.h"

ventanaSalida::ventanaSalida(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::ventanaSalida)
{
    ui->setupUi(this);
    QObject::connect(ui->button,SIGNAL(clicked()),this,SLOT(close()));
}

ventanaSalida::~ventanaSalida()
{ ... }

void ventanaSalida::changeEvent(QEvent *e)
{ ... }
}
```

6. Aplicación

Para finalizar, se crean dos aplicaciones utilizando los conceptos explicados previamente mediante ejemplos. La primera, `editor`, genera una lista de "palabras" ingresadas mediante un widget `QLineEdit` y almacenadas a través de un widget `QListWidget`, pudiendo ser convertidas a mayúscula o minúscula previo a ser ingresadas a la lista. Luego, al *finalizar* la aplicación, es generado un fichero *lista.data* en el directorio raíz del proyecto, el que contiene los datos ingresados a la lista. El objetivo de *editor* es mostrar el manejo de strings de texto. La segunda, implementa una *calculadora*, con el objetivo de mostrar el manejo de variables numéricas a partir de strings de texto. Se compone de un objeto de `QLineEdit`, mediante el cual se ingresan valores, ya sea por botones o directamente desde teclado, pudiendo realizar operaciones de suma, resta, multiplicación y división.

6.1. Editor

Como en los ejemplos mostrados anteriormente, el primer paso es generar un nuevo proyecto en `Qt Creator`, *File→New File or Project→Qt4 GUI Application*. Una vez creado, se edita el fichero *editor.ui* mediante `Qt Designer`, eliminando los widgets innecesarios y agregando un objeto de `QLineEdit`, uno de `QListWidget`, tres de `QPushButton` y dos de `QRadioButton`. Así, modificando las propiedades de cada uno y utilizando las opciones de agrupamiento (en la barra superior) la interfaz resultante es como se muestra en la figura 4.

Es importante el nombre que se da a cada objeto, ya que mediante él serán referenciados. Por esto, se modifican los nombres de los objetos de las diferentes clases de widgets según:

Clase de Widget	Nombre Objeto
`QListWidwet`	*lista*
`QLineEdit`	*entrada*
`QPushButton`	*agregar*
`QPushButton`	*convertir*
`QPushButton`	*finalizar*
`QRadioButton`	*mayuscula*
`QRadioButton`	*minuscula*

Luego de esto, las modificaciones para programar las funcionalidades descrita deben realizarse directamente al código. De esta forma, se agregaran tres slots, *add()*, *conv()* y *finish()*. El primero agrega el

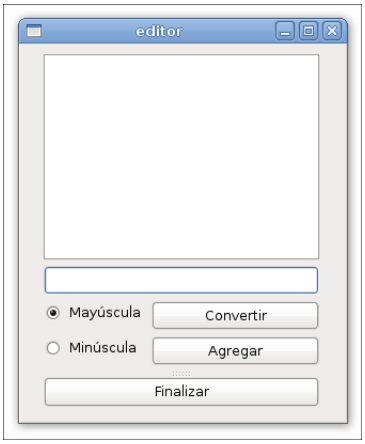

Figura 4: Interfaz *Editor*

texto escrito en *entrada* a *lista*. El segundo convierte el texto en *entrada* a mayúsculas o minúsculas, dependiendo de que objeto de `QPushButton` se encuentre activado (true: activado, false: desactivado). Por último, *finish()*, genera el archivo *lista.data* y agrega el texto en *lista*, luego termina la aplicación. Así, los slots son definidos en *editor.h* como se muestra en el cuadro 5 e implementados en *editor.cpp* como se muestra en el cuadro 6.

Cuadro 5: Definición slots de *editor*

```
1 #ifndef EDITOR_H
2 #define EDITOR_H
3
4 #include <QMainWindow>
5 #include <ctype.h>
6 #include <stdio.h>
7 #include <stdlib.h>
8
9 ...
10
11 private slots:
12     void add();
13     void conv();
14     void finish();
15
16 };
17
18 #endif // EDITOR_H
```

Posterior a esto, se establece la conexión entre las señales emitidas al ser presionado cada boton y el slot correspondiente. Dicha conexión se realiza modificando el constructor de la clase *editor* como se muestra en el cuadro 7. Además, mediante las líneas 6 y 7 se "entrega el foco"[7] a *entrada* y se selecciona por defecto el objeto de `QPushButton` *mayuscula*, respectivamente. Cabe destacar, que los objetos de la clase `QPushButton` son mútuamente excluyentes, por lo que no es necesario programar este comportamiento. Es importante recalcar que los widgets agregados no efectúan comportamientos que no se programen previamente. Un ejemplo de esto es la inserción de datos a *lista*, ya que mediante la línea 8 del cuadro 7 se establece que al emitir la señal *clicked()* por el botón *agregar* se ejecuta el slot *add()*, pero además, en la línea siguiente, se establece que al emitirse la señal *returnPressed()* por *entrada*, se ejecuta el mismo

[7]Se resalta el objeto, y en el caso de los objetos de `QLineEdit`, se prepara para ingresar texto (cursor parpadeante).

Cuadro 6: Implementación slots de *editor*

```
#include "editor.h"
#include "ui_editor.h"

...

void editor::add()
{
    if(ui->entrada->text() != NULL)
    {
    ui->lista->addItem(ui->entrada->text());
    ui->entrada->clear();
    }
    ui->entrada->setFocus();
}

void editor::conv()
{
    if(ui->mayuscula->isChecked())
    {
        QString text(ui->entrada->text());
        ui->entrada->clear();
        ui->entrada->setText(text.toUpper());
    }

    if(ui->minuscula->isChecked())
    {
        QString text(ui->entrada->text());
        ui->entrada->clear();
        ui->entrada->setText(text.toLower());
    }
    ui->entrada->setFocus();

}

void editor::finish()
{
    FILE *file;
    file = fopen("lista.dat","wb");
    for(int i = 0; i < ui->lista->count(); i++)
        fprintf(file,"%s\n",ui->lista->item(i)->text().toUtf8().data());
    fclose(file);

    close();
}
```

slot. De esta forma, los datos pueden ser ingresados presionando el botón *agregar* o presionando la tecla *enter* del teclado, una vez finalizada la escritura. Si no se agrega la línea 9 al constructor, el presionar la tecla *enter* no genera acción alguna.

Una vez finalizado este proceso basta compilar y ejecutar la aplicación para corroborar que se cumplan los objetivos propuestos.

6.2. Calculadora

Como se explica anteriormente, *calculadora* tiene por objetivo mostrar el manejo de datos numérico a partir de cadenas de texto ingresadas mediante objetos de la clase `QLineEdit`. Esta aplicación no es trivial, ya que los datos ingresados son cadenas de texto, debiendo ser convertidos a variables numéricas antes de ser utilizados, y posterior al procesamiento, deben ser re-convertidos a cadenas de caracteres para ser mostrados al usuario. La generación gráfica de la interfaz se realiza siguiendo el mismo procedimiento

Cuadro 7: Constructor de *editor*

```
editor::editor(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::editor)
{
    ui->setupUi(this);
    ui->entrada->setFocus();
    ui->mayuscula->setChecked(true);
    QObject::connect(ui->agregar,SIGNAL(clicked()),this,SLOT(add()));
    QObject::connect(ui->entrada,SIGNAL(returnPressed()),this,SLOT(add()));
    QObject::connect(ui->convertir,SIGNAL(clicked()),this,SLOT(conv()));
    QObject::connect(ui->finalizar,SIGNAL(clicked()),this,SLOT(finish()));
}
```

mostrado en el ejemplo anterior, creando la aplicación mostrada en la figura 5.

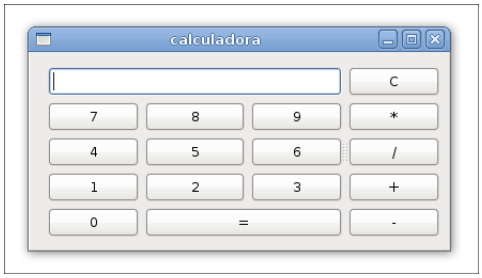

Figura 5: Interfaz de *calculadora*

Así los nombres dados a los objetos incluidos son los mostrados a continuación, y representan la funcionalidad de cada uno:

Clase de Widget	Nombre Objeto
QLineEdit	*entrada*
QPushButton	*clear*
QPushButton	*div*
QPushButton	*mult*
QPushButton	*resta*
QPushButton	*suma*
QPushButton	*igual*
QPushButton	*uno*
QPushButton	*dos*
QPushButton	*tres*
QPushButton	*cuatro*
QPushButton	*cinco*
QPushButton	*seis*
QPushButton	*siete*
QPushButton	*ocho*
QPushButton	*nueve*
QPushButton	*cero*

Una vez creada la interfaz gráfica, se debe generar la funcionalidad de esta. La lógica implementada para el ingreso de datos al objeto *entrada*, consiste en que cuando un botón de número se presiona,

todo el texto en en él se almacena en una variable temporal, para posteriormente agregar el número correspondiente y volver a escribirlo en *entrada*. Para ello se crea un slot para cada botón, el que es "lanzado" al presionarlos. Este proceso es mostrado en el cuadro 8, dando como ejemplo el botón *uno*.

Cuadro 8: Ingreso de valores a *calculadora*

```
...
void calculadora::b_uno()    { agregar_datos("1"); }
...
void calculadora::agregar_datos(const char* numero)
{
    // recuperacion de datos desde entrada
    unsigned int lenght = ui->entrada->size().height(); // cantidad de caracteres en
        entrada
    char data[lenght+1]; // variable de almacenamiento
    strcpy(data,ui->entrada->text().toUtf8().data());
    // agregacion numero
    strcat(data,numero);
    // escritura de nuevo dato en entrada
    QString input(data);
    ui->entrada->setText(input.toUtf8().data());
    ui->entrada->setFocus();
}
```

Una vez finalizado el ingreso del primer número, se debe presionar el botón asociado a la operación requerida, para luego ingresar el segundo. Para esto se utilizan dos atributos privados de la clase *calculadora*, *numero1* y *numero2*. Así, al presionar el botón de operación se almacena el valor en la variable *numero1*, permitiendo el ingreso del próximo. Para mostrar esto, se utiliza como ejemplo el operador "+", mostrando su implementación en el cuadro 9

Cuadro 9: Ingreso segundo valor de una operación

```
...
void calculadora::b_suma()
{
    guardar_datos();
    strcpy(operador,"SUM");
    ui->entrada->clear();
    ui->entrada->setFocus();
}
...
void calculadora::guardar_datos()
{
    // conversion y almacenamiento de entrada
    bool* ok = false;
    if(operador[0] == (char)NULL) numero1 = ui->entrada->text().toInt(ok,10);
    else numero2 = ui->entrada->text().toInt(ok,10);
}
```

De esta forma, a través del método privado *guardar_datos()* se almacena el primer valor en *numero1*, ya que el atributo privado *operador* es inicializado previamente como *NULL*. Luego de esto se indica que la operación a realizar es *SUM*, se borran los datos en *entrada* y se "entrega el foco" a *entrada* para el ingreso del segundo número. En el método *guardar_datos()*, se utiliza la función *toInt(...)* de la clase `QObject` (el método *text()* retorna un objeto de la clase `QObject`) para convertir los caracteres en una variable de tipo *int*.

Una vez finalizada la inserción del segundo valor, se presiona el botón *igual*, por lo que, como se

Cuadro 10: Cálculo de la operación y exposición del resultado

```
1 void calculadora::b_igual()
2 {
3     int n_resultado;
4
5     if(operador[0] != (char)NULL)
6     {
7         guardar_datos();
8
9         if(!strcmp(operador,"SUM")) n_resultado = numero1 + numero2;
10        if(!strcmp(operador,"RES")) n_resultado = numero1 - numero2;
11
12        ui->entrada->clear();
13        char *temp = itoa(n_resultado);
14        QString resultado(temp);
15        free(temp);
16        ui->entrada->setText(resultado);
17
18        numero1 = n_resultado;
19        operador[0] = (char)NULL;
20    }
21
22    ui->entrada->setFocus();
23 }
```

muestra en el cuadro 10, se almacena el segundo valor mediante el método *guardar_datos()*. En este caso, ya que *operador* contiene el valor *SUM*, es almacenado en la variable *numero2*. Luego, se realiza la operación correspondiente, y se muestra el resultado en *entrada*, para lo que debe convertirse la variable numérica *n_resultado* en una cadena de caracteres. Así, este proceso se realiza desde la línea 12 a la 16 del cuadro 10. Para ello se utiliza una variable temporal de tipo *char **, la que almacena el valor retornado por el método privado *itoa()*[8] a partir de la variable de tipo *int, n_resultado*. Luego se crea el objeto *resultado* de la clase `QString`, para almacenar el valor, en caracteres, del resultado obtenido e ingresarlo a *entrada* para que el usuario puede visualizarlo. El detalle del método *itoa()* es mostrado en el cuadro 11.

Así, luego de la implementación es posible ingresar valores mediante los botones generados para ello o mediante el teclado. La implementación completa se encuentra anexada al tutorial y se muestra el código para las operaciones de suma y resta, para cada botón numérico, para el botón igual y para el botón clear. Las operaciones de multiplicación y división son dejadas como trabajo personal para el lector.

7. Conclusiones

Qt es un poderoso framework para la generación de interfaces gráficas de usuario que no solo permite el diseño de estas, sino que, además, cuenta con métodos que facilitan el manejo de los datos, siendo considerado una completa herramienta multiplataforma para este propósito. Un ejemplo de ello, es el IDE de programación Qt Creator y el IDE de desarrollo gráfico de interfaces Qt Designer, los que trabajando en conjunto generan el código referente a la programación gráfica, reduciendo de manera notable los tiempos de desarrollo.

Para generar la interacción entre widgets se utiliza el método de *señales* y *slots* (*SIGNAL-SLOT*). Al producirse una *señal*, se ejecuta el código implementado en el *slot* correspondiente. Dicha correspondencia se realiza mediante el método *connect* de la clase `QObjects` en el constructor de cada "ventana".

[8] Convierte una variable de tipo *int* en una cadena de caracteres, retornando el puntero a ella.

Cuadro 11: Conversión *int to char**, método *itoa()*

```
1 char* calculadora::itoa(int number) // int a char
2 {
3     char* word;
4     unsigned int i;
5
6     word = (char*)malloc(sizeof(char));
7
8     // conversion int a char, orden inverso
9     for(i = 0; number > 0; i++)
10    {
11        word = (char*)realloc(word,(i+1)*sizeof(char));
12        word[i] = (char)((number%10) + 48);
13        number = (int)(number/10);
14    }
15    word = (char*)realloc(word,(i+1)*sizeof(char));
16    word[i] = (char)NULL;
17
18    // reordenamiento valores
19    char temp[strlen(word)];
20    for(i = 0; i < strlen(word); i++)
21        temp[i] = word[strlen(word)-i-1];
22    for(i = 0; i < strlen(word); i++)
23        word[i] = temp[i];
24
25    word = (char*)realloc(word,(strlen(word)+1)*sizeof(char));
26    word[strlen(word)] = (char)NULL;
27
28    return word;
29 }
```

Qt hoy en día es considerado una de las principales herramientas para la creación de GUI's, mostrando su estabilidad y rapidez en grandes proyectos, como KDE. Por lo que, si la aplicación no se comporta como se desea, es importante verificar nuestros códigos y recordar que las funcionalidades de la GUI deben ser programadas, por lo que cualquier acción que no lo sea, simplemente no funcionará.

Referencias

[1] D. G. Gutiérrez, *Tutorial de Qt4 Designer y QDevelop*, FIB-UPC, 2011.

[2] L. S. M. Gómez, *Diseño de Interfaces de Usuario. Principios, Prototipos y Heurísticas para Evaluación.*

[3] Zona qt. Sitio web dedicado al framework Qt, accesado el 3 de Octubre de 2012. [Online]. Available: http://www.zonaqt.com/