GUI – Programmierung mit Qt

C++ vs. JAVA

Einleitung

- Qt wird von der norwegischen Firma Trolltech entwickelt
  - Es ist kommerzielle Software, die aber von Trolltech für nicht kommerzielle Projekte unter dem Namen *Qt Free Edition* frei zur Verfügung gestellt wird
- Qt ist eine in C++ geschriebene Klassen-Bibliothek
  - stellt auf der einen Seite eine Menge von sichtbaren (Buttons, Labels) und unsichtbaren (timer) Elementen zur Verfügung
  - andererseits ist der Kommunikationsmechanismus (Signals/Slots) der von Qt bereitgestellt wird ein sehr mächtiges Werkzeug
- Qt ist die Grundlage von KDE
  - erleichtert durch die zur Verfügung gestellten Klassen / Elemente die GUI – Programmierung
- Qt enthält alle wichtigen Elemente die für eine GUI – Programmierung gebraucht werden, aber auch nicht mehr
  - der zu verwaltende Overhead ist also sehr gering
- Qt in C++ programmiert
  - es bietet alle Vorteile der objektorientierten Programmierung
  - Wiederverwendbarkeit und Effizienz sind hier als erstes anzuführen

Herkunft und Bedeutung des Namens

- Ursprünglich stand die Abkürzung *Qt* für **Quasar toolkit**.
- Heute hat die Abkürzung *Qt* jedoch nicht mehr diese Bedeutung und wird offiziell wie das englische Wort *cute* ausgesprochen.
- Qt wird stets mit einem kleinen "t" geschrieben und nicht als *QT*, welches für Apples Multimediasoftware QuickTime steht.

Varianten

- Qt / X11
  - Qt für das X Window System (GPL oder Proprietär)
- Qt / Mac
  - Qt für Apple Mac OS X (früher nur proprietär, jetzt auch unter der GPL)
- Qt / Windows
  - Qt für Microsoft Windows (früher nur proprietär, ab Version 4.0 auch unter der GPL)
- Qt / Embedded
  - entwickelt für PDAs und Embedded Linux (GPL oder Proprietär)

Prominente Beispiele

- die freie KDE-Desktopumgebung,
- der Opera-Webbrowser,
- das Bildbearbeitungsprogramm Photoshop Album von Adobe
- das Videoschnittprogramm MainActor der Firma MainConcept sowie
- die Office-Bibliothek von den Verlagen Brockhaus und Langenscheidt.

Lizenzierung

- Anfänglich wurde Qt für Linux unter einer eigenen Lizenz, der QPL (Q Public License), veröffentlicht, deren Qualifizierung als Freie Software strittig war.
- Seit Version 2.2 gibt es eine Duallizenzierung GPL/QPL.
- Im Februar 2005 kündigte Trolltech an, ihr Qt ab der Version 4.0 auch für die Windows-Plattform unter die GPL stellen zu wollen.
- Auch wenn die *Qt Free Edition* keinen Support durch die Firma Trolltech beinhaltet, werden trotzdem alle Verbesserungen in den kommerziellen Versionen von Qt auch an die *Free Edition* weitergegeben. Folglich entspricht auch die *Free Edition* immer dem neuesten Stand der Entwicklung von Qt.

Die Vorteile dieses Systems liegen auf der Hand

- plattformunabhängig
- für nicht – kommerziellen Gebrauch, zumindest auf Linux – Systemen kostenlos
- komplett C / C++ kompatibel
- sehr hohe Ausführungsgeschwindigkeit
- Mit Qt entwickelte Programme sind sofort ohne zusätzlichem Portierungsaufwand sowohl unter allen Unix – wie auch unter allen Windows – Systemen lauffähig
- Der härteste Konkurrent von Qt ist wohl Java, das auch mit einer absoluten Plattformunabhängigkeit daher kommt
  - letztere Sprache ist aber im Vergleich zu Qt wesentlich langsamer

Qt im Überblick

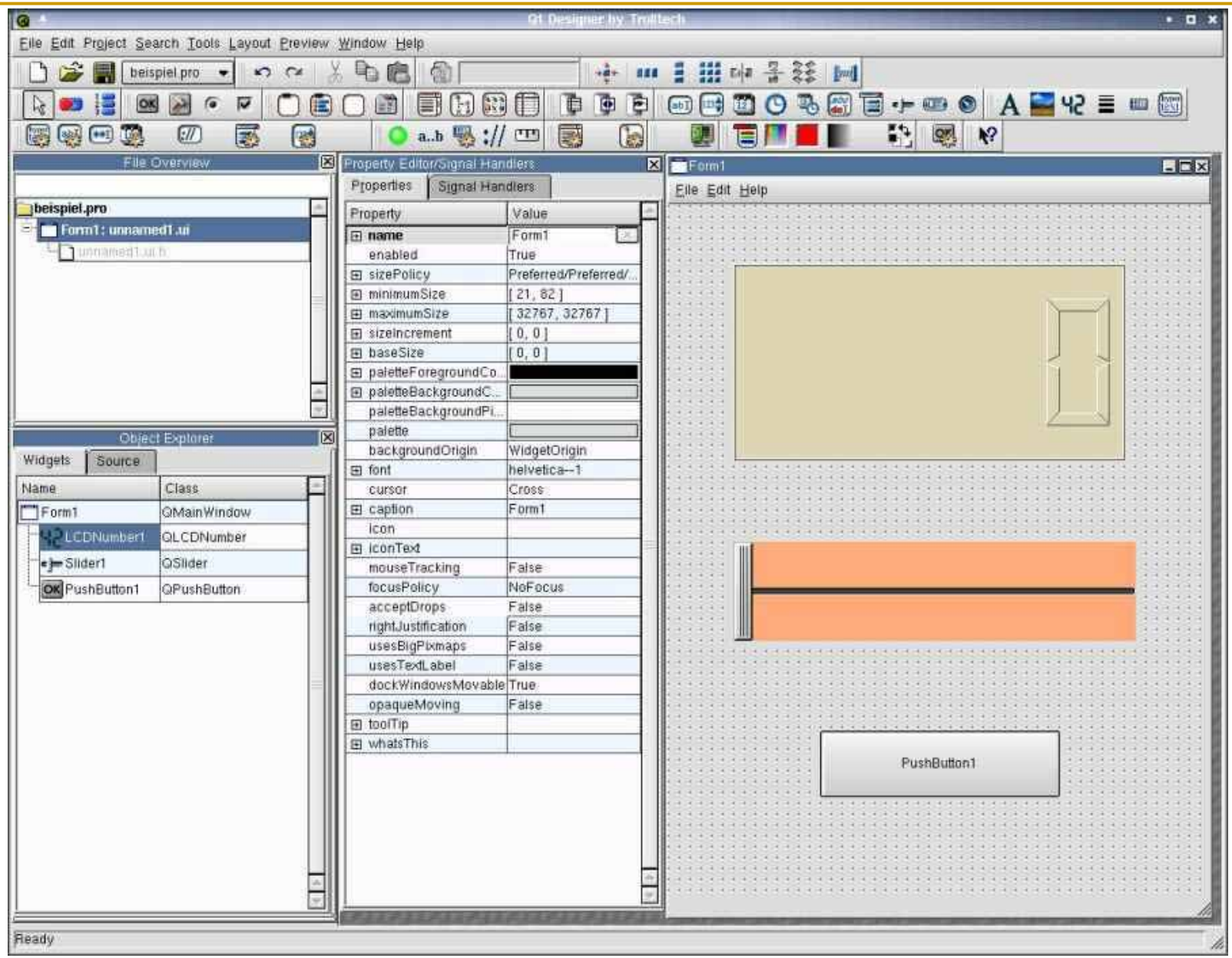
- C++ Entwicklungsumgebung
  - beinhaltet eine Klassenbibliothek
  - einen GUI - Designer
  - und Tools für OS – übergreifendes Programmieren und zur Internationalisierung

Qt - Klassenbibliothek

- Hauptkomponente
- fast 400 objektorientierte Klassen für:
  - GUI
  - GUI – Layout
  - Datenbanken
  - Internationalisierung
  - Netzwerk
  - XML
  - ...

GUI – Design

- Qt – Designer
  - eigenständiger GUI – Builder
  - mittels drag & drop oder per Editor zeilenweise Layouts für Programme erstellen
  - Qt – Designer kann Programme direkt testen


Qt Designer by Trolltech
File Edit Project Search Tools Layout Preview Window Help
beispiel.pro
File Overview
beispiel.pro
Form1 : unnamed1.ui
unnamed1.ui.h
Object Explorer
Widgets
Source
Name
Class
Form1
QMainWindow
LCDNumber1
QLCDNumber
Slider1
QSlider
PushButton1
QPushButton
Property Editor/Signal Handlers
Properties
Signal Handlers
Property
Value
name
Form1
enabled
True
sizePolicy
Preferred/Preferred/...
minimumSize
[21, 82]
maximumSize
[32767, 32767]
sizeIncrement
[0, 0]
baseSize
[0, 0]
paletteForegroundCo...
paletteBackgroundC...
paletteBackgroundPi...
palette
backgroundOrigin
WidgetOrigin
font
helvetica--1
cursor
Cross
caption
Form1
icon
iconText
mouseTracking
False
focusPolicy
NoFocus
acceptDrops
False
rightJustification
False
usesBigPixmaps
False
usesTextLabel
False
dockWindowsMovable
True
opaqueMoving
False
toolTip
whatsThis
Form1
File Edit Help
PushButton1
Ready

Signal – / Slot – Konzept

- Bei Qt können Objekte Signale aussenden und empfangen. Ein Slot ist ein Unterprogramm, das Signale von anderen Objekten auswertet.
- Connections – Dialog um Signale mit Slots zu verbinden.
- User kann eigene Slots und Signale anlegen (neben vorh. Standards)

Signal – / Slot – Konzept

- Callback – Routinen in Qt werden durch das sehr mächtige Signal – / Slot – Konzept realisiert
  - in den Klassendefinitionen der Qt – Klassen zusätzliche Methoden als Signal oder Slot deklariert
  - diese können beliebige Parameteranzahlen und Typen haben, der Rückgabewert muss aber in jedem Fall void sein
  - damit eine Klasse Signal- und Slot – Methoden benutzen kann, muss sie von der Klasse QObject abgeleitet sein

Signal – / Slot – Konzept

- Slot – Methoden sind Methoden, die eine Aktion als Folge einer Interaktion ausführen sollen
  - Sie müssen vom Programmierer mit Code gefüllt werden.
  - werden bei einem auftretendem Signal abgearbeitet, können aber auch wie ganz normale Methoden aufgerufen werden
  - Routine zur Speicherung einer Datei sollte z.B. als Slot – Methode implementiert sein

- Signal-Methoden werden vom Programmierer nur deklariert, jedoch nicht mit Code gefüllt
  - führen keine Aktionen aus, sondern rufen nur Slot – Methoden auf, mit denen sie verbunden sind

Beispielimplementation zur Benutzung des Signal – / Slot – Konzept

```
1
2   MyWindow::MyWindow() : QWidget()
3   {
4   // Create button1 and connect button1->clicked() to this->slotButton1()
5   button1 = new QPushButton("Button1", this);
6   connect(button1, SIGNAL(clicked()), this, SLOT(slotButton1()));
7
8   // Create button2 and connect button2->clicked() to this->slotButton2()
9   button2 = new QPushButton("Button2", this);
10  connect(button2, SIGNAL(clicked()), this, SLOT(slotButton2()));
11
12  // When any button is clicked, call this->slotButtons()
13  connect(button1, SIGNAL(clicked()), this, SLOT(slotButtons()));
14  connect(button2, SIGNAL(clicked()), this, SLOT(slotButtons()));
15  }
16
17  void MyWindow::slotButton1() // Slot is called when button1 is clicked.
18  { cout << "Button1 was clicked" << endl; }
19
20  void MyWindow::slotButton2() // Slot is called when button2 is clicked
21  { cout << "Button2 was clicked" << endl; }
22
23  void MyWindow::slotButtons() // Slot is called when 1 or 2 were clicked
24  { cout << "A button was clicked" << endl; }
25
```

Signal – / Slot – Konzept

- Verbinden von Signalen ist kein statischer Prozess
  - es ist möglich dynamisch zur Laufzeit des Programms Verbindungen zu erstellen oder diese mit Hilfe der Methode *disconnect* auch wieder zu lösen
  - Qt übernimmt das Lösen von Verbindungen automatisch im Destruktor von QObject, so dass ein Programmabsturz aufgrund eines Aufrufs einer nicht mehr vorhandenen Slot – Methode ausgeschlossen ist.

Meta – Object – Compiler (MOC)

- Der Meta – Object – Compiler ist ein nützliches Tool im Qt Paket
  - kein wirklicher Compiler
  - MOC konvertiert Qt Klassen Definitionen in C++ Code
  - MOC sucht im Quellcode der Klassen nach Qt – Schlüsselwörtern und ersetzt diese durch deutlich längeren und komplizierten Quellcode
- MOC Schlüsselwörter sind:
  - *Q_OBJECT*, *public slots:*, *protected slots:*, *private slots:*, und *signals:*
- Für den Applikationsentwickler bedeutet dies eine riesige Zeit – und Aufwandsersparnis
  - muss sich nicht um die interne Verarbeitung des Signal – / Slot – Konzepts kümmern

MOC – Beispiel

Headerdatei mit Qt – Schlüsselwörtern

```
1
2    #include <qwidget.h>
3    #include <qpushbutton.h>
4
5    class MyWindow : public QWidget
6    {
7    Q_OBJECT // Enable signals and slots
8    public:
9    MyWindow();
10   public slots: // This slots section is public
11   void slotButton(); // A public slot
12   private:
13   QPushButton *button;
14   };
15
```

Programmcompilation:

```
1
2  g++ -I$QTDIR/include -c main.cpp
3  moc mywindow.h -o mywindow.moc
4  g++ -I/$QTDIR/include -c mywindow.cpp
5  g++ -o myprog main.o mywindow.o -L/$QTDIR/lib -lqt
6
```

Ein einfaches Programm

```
1  #include <qapplication.h> // QApplication
2  #include <qmainwindow.h> // QMainWindow
3
4  using namespace std;
5
6  int main(int argc, char **argv)
7  {
8  QApplication app(argc, argv);
9
10 QMainWindow *window = new QMainWindow();
11
12 window->setGeometry(200, 200, 400, 300);
13
14 app.setMainWidget(window);
15
16 window->setCaption("My QMainWindow example");
17
18 window->show();
19
20 return app.exec();
21 }
```

Quellen

- **www.trolltech.com**
- **Das Komponentenmodell von KDE 2 –** Torben Kuseler – Fachhochschule Wedel
- **Testbericht von „Das Qt Buch“ von Prof. Dr. Helmut Herold** - Johannes Niederlöhner – www.hardwares.de
- **„Qt“ –** de.wikipedia.org
- **Artikel „Qt – Programmierung“ von Thomas Gern** – www.pro-linux.de