GUI – Programmierung mit Qt

C++ vs. JAVA

Ausarbeitung für einen Vortrag im Oberseminar Softwareentwicklung an der HTWK – Leipzig

Vortragender: Norman Wolf

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

Die traditionelle X Window Programmierung war eine komplizierte, anstrengende und zeitraubende Aufgabe.
Für ein so großes Projekt wie KDE war es nicht mehr möglich, diesen traditionellen Weg zu gehen. Das KDE-Projekt musste sich im Laufe seiner Entwicklung für ein Toolkit entscheiden, um der ständig wachsenden Größe des Projektes Rechnung zu tragen. Die Wahl viel auf Qt und nicht auf konkurrierende Toolkits wie gtk, xforms, xlib, da es von den Mitgliedern des Projektes für das am besten geeignete Toolkit gehalten wurde.

- 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**.

Quasare sind die dauerhaft am stärksten strahlenden Objekte im Universum (sie werden nur kurzzeitig von den Gammablitzen übertroffen). Durch den Bezug auf solche unvorstellbar energieintensiven Objekte sollte wohl der ehrgeizige Anspruch der Entwickler zum Ausdruck kommen, ein bedeutendes Programmierwerkzeug zu schaffen. Ihre Firma Trolltech hieß daher ursprünglich auch *Quasar Technologies*.

- Heute hat die Abkürzung *Qt* jedoch nicht mehr diese Bedeutung und wird offiziell wie das englische Wort *cute* ausgesprochen.

Dieses Wort soll die Ansicht der Entwickler ausdrücken, dass der Quelltext und die API von Qt eben *cute* sei, was auf Deutsch unter anderem so viel wie *pfiffig*, *schlau*, *hübsch* usw. heißt.

- 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.

Dies führte dazu, dass sich viele Linux-Entwickler statt für Qt und KDE für das GIMP-Toolkit GTK+ entschieden, auf dessen Basis sich das GNOME-Projekt begründete.
Als der Druck auf Trolltech größer wurde und das Debian-Projekt begann, KDE als unfreie Software zu klassifizieren, änderte Trolltech im Jahre 2000 die Lizenz für die Linux-Version des Toolkits.

- 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.

Vorteile des Systems

- 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

Sie müssen lediglich mit den richtigen Compilern (Visual C++, Borland C++ unter Windows) oder dem entsprechenden cc-Compiler auf dem Linux/Unix-System kompiliert werden.

- 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

Außerdem können C Programmierer mit nur geringen Kenntnissen in C++ bereits schnell Programme in Qt schreiben. Und die Anzahl an Entwicklern mit diesem Profil ist sehr hoch. Das ist wohl auch ein Grund warum Qt auf so große Beliebtheit stößt.

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

Signal – / Slot – Konzept

Eine Besonderheit ist die Verwendung von „signals“ und „slots“, die auf einfache Art und Weise die Kommunikation zwischen einzelnen Objekten ermöglicht. Bei den meisten anderen Klassenbibliotheken wird dies durch Callback-Funktionen realisiert. Die Herangehensweise mittels Callback-Funktionen hat zwar einen geringen Vorteil bezüglich der Ausführungsgeschwindigkeit, aber beachtliche Nachteile bei der Pflege der Programme.

- 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)

- 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

Callback – Routinen sind Funktionen oder Methoden, die ausgeführt werden sollen, sobald ein Widget einer Interaktion des Benutzers unterliegt, z.B. wenn ein Button gedrückt wird, ein Menüpunkt ausgewählt oder eine Scrollbar verschoben wird.
Bei Callback – Routinen Konzepten anderer Toolkits besteht diese Möglichkeit nicht, sie lassen meist nur eine sehr einfache Parameterstruktur zu. Dies ist zurückzuführen auf die Tatsache das Callbacks Pointer auf Funktionen sind, die dann bei einem eintretenden Ereignis ausgeführt werden. Diese Funktionspointer sind aber nicht in der Lage, jede Art und Anzahl von Parametern zu verwenden.
(Das sind z.B. alle GUI – Widgets, da sie von QWidget abgeleitet sind, welches seinerseits von QObject abgeleitet ist.)
Sie würde auf ein Drücken des Menüpunktes Speichern reagieren, ebenso wäre es möglich, diese Routine regelmäßig aufzurufen, um eine Sicherheitsspeicherung zu erzielen.
Der Code für die Methode wird vom Meta – Object – Compiler (MOC) generiert und nachher zum Programm hinzugelinkt.

- 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

```
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
```

Nachdem der Knopf *button1* erzeugt wurde (Zeile5), wird er mit Hilfe des *connect* Befehls (Z.6) an eine Slot – Methode gebunden. In diesem Fall handelt es sich um die Methode *slotButton1* (Z.17).
button2 (Z.9) wird an die Methode *slotButton2* (Z.20) gebunden (Z.6).
Beim Klicken auf einer der Buttons würde dann die entsprechende Slot – Methode ausgeführt werden.

Das Signal – / Slot – Konzept von Qt lässt aber auch die Verknüpfung von mehreren Signalen an eine Slot – Methode zu. Die Zeilen 13 und 14 verbinden (*connect*) beide Buttons mit der selben Slot – Methode *slotButtons*, es wäre hiermit egal, auf welchen der Button geklickt wird, die Methode würde ausgeführt werden.
Ebenso ist es möglich mit einem Signal mehrere Slot – Methoden auszuführen.

- 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

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

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
```

Nachdem die Datei: *main.cpp* (Zeile2) compiliert wurde, werden mit Hilfe des Meta – Object – Compilers sämtliche Qt – Schlüsselwörter durch C++ Code ersetzt(Z.3). Jetzt erst ist es möglich die Datei: *mywindew.cpp* (Z.4) zu übersetzen, da ihr Headerfile: *mywindow.moc* kein Qt – Code mehr enthält.
Abschließend werden die Dateien noch zu dem ausführbarem Programm: *myprog* zusammengelinkt (Z.5).

Ein einfaches Programm

```
1  #include <qapplication.h>  // QApplication
2  #include <qmainwindow.h>   // QMainWindow
```

Die ersten beiden Zeilen des Quellcodes inkludieren die benötigten Header-Dateien. In den meisten Fällen entsprechen die Namen der Header-Dateien den Klassen, die sie enthalten. Einzige Ausnahmen bilden einige wenige Header, die mehrere Klassendefinitionen enthalten. Als Beispiel sei hier auf den Header qlistview.h verwiesen, der neben der Definition der Klasse QListView auch noch die Definition der Klasse QListViewItem enthält.
Der Header qapplication.h enthält die Klasse QApplication und ist in jedem Qt-Programm notwendig.

```
8    // Create a QApplication object (required)
9    QApplication app(argc, argv);
```

In jeder Qt-Applikation ist die Instanziierung eines QApplication-Objekts notwendig. Das QApplication-Objekt übernimmt das ganze Event-Handling.

```
11   // Create a QMainWindow object
12   QMainWindow *window = new QMainWindow();
```

Diese Zeile erzeugt eine Instanz der Klasse QMainWindow. Dieses Widget wird das Hauptfenster der Applikation darstellen.

```
14   // Move and resize the QMainWindow object
15   // left=200, top=200, width=400, height=300
16   window->setGeometry(200, 200, 400, 300);
```

QMainWindow bietet die Funktion setGeometry(int, int, int, int) an, mit der die Position und Größe des Fensters festgelegt werden kann. Die ersten beiden Parameter legen die X- und Y-Koordinaten (in Pixel) auf dem Bildschirm fest und beiden weiteren Parameter definieren Breite und Höhe (in Pixel) des Fensters.

```
18     // Main window = QMainWindow object
19     app.setMainWidget(window);
```

Dem QApplication-Objekt app muss noch mitgeteilt werden, dass window die Rolle des Hauptwidgets übernehmen soll.

```
21   // set Window Title
22   window->setCaption("My QMainWindow example");
```

Diese Anweisung setzt den Fenstertitel der Applikation.

```
24   // Show the window
25   window->show();
```

Jedes Widget kann entweder sichtbar oder unsichtbar sein. Per default sind nur Widgets sichtbar, die Subwidgets von einem sichtbaren Widget sind. Alle anderen Widgets sind erst einmal unsichtbar und müssen explizit mit show() sichtbar gemacht werden.

```
27   // Go to the main loop (required)
28   return app.exec();
```

Als letztes wird die Kontrolle wieder dem QApplication-Objekt app übergeben.

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