Qt Framework – Ein Überblick

Jan Klass
Hochschule Offenburg

Mai 2009

Abstract: Diese Arbeit stellt die Bibliothek Qt, genauer ihre Geschichte sowie ihre Features vor.

1 Einleitung

Bei der Entwicklung von Software mit GUI-Oberfläche stellt sich immer die Frage, welche Bibliothek zur Vereinfachung der Programmierung einer solchen verwendet werden soll.

Soll die Software auch noch plattformunabhängig sein, so schränkt dies die Auswahl an möglichen, guten, Bibliotheken bereits stark ein.

Nicht zuletzt im Linux Umfeld, vor allem durch den KDE Desktop, hat sich Qt einen sehr guten Namen gemacht und wird nach wie vor in zahlreichen Programmen verwendet. Neben dem Linux und KDE Umfeld wird es häufig für plattformunabhängige Software verwendet: Qt erlaubt die Ausführung auf den Desktop Systemen Unix, Windows und Mac, sowie auf Embedded Systemen mit Embedded Linux und Windows CE und S60 (Smartphones).

Kritik gab es, vor allem in der Linux/Unix Gemeinde, in der Vergangenheit hauptsächlich wegen der nicht freien Lizenzen, unter denen Qt damals stand. Nach dem Wechsel auf die GPL wurde mit der Version 4.5, die in diesem Jahr veröffentlicht wurde, Qt nun, neben der kommerziellen Lizenz, unter die LGPL gestellt. Die LGPL erlaubt es die Bibliothek auch in proprietärer Software zu verwenden, sofern diese Qt extern einbindet, also nicht verändert oder mit in die Software kompiliert. Vor allem für Firmen, die zusätzlich zu der offenen, umfangreichen Dokumentation noch offiziellen Support wünschen, oder ihnen die LGPL nicht ausreicht, steht noch eine kommerzielle Lizenz zur Verfügung. Dabei fährt Qt Software ein variables Preismodell.

Grund genug sich einmal genauer mit Qt, das neben der Unterstützung bei der GUI-Erstellung und der Plattformunabhängigkeit noch viele weitere den Entwickler unterstützende Funktionen bietet, auseinander zu setzen.

2 Historie

1998 war Qt zunächst eine C++ Klassenbibliothek, welche die Entwicklung von GUI-Software, die eventuell auch auf mehreren Plattformen laufen sollte, vereinfachen sollte. Qt unterstützte damals lediglich X11 (Unix) und Windows.

Mit der Version 3.0, die im Jahr 2003 [QtS09] veröffentlicht wurde, konnten dann auch Mac-OS-X Versionen kompiliert werden.

Bis heute folgte entsprechende Unterstützung für Embedded Linux, Windows CE und S60.

Die um diese Kernfunktionalitäten vorhandenen Features wuchsen jedoch enorm an. Diese sind heute in Module eingeteilt, so dass für ein Projekt nur die notwendigen eingebunden werden, was den Overhead reduziert.

So gibt es mittlerweile, aktuell ist Version 4.5, Unterstützung für Multi-threading Anwendungen, Ressourcen-Dateien, Internationalisierung, SVG Grafiken, Zugriff auf OpenGL, eine QScript getaufte Scriptsprache, ein PDF Backend, Unterstützung für das Open Document Format, und zahlreiche weitere. [QtS09]

3 Lizenzen

Die aktuelle Version 4.5 wird mit drei Lizenzen angeboten: [QtS09]

1. Qt Commercial Version,
   die kostenpflichtige Version, gedacht für Unternehmen, mit offiziellem Support.
2. Qt GNU LGPL v. 2.1,
   kostenlos, erlaubt die Verwendung von Qt als Bibliothek ohne den eigenen Programmcode offen zu legen.
3. Qt GNU GPL v. 3.0,
   gedacht für Software Projekte die explizit unter der GPL entwickelt und veröffentlicht werden sollen.

Mit der Möglichkeit Qt unter der LGPL zu verwenden wurde einer der wichtigsten Kritikpunkte entfernt. Unternehmen können nun, seit der Version 4.5, die im März 2009 veröffentlicht wurde, mit einer kostenlosen Lizenz proprietäre und kommerzielle Software entwickeln, die Qt als Bibliothek nutzt.

4 Features

4.1 Qt Projekte und qmake

Bei der Entwicklung mit Qt wird [Wol07] unter anderem auch nicht Standard-konformer C++ Code verwendet. Aus diesem, und weiteren Qt verwaltungstechnischen Gründen, wird für jedes Qt Programm eine Qt Projektdatei (Dateiendung pro) angelegt.
In dieser Projektdatei wird unter anderem angegeben, welche Qt Module verwendet werden, ob das Programm eine GUI-Anwendung, Bibliothek, oder Konsolen-Anwendung ist, welches die Quelldateien sind, welche Bibliotheken verwendet werden, welche Übersetzungsdateien verfügbar sind, welche Ressourcen-Dateien zu erstellen sind, usw.

Der Umfang der Qt Projektdatei lässt sich vor allem durch einen Umstand erklären: Mit dem Tool qmake wird sämtlicher Quellcode in C++-Standard konformen Code umgewandelt, und die für dieses Betriebssystem angepassten Makefile-Dateien erstellt.
Der eigentliche Übersetzungsvorgang zu einem Programm kann dann von einem GNU-make kompatiblen Compiler durchgeführt werden.

Ein solcher Vorübersetzungsschritt ist aufgrund des Umfangs und der Funktionen von Qt unabdingbar.

4.2 Signal-Slot-Konzept

Im Gegensatz zu Qt verwenden viele [Wol07] grafische Bibliotheken Rückruffunktionen, also Funktionszeiger, um Nachrichten zwischen GUI-Steuerelementen zu versenden. Rückruffunktionen haben aber zwei entscheidende Nachteile: [Wol07]

- Sie sind nicht typensicher
- Die Rückruffunktion ist fest mit der auszuführenden Funktion Verbunden

Qt nutzt hier ein dynamisches, Ereignis-getriebenes, so genanntes Signal-Slot Prinzip. Bei diesem werden Verbindungen automatisch getrennt, wenn eines der kommunikationsfähigen Objekte der Verbindung zerstört wird. Das verhindert den Verweis auf nicht existierende Objekte.

In der Praxis wird dazu die statische Methode connect von QObject aufgerufen, welcher man das Sender Objekt und das aktivierende Signal, sowie das Empfängerobjekt und den Namen von dessen Slot-Funktion übergibt.
Da fast alle Qt Klassen von der Basisklasse QObject erben, ist diese Funktion auch Klassenfunktion der meisten Klassen.
Wird zur Laufzeit dann beim Sender das Signal getriggert, so wird die Slot-Funktion des Empfängers aufgerufen.

Slot-Funktionen müssen im Quellcode explizit als solche gekennzeichnet sein. [Wol07]
Ein Slot kann stets nur aufgerufen werden, wenn das Signal mindestens die vom Slot erwarteten Parameter bereit stellt. Ist diese Bedingung nicht erfüllt, wird keine Verbindung zwischen Signal und Slot erzeugt, der Übersetzungs- und Ausführungsprozess funktioniert aber ohne Probleme.

Möchte man eigene Signal- oder Slot-Funktionen realisieren, so ist auch dies möglich:
Die Klasse muss von der Basisklasse QObject oder einer Unterklasse erben, das Makro Q_OBJECT unmittelbar nach dem Klassenkopf enthalten sein

und dann können Slot-Funktionen in einem slots: Bereich, und Signal-Funktionen in einem signals: Bereich definiert werden.

Um innerhalb der Klasse dann ein Ereignis zu triggern wird die Signal-Funktion, wahlweise mit dem vorangestellten und verdeutlichenden Stichwort emit, aufgerufen.

Eine Slot-Funktion muss dabei selbst implementiert werden, eine Signal-Funktion dagegen nicht. Diese wird von dem von qmake ausgeführten Meta Object Compiler (MOC) erstellt. [Wol07]

4.3 Widgets

Als "Widgets", zu deutsch Steuerelemente, werden in Qt GUI Klassen bezeichnet, die eine Anzeigefunktionalität besitzen. Dabei dient die Klasse QWidget als Basisklasse.

Ein übliches Prinzip [Wol07] bei der Entwicklung einer eigenen GUI-Anwendung ist für ein Fenster eine eigene Klasse zu erstellen, welche etwa von QWidget oder QDialog erbt und in dessen Konstruktor dann bereits die benötigten Fensterelemente erzeugt und positioniert werden und mittels der Signal-Slot-Funktionen funktional miteinander verbunden werden.

Muss der von einem Fenster dann geändert werden, so muss nur eine Klasse und damit nur eine Quelldatei bearbeitet werden.

4.3.1 Layout-Widgets

Layout Klassen, von QLayout erbend, bieten Funktionen um Widgets einem Fenster auf eine bestimmte Weise hinzuzufügen und anzuordnen. In der Praxis werden GUI Fenster mit dem Qt Designer, mittlerweile im Qt Creator IDE integriert, erstellt, wo Widgets auch ohne Layout Elemente eingefügt und angeordnet werden können. Aber auch dort vereinfachen sie die Anordnung von Widgets immens, da sie diese gruppieren und automatisch anordnen.

4.3.2 Hauptfenster

Die Klasse QtMainWindow ist als Hauptfenster für Applikationen gedacht.
Es hat bereits die in den meisten Anwendungen vorkommenden Widgets bereits.

4.3.3 Weitere Widgets

Neben dem Hauptfenster gibt es weitere Standarddialog-Widgets wie ein Druck- oder Datei-Dialog.

Container sind Widgets, die andere Widgets enthalten. Sie sind dabei aber, im Gegensatz zu Layout Widgets, sichtbar und ordnen enthaltene Elemente nicht automatisch an. Etwa das QTabWidget, welches Tab-Fenster erlaubt.

Neben den üblichen GUI-Elementen Label, List, Combobox usw. bietet bereits das Text-Bearbeitungs-Feld die Formatierung von Text mit HTML an.

4.4 Internationalisierung

Um die Internationalisierung von Software zu unterstützten bietet Qt sehr hilfreiche Methoden. Im Quellcode wird die Software vom Entwickler in einer Standard-Sprache entwickelt. Jeglicher, später auf der Oberfläche und damit zu übersetzender, Text wird dabei auf folgende Weise "markiert":

```
QObject::tr("Ausgabetext");
```

Um eine Eindeutigkeit auch bei Mehrdeutigen Wörtern zu gewährleisten, wie auch um generell Wörter oder Sätze bzw. deren Nutzen erläutern zu können, erlaubt Qt das kommentieren von als lokalisierbar markiertem Text. Dies wird etwa auf folgende Weise getan:

```
tr("Ausgabetext", "Ein Test-Text");
```

Da sämtliche Widget-Klassen von QObject erben, ist die Funktion tr dort auch immer eine Klassenfunktion.

Oft sollen in den zu übersetzenden Text Variablenwerte, die erst zur Laufzeit bekannt sind, eingefügt werden. Die Funktion tr erlaubt für diesen Fall die Verwendung von Platzhaltern, welche mit arg(<einzufügender Wert>) ersetzt werden:

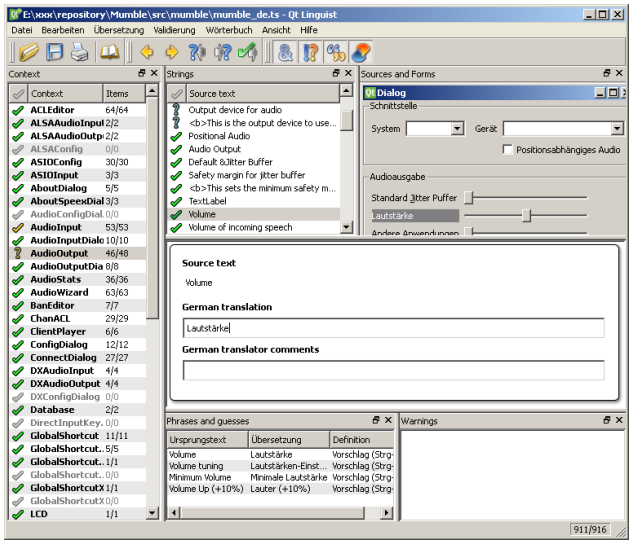
```
tr("Wert %1").arg(ivar);
```

Bevor die zu übersetzenden Textstellen in eine Übersetzungsdatei extrahiert werden können, so müssen zuvor die gewollten Übersetzungen in der Qt Projektdatei angegeben werden, beispielsweise durch:

```
TRANSLATIONS = MeinFenster_fr.ts
```

Wird nun das Tool lupdate auf das Projekt ausgeführt, so wird die Übersetzungsdatei erstellt bzw., sofern bereits vorhanden, aktualisiert.

Mit dem Programm Qt Linguist lassen sich diese Übersetzungsdateien öffnen und die Texte übersetzen.



Qt Linguist

Das Programm bereitet dafür die XML-Übersetzungsdateien in einer GUI auf, ordnet übersetzbaren Text nach Kontext, üblicherweise Klassen, in denen sie vorkommen, und zeigt für jeden Text den Originaltext, den Entwickler Kommentar, sowie den übersetzten Text, sofern bereits vorhanden.
Ist auch der Quellcode vorhanden, so wird ebenfalls der Code an jener Stelle gezeigt, an der der zu übersetzende Text vorkommt, oder wenn dieser auf einer GUI-Oberfläche ist diese Oberfläche mit einer Markierung, an welcher Stelle der Text steht.

Übersetzungen können verschiedene Status annehmen. So können nicht vorhandene, vorläufige oder finale Übersetzungen eindeutig ausgemacht werden.
Dies erlaubt beispielsweise nach Änderungen im Code, welche beispielsweise die Bedeutung ändern, und einem lupdate auf das Projekt das einfache ausmachen von Änderungen in der Übersetzungsdatei, wodurch nur diese Texte überprüft und ggf. neu übersetzt werden müssen.

Die XML Übersetzungsdateien können also auch, beispielsweise an externe Übersetzer, versendet werden, wo sie ebenfalls mit dem Qt Linguist übersetzt werden können.

Mit dem Tool lrelease wird die Übersetzungsdatei dann in eine qm Datei umgewandelt, welche dann mit der Klasse QTranslator in das Programm eingebunden werden kann.

4.5 Einstellungsverwaltung

Mit der Klasse QSettings liefert Qt sogar eine Programmeinstellungsverwaltung. Bei Verwendung dieser muss man sich nicht mehr um die verschiedenen üblichen Konfigurationsorte und -arten auf den verschiedenen Betriebssystemen kümmern.
Dies wird von Qt erledigt.

Das Erzeugen eines Objektes der Klasse QSettings könnte folgendermaßen aussehen:

```
QSettings* settings = new QSettings("Organisation",
"Programmname");
```

Anschließend können einem Schlüssel Werte mit der Funktion setValue zugewiesen werden und mit der Funktion value wieder abgefragt werden.

4.6 Daten Ein-/Ausgabe

Qt bietet eine einheitliche Schnittstelle um Daten zu lesen, zu speichern und zu übertragen.

Die Klasse QIODevice bildet dabei die Basisklasse aller E/A-Klassen. [Wol07]

Neben den Klassen für die Datei-, Stream- und Socket-Funktionalitäten bietet die ebenfalls von QIODevice erbende Klasse QProcess die Möglichkeit andere Programme zu starten. Eine einfache Interprozess-Kommunikation zu implementieren ist dann, aufgrund der vorhandenen E/A-Funktionen, sehr einfach.

4.7 SQL

Mit dem SQL-Modul wird eine Datenbank-unabhängige Schicht angeboten. [Wol07] Treiber sind für die Datenbanksysteme DB2, Interbase, MySQL, Oracle, ODBC, PostgreSQL, SQLite sowie Sybase vorhanden.

Mit Hilfe des SQL-Moduls kann also nicht nur Plattform-unabhängig, sondern zusätzlich Datenbank-unabhängig programmiert werden.

4.8 Threads

Die Klasse QThread ermöglicht ein einfaches, plattformunabhängiges Programmieren von Threads, um Multi-Prozessor oder -Kern Systeme, oder wenn komplexere Aufgaben auf andere Threads ausgelagert werden sollen, besser zu unterstützen.

Um einen Thread zu implementieren, wird eine neue Klasse erzeugt die von QThread erbt und deren abstrakte Methode run implementiert.

Klassen für Mutexe, eine allgemeine sowie zwei speziellere, für Semaphore und eine QWaitCondition genannte Klasse sind zur Threadsynchronisation ebenfalls vorhanden.

4.9 Daten-Typen und -Klassen

Um die Plattformunabhängigkeit zu gewährleisten bietet Qt alle gängigen Datentypen und -Klassen als eigene Implementierung an. [Wol07]

So werden beispielsweise die Datentypen qint8, qint16, qint32 usw. verwendet und eine eigene Klasse QString ist vorhanden.

Die Funktionen der C++- und der STL-Pendants sind auch vorhanden

QString hat gegenüber der C++ Klasse eine Besonderheit: [Wol07]
Sie arbeitet mit einem 16-Bit Unicode Zeichensatz um beispielsweise auch asiatische Sprachen und Zeichen zu unterstützen.
Eine string entsprechende Klasse ist mit dem Namen QByteArray implementiert.

Mit QDate, QTime und QDateTime sind außerdem Klassen für die Zeit- und Datumsverarbeitung vorhanden.

4.10 Zeichnen, Grafik und Drucken

Neben dem Zeichnen auf Bildflächen können auch eigene GUI Widgets gezeichnet werden.

4.11 OpenGL

Neben den Qt-eigenen Zeichenfunktionen bietet Qt auch Zugriff auf die 3D-Beschleunigungs-API OpenGL.

Dies erlaubt es beispielsweise anspruchsvollere 3D Anwendungen anzuzeigen.

OpenGL selbst ist in Grafikkartentreibern meist integriert und, im Gegensatz beispielsweise zu DirectX, auf den gängigen Systemen verfügbar.

4.12 SVG

Das SVG Modul ermöglicht die Darstellung von SVG Grafiken (Vektor-Grafiken).

4.13 Debug

Eine Methode des Debuggens ist das Ausgeben von Zwischenwerten oder Hilfskommentaren um den Programmablauf verfolgen zu können. Diese Art des Debuggens wird durch die Funktion qDebug() unterstützt. [Wol07]

Um der Funktion qDebug QString oder QByteArray Objekte als C-konforme Strings zu übergeben wird die Funktion qPrintable verwendet, die die übergeben Text-Objekte konvertiert.

Beim Kompilierungsvorgang können dann, etwa für eine Release-Version, sämtliche Debug-Meldungen und -Befehle ignoriert werden.

Standardmäßig werden solche Debug Meldungen bei Unix und Mac an stderr gesandt, in Windows aber an einen Debugger.
Mit der Funktion qInstallMsgHandler lässt sich auch eine eigene Funktion angeben, an welche die Nachrichten gesandt werden, wodurch eigene Implementierungen, etwa das Schreiben in eine Log Datei, realisiert werden können.

4.14 Fenster Design

Mit dem Qt Creator lassen sich, in einer IDE üblichen Weise, Fenster-Oberflächen erstellen und als Design Datei (Dateiendung ui) abspeichern.

Diese Dateien werden dann, sofern in der Qt

Projektdatei angegeben, beim Übersetzungsvorgang von qmake in C++-Header-Dateien umgewandelt.

Zum erstellen des Fensters wird dann im Quellcode die Fensterklasse der Design/Layout Klasse übergeben, welche das Fenster mit dem passenden Aussehen erzeugt.

Zusätzlich wird im Namensraum Ui eine Klasse abgeleitet, um eine möglichst einfache Objekt-Erzeugung zu ermöglichen.

Die beste Methode [Wol07] ein solches Design zu verwenden ist die Fensterklasse von der Design Klasse erben zu lassen, damit einfacher auf die Klassenvariablen/-elemente zugegriffen werden kann, und im Konstruktor die Funktion auszuführen, welche das Design anwendet.

4.15 Ressourcen

Qt hat, wie viele andere Frameworks[Wol07], ein eigenes Ressourcen System, welches das Einbetten von Ressourcen in den Programmcode erlaubt.
In einer Ressourcen-Datei (Dateieindung qrc) werden in XML Syntax die einzubettenden Dateien angegeben. Hier kann als Attribut auch die Sprache angegeben werden, welche später automatisch bestimmt wird und dann die entsprechende Datei geladen wird.

Im Programm-Quellcode kann mit der Pfadangabe ":/<Pfad>" auf die Dateien zugegriffen werden.

4.16 Unit Testing

Seit Version 4.1 bietet Qt auch ein einfaches Unit Testing.
Um auch GUI-Funktionalitäten testen zu können, können zusätzlich Maus und Tastatur simuliert werden.

5 Zusammenfassung

Qt ist schon lange keine einfache Grafikbibliothek mehr.
Neben der konsequenten Umsetzung einer plattformunabhängigen Bibliothek bietet Qt zahlreiche weitere Features.
Diese sind dabei in Module gegliedert und können einfach in das Projekt eingebunden werden.

Mit der Qt Creator IDE, Qt Linguist und qmake werden dem Entwickler außerdem mächtige Tools angeboten, die die Entwicklung mit Qt stark vereinfachen und unterstützen.

Spätestens mit dem Umstieg auf die LGPL Lizenz dürfte Qt für jedes C++ Projekt zumindest als eines der in der engeren Auswahl befindlichen Frameworks landen, egal ob mit oder ohne GUI, plattformab- oder unabhängig.

Literatur

[Wol07] Jürgen Wolf: Qt 4, GUI-Entwicklung mit C++
Erschienen bei Galileo Computing, 1. Auflage 2007

Johan Thelin: Foundations of Qt Development
Erschienen bei Apress, 2007

[QtR09] Qt 4.5 Reference Documentation – http://doc.trolltech.com/4.5/index.html
Stand Mai 2009

[QtS09] Offizielle Qt Software Homepage, http://www.qtsoftware.com/
Stand: Mai 2009