KDE-Programmierung

Burkhard Lehner
b_lehner@informatik.uni-kl.de

Version V1.2 – 30. Mai 1998

Inhaltsverzeichnis

1 Was ist KDE? Was ist Qt?

KDE (K Desktop Environment) ist ein Projekt, das es sich als Ziel gesetzt hat, eine einheitliche grafische Benutzeroberfläche für alle Applikationen zur Verfügung zu stellen, die auf möglichst vielen UNIX-Plattformen läuft. Es soll dadurch eine benutzerfreundliche, intuitive Bedienbarkeit erreicht werden.

Die wichtigesten Programme des KDE-Projektes sind:

- **`kwm`:** Der Window-Manager des KDE-Projektes erfüllt neben den normalen Aufgaben eines Window-Managers – wie z. B. Anzeigen einer Titelleiste für jedes Fenster, Bearbeitung der Buttons in der Fensterleiste (maximize, iconify) und der Erzeugung von Systemklängen – noch einige besondere Features: Er ist sehr komfortabel über grafische Dialogfenster konfigurierbar und ermöglicht ein Drag & Drop zwischen KDE-Applikationen. KDE-Applikationen lassen sich auch ohne kwm benutzen, jedoch evtl. mit eingeschränktem Funktionsumfang (kein Drag & Drop).
- **`kfm`:** Der File-Manager hat sich im Laufe der Entwicklung von KDE zu einem extrem mächtigen Werkzeug entwickelt. Neben den normalen Aufgaben – wie z. B. Verschieben, Kopieren, Löschen von Dateien und Verzeichnissen, Verwalten des Papierkorbes, Bearbeiten von Dateiattributen – ersetzt er inzwischen sogar einen ftp- und einen http-Client und stellt einen vollwertigen WWW-Browser zur Verfügung. Der kfm kann von Applikationen zum Laden von Dateien mit der Angabe einer beliebigen URL benutzt werden, so dass diese Applikationen sehr einfach Dateien aus dem Internet bearbeiten können. Auch der kfm muss nicht unbedingt gestartet werden, um KDE-Applikationen laufen zu lassen.
- **weitere Applikationen:**
  - `kab`: Adressbuchverwaltung
  - `kmail`: Mail-Programm
  - `kedit`: einfacher Texteditor
  - `knotes`: Notiz-Verwaltung
  - `kzip`: Verwaltung von gepackten Dateien
  - `kvt`: grafische Terminal-Emulation
  - `kcontrol`: komfortables grafisches Tool zur Einstellung der KDE-Konfiguration
  - `kdehelp`: Applikation zur Anzeige von KDE-Hilfedateien (im HTML-Format)

…und viele andere mehr. Fast täglich erscheinen neue Applikationen für das KDE-Projekt im Internet. Nahezu alle bisher im KDE-Projekt veröffentlichten Applikationen sind unter der General Public Licence (GPL bzw. LGPL) veröffentlicht, daher sind Programme und Quellcodes frei verfügbar.

Die Grundlage für die grafische Benutzeroberfläche für KDE-Programme ist die Bibliothek Qt der Firma Troll-Tech in Norwegen. Qt stellt eine sehr komfortable und mächtige Schnittstelle zur XLib dar und enthält eine große Anzahl an vordefinierten Widgets[1] wie z. B. Buttons, Input-Felder, Menüs, Combo-Boxen, Schieberegler, Tabellen und vieles andere mehr. Außerdem enthält es eine Reihe von sehr nützlichen C++-Klassen und Templates, die einem Programmierer das Leben erleichtern, z. B. Strings, dynamische Arrays, Listen, Hash-Tabellen.

Qt ist ein kommerzielles Produkt, Lizenzen verkauft die Firma Troll Tech auf Developer-Basis: Für $1200 kann ein Programmierer eine Lizenz für UNIX-Systeme erhalten, und darf dann alle von ihm mit Qt entwickelten Programme verkaufen, ohne dass weiter Lizenzgebühren fällig werden. Er darf dabei sowohl statisch als auch dynamisch gelinkte Programme vertreiben. Speziell für den Free Software-Bereich bietet Troll Tech aber auch die Qt Free Edition an: Man kann das vollwertige Qt kostenlos als Binaries für verschiedenen Unix-Plattformen oder als Quelltext(!) bei Troll Tech downloaden und benutzen, mit zwei Einschränkungen:

- Qt darf nur unverändert weitergegeben werden. Troll Tech behält sich also insbesondere das Recht auf Bugfixes vor (um Versionschaos zu vermeiden).
- Mit der Qt Free Edition entwickelte Programme dürfen nur als freie Software weiterverbreitet werden, z. B. unter der GPL. Für eine kommerzielle Nutzung ist in jedem Fall eine Developers Licence nötig.

Eine Anmeldung oder Registrierung bei Troll Tech ist für die Benutzung der Qt Free Edition nicht nötig.

Qt ist also **keine** freie Software, darf aber zur Entwicklung von freier Software kostenlos genutzt werden. Die genauen Lizenzbedingungen, die sehr ausführliche und übersichtliche Dokumentation sowie ein Tutorial für Einsteiger sind unter `http://www.troll.no/` erhältlich. Unter dieser URL ist auch ein Download der Qt Free Edition möglich.

Als KDE-Programmierer sollte man sich also in folgende Dinge einarbeiten:

C++: KDE und Qt sind in C++ geschrieben und machen intensiven Gebrauch von der Klassenstruktur. Als KDE-Programmierer sollte man also neben C-Kenntnissen auch die Objektorientierung von C++, insbesondere Klassendefinitionen und Vererbung, kennen. Kenntnisse über Templates, virtuelle Methoden und Operatorüberladungen sind auch recht nützlich.

[1] Elemente für ein grafisches User-Interface

Qt-Klassen: Man sollte sich einen guten Überblick über die von Qt zur Verfügung gestellten Klassen und Templates verschaffen, um alle Möglichkeiten von Qt nutzen zu können und nicht das Rad neu erfinden zu müssen.

KDE-Klassen: KDE stellt einige spezielle Klassen zur Verfügung. Neben weiteren Widgets gibt es Klassen zum komfortablen Einlesen und Ändern von KDE-Konfigurationsdateien, sowie den Zugriff auf den kwm und den kfm.

Kenntnisse über den Aufbau und die Funktionsweise von X-Servern und der XLib sind nicht nötig, können aber vielleicht manchmal hilfreich sein. Kenntnisse über Makefiles sind oft nützlich.

Für Ihre eigenen Versuche benötigen Sie das Paket Qt (zur Zeit in der Version 1.33) in der Developers-Version (!), sowie mindestens die KDE-Pakete kdebase, kdelibs, kdesupport und kdesdk (zur Zeit in der Version Beta-4). Diese finden Sie zum Beispiel auch auf der CD zum Linuxtag 1998.

2 Hello, World!

Folgendes Programm stellt wahrscheinlich gleichzeitig eines der kürzesten KDE-Programme und eines der komfortabelsten "Hello, World"-Programme dar.

```
// khelloworld.cpp

// Hello World for KDE
// by Burkhard Lehner <b_lehner@informatik.uni-kl.de>

#include <kapp.h>
#include <kmenubar.h>
#include <ktopwidget.h>
#include <qlabel.h>
#include <qkeycode.h>

int main (int argc, char **argv) {

  const char *aboutText = "KDE Hello World\nby Burkhard Lehner";

  KApplication a (argc, argv);
  KTopLevelWidget top;

  QPopupMenu *filePopup = new QPopupMenu ();
  filePopup->insertItem (i18n ("&Quit"), &a, SLOT (quit ()), CTRL + Key_Q);

  KMenuBar *menu = new KMenuBar (&top);
  menu->insertItem (i18n ("&File"), filePopup);
```

```
24.   menu->insertSeparator ();
25.   menu->insertItem (i18n ("&Help"), a.getHelpMenu (TRUE, aboutText));
26.
27.   top.setMenu (menu);
28.
29.   QLabel *text = new QLabel (i18n ("Hello, World"), &top);
30.   text->setAlignment (AlignCenter);
31.   text->setFont (QFont ("charter", 64, QFont::Bold, TRUE));
32.   text->setMinimumSize (text->sizeHint ());
33.   top.setView (text);
34.
35.   top.show ();
36.   return a.exec ();
37. }
```

Kurze Erläuterung zum Quellcode:

Zeilen 6 - 10: Alle benutzen Klassen von KDE und Qt müssen durch Includen der entsprechenden Header-Dateien definiert werden. Die Header-Dateien der KDE-Klassen beginnen in der Regel mit einem ‘k’, die der Qt-Klassen mit einem ‘q’. Die Header-Dateien sind in diesem Fall zuständig für die Definition der Klassen `KApplication`, `KMenuBar`, `KTopLevelWidget` und `QLabel`, die fünfte Header-Datei definiert die Konstanten `CTRL` und `Key_Q` für Zeile 20.

Zeilen 16, 17: Ein KDE-Programm besitzt meist eine Instanz der Klasse `KApplication` und eine oder mehrere Instanzen der Klasse `KTopLevelWidget`. `KApplication` übernimmt die Initialisierung aller KDE-internen Daten (Einlesen der applikationsspezifischen Konfigurationsdatei, Erstellen der Übersetzungstabelle für länderspezifische Menüs, Analysieren der KDE und X betreffenden Kommandozeilenparameter). Bevor diese internen Daten nicht angelegt sind, sollte auf kein KDE- oder Qt-Objekt zugegriffen werden. Vorsicht daher mit statischen Objekten: deren Konstruktoren werden *vor* der `main`-Funktion ausgeführt! `KTopLevelWidget` stellt das Applikationsfenster dar, in das dann Menüleisten, Statusleisten und weitere Widgets eingefügt werden können.

Zeile 19, 20: Hier wird ein neues Popup-Menü erzeugt, das erscheinen soll, wenn im Hauptmenü auf `File` geklickt wird. Dieses Popup-Menü enthält nur einen Eintrag `Quit`. Dieser Eintrag wird mit dem Slot `quit ()` des Objektes `a` (also des KApplication) verbunden. Eine Auswahl dieses Menüpunktes bewirkt also ein Beenden des Programms. Für diesen Menüpunkt gibt es noch den globalen Shortcut `Strg-Q`. Die Eingabe von `Strg-Q` bewirkt also ebenfalls die sofortige Beendigung des Programms.

Zeile 22 - 25: Es wird die Menüleiste erzeugt, die zwei Einträge `File` und `Help` enthält. Unter `File` wird das Popup-Menü aus den Zeilen 16 und 17 aufgerufen, unter `Help` das Standard-Helpmenü. Dieses enthält drei Punkte:

Contents: Aufruf dieses Menüpunktes führt zum Aufruf des Programms kdehelp, das die HTML-Hilfedatei für diese Applikation anzeigt.

About *Applicationname*: Zeigt in einem kleinen Fenster den About-Text an, der in Zeile 14 definiert wurde

About KDE: Zeigt in einem Fenster allgemeine Informationen zu KDE.

Zeile 27: Fügt die Menüleiste in das Hauptfenster ein.

Zeile 29 - 33: Ein Label-Widget wird erzeugt, das zur Anzeige von Text dient. Dieses Label enthält den Schriftzug "Hello, World" (Zeile 29), stellt ihn zentriert dar (Zeile 30) mit dem Zeichensatz "Carter" in Größe 64 Punkt, fett und kursiv (Zeile 31), und ist immer mindestens so groß, dass der ganze Text dargestellt werden kann (Zeile 32). Dieses Widget wird als Hauptfenster im `KTopLevelWidget` eingetragen (Zeile 33).

Zeile 35: Bisher wurde von diesen Widgets auf dem Bildschirm noch nichts angezeigt. Diese Zeile stellt jetzt das `KTopLevelWidget` sowie seine Child-Widgets (die Menüleiste und das Label-Widget) auf dem Bildschirm dar.

Zeile 36: Die Event-Loop der Applikation wird gestartet. Ab diesem Zeitpunkt werden Mausbewegungen, Tastatureingaben und so weiter verarbeitet. Die Event-Loop wird erst verlassen, wenn das Programm über den Slot `quit ()` der `KApplication` beendet wird.

Die Funktion `i18n (char *)`: Diese Funktion (genaugenommen ein Makro als abkürzende Schreibweise von `KApplication::getKApplication()->getLocale()->translate(char *)`) übersetzt den übergebenen String in einen String der gleichen Bedeutung in der für KDE eingestellen Sprache. Hat man in KDE beispielsweise als Sprache "deutsch" ausgewählt, so werden die Menüeinträge `Datei` und `Hilfe` heißen, und der angezeigte Text "Hallo, Welt". Woher die korrekten Übersetzungen kommen – schließlich enthält KDE keinen automatischen Übersetzer mit künstlicher Intelligenz – wird im Punkt "4.3 Vielsprachige Programme mit KLocale und i18n" besprochen. Es empfiehlt sich, alle Texte, die angezeigt werden sollen, mit dieser Funktion zu übersetzen. i18n ist übrigens die Abkürzung für "Internationalisation", die 18 steht dabei für die 18 Buchstaben, die ausgelassen worden sind.

3 Überblick Qt

3.1 Bereitgestellte Klassen

Hier eine kleine Auswahl der wichtigsten Klassen und Templates:

Allgemeine Klassen / Templates:

- `QList`, `QQueue`, `QStack`, `QStrList`: Listen
- `QString`: Zeichenketten (bietet unter anderem Suchen/Ersetzen von regulären Ausdrücken)
- `QDict`: Hash-Tabellen
- `QBitArray`: dynamisches Array von einzelnen Bits

Klassen für Plattformunabhängigkeit:

- `QFileInfo`: Informationen über Filegröße, Attribute, Änderungsdatum ...
- `QDir`: Auslesen von Inhaltsverzeichnissen
- `QDate`, `QTime`, `QDateTime`: Zeit- und Datumsangaben

Klassen für X-Strukturen:

- `QColor`, `QColorGroup`: Bildschirmfarben
- `QCursor`: Mauscursor
- `QFont`, `QFontInfo`, `QFontMetrics`: Strukturen und Informationen über Fonts
- `QEvent`: Struktur für X-Events, davon abgeleitet Unterklassen für die verschiedenen Eventtypen: `QCloseEvent`, `QFocusEvent`, `QMouseEvent`, `QKeyEvent`, `QPaintEvent`, `QTimerEvent`, ...

- `QPen`: Klasse für Liniendicken und -muster

Klassen für elementare Zeichenroutinen:

- `QPainter`: mächtige Klasse zum Zeichnen von 2D-Primitiven in ein `QPaintDevice`. Zeichnen von Linien, Polygonen, Kreisen, Ellipsen, Bezier-Kurven, Text, auch transformiert (gedreht, geschert, gespiegelt).
- `QPaintDevice`: abstrakte Klasse, auf der Zeichenoperationen ausgeführt werden können. Davon abgeleitete Klassen u.a.:
  - `QPixmap`: Pixmap auf dem X-Server
  - `QBitmap`: 2-Farben-Pixmap
  - `QWidget`: Bildschirmrechteck (Window auf X-Server-Basis); von dieser Klasse sind auch die meisten GUI-Elemente abgeleitet
  - `QPrinter`: Drucker
- `QImage`: Schnelle Pixmap auf Client-Seite, direkter Pixelzugriff
- `QMovie`: Animated GIFs

Klassen für GUI-Elemente:

- `QLabel`: Einfaches Textelement
- `QLCDNumber`
- `QProgressbar`
- `QButton` (davon abgeleitet: `QPushButton`, `QCheckButton`, `QRadioButton`)
- `QPopupMenu`
- `QScrollBar`
- `QSlider`
- `QComboBox`: Auswahlmenü
- `QListBox`: Auswahlbox
- `QLineEdit`: einzeilige Texteingabe
- `QMultiLineEdit`: einfacher, mehrzeiliger Texteditor

- `QDialog`: allgemeine Klasse für Bildschirmdialoge (davon abgeleitet: `QFileDialog`, `QMessageBox`)

weitere Klassen:

- `QLayout`: Klasse für automatische Anordnung (und Anpassung) von GUI-Elementen (davon abgeleitet: `QBoxLayout` für übereinander- oder nebeneinanderliegende Elemente, `QGridLayout` für tabellenartige Anordnung)
- `QToolTip`: Klasse für Tooltips (auch Balloon Help genannt)
- `QAccelerator`: Klasse für direkte Tastaturbefehle (z. B. F-Tasten, ...)

Eine sehr ausführliche Referenz über alle Qt-Klassen finden Sie auf der Homepage der Firma Troll-Tech (`http://www.troll.no/`) sowie in der Dokumentation des Qt-Pakets (Developers Version).

3.2 Signal-/Slot-Konzept

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 ein Scrollbar verschoben wird.

Callback-Routinen in Qt werden durch das sehr mächtige Signal-/Slot-Konzept realisiert. Dazu sind in den Klassendefinitionen der Qt-Klassen zusätzliche Methoden als `signal` oder `slot` deklariert. Diese Methoden können beliebige Parameteranzahlen und Typen haben, der Rückgabewert muss aber in jedem Fall `void` sein. Verwechseln Sie nicht Signal-Methoden von Qt mit Signalen des UNIX-Kernels, wie z. B. `SIGTERM` oder `SIGKILL`. Diese beiden Dinge haben nichts miteinander zu tun. Damit eine Klasse Signal- und Slot-Methoden benutzen kann, muss sie von der Klasse `QObject` abgeleitet sein. (Das sind z. B. alle GUI-Widgets, da sie von `QWidget` abgeleitet sind, welches seinerseits von `QObject` abgeleitet ist.)

Die Slot-Methoden sind Methoden, die eine Aktion als Folge einer Interaktion ausführen sollen. Sie müssen vom Programmierer mit Code gefüllt werden. Sie können weiterhin als `private`, `protected` oder `public` deklariert werden, ganz analog zu anderen Methoden, und können auch wie ganz normale Methoden aufgerufen werden.

Signal-Methoden werden vom Programmierer nur deklariert, jedoch nicht mit Code gefüllt. Der Code für die Methode wird vom Meta-Object-Compiler (`moc`) generiert und nachher zum Programm hinzugelinkt. Signal-Methoden werden innerhalb der Klasse aufgerufen, um anzuzeigen, dass eine Interaktion erfolgt ist, die eventuell eine Reaktion hervorrufen soll. Signal-Methoden führen keine Aktionen aus, sondern rufen nur Slot-Methoden auf, mit denen sie verbunden sind.

Um ein Signal mit einem Slot zu verbinden, benutzt man die Methode `connect`. Die einfachste Form der `connect`-Methode hat folgende Gestalt:

```
bool QObject::connect (const QObject * sender, const char * signal,
                       const QObject * receiver, const char * member)
```

Die Parameter sind dabei:

sender: Objekt, dessen Signal-Methode verbunden werden soll

signal: Name der Signal-Methode

receiver: Objekt, dessen Slot-Methode beim Signal aufgerufen werden soll

member: Name der Slot-Methode

Die Signal- und die Slot-Methode müssen in Parameteranzahl und -typen übereinstimmen.

Ein einfaches Beispiel soll zunächst erläutern, wie Signals und Slots verbunden werden:

```
    QScrollBar *scroll = new QScrollBar;
    QLabel     *label  = new QLabel;
    QObject::connect(scroll, SIGNAL(valueChanged(int)),
                     label,  SLOT(setNum(int)));
```

Die Klasse `QScrollBar` enthält eine Signal-Methode `void valueChanged (int)`, die aufgerufen wird, sobald die Position des Scrollbars sich geändert hat. Die Klasse `QLabel` besitzt eine Slot-Methode `void setNum (int)`, mit der man den Labeltext auf eine Zahl setzen kann.

Durch den Aufruf der Methode `connect` werden diese nun verbunden. Die Makros `SIGNAL` und `SLOT` übernehmen dabei die Umsetzung der Methodennamen in korrekte Zeichenstrings und müssen in der Methode `connect` **immer** benutzt werden. Nachdem Signal und Slot miteinander verbunden sind, führt jede Veränderung der Scrollbarposition durch den Benutzer zu einem Aufruf des Signals `valueChanged`, das alle damit verbundenen Slots aufruft, in diesem Fall also `setNum`, wodurch sich der Wert im Label ändert.

Ein Signal kann mit einer beliebigen Anzahl von Slots verbunden sein. Insbesondere kann sein Signal auch unverbunden bleiben. Verbindungen können mit der Methode disconnect wieder gelöst werden. 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.

Hier nun ein Beispiel einer selbstdefinierten Klasse, die einen Slot definiert:

```
class myObject : QObject {

    Q_OBJECT            // Dieses Makro definiert Signals und Slots

public:
   myObject (QObject * parent=0, const char * name=0);
   ~myObject ();

public slots:
    void printMessage () {printf ("Knopf gedrückt!\n");}
}
```

Ist diese Klassendeklaration z. B. in der Datei myObject.h enthalten, so ist anschließend der Meta Object Compiler für dieses Header-File aufzurufen:

```
% moc myObject.h -o m_myObject.cpp
```

Dieser Aufruf erzeugt eine Datei m_myObject.cpp, die anschließend noch compiliert und zur ausführbaren Datei hinzugelinkt werden muss. (Alternativ kann man auch in der Datei, in der die Methoden der Klasse definiert sind, eine Zeile #include "m_myObject.cpp" einfügen.)

Eine Verbindung sieht dann z. B. so aus:

```
QPushButton *button;
myObject *obj;
QObject::connect (button, SIGNAL (clicked ()),
                  obj,    SLOT (printMessage ()));
```

Anschließend wird bei jedem Druck auf die Taste der Text "Knopf gedrückt" ausgegeben.

4 Überblick KDE

4.1 Bereitgestellte Klassen

Neben den Klassen, die Qt dem Programmierer zur Verfügung stellt, enthält auch das KDE-Paket eine Reihe von Klassen, von denen hier einige stichwortartig vorgestellt werden sollen:

Nicht-grafische Klassen (KDE-core):

- `KApplication`: Übernimmt die Initialisierung wichtiger Strukturen. Jede KDE-Applikation muss genau eine Instanz dieser Klasse enthalten.
- `KLocale`: Übersetzung von Meldungen, Menüeinträgen usw. in die eingestellte Landessprache
- `KIconLoader`: Einlesen von Icons, diese werden für weitere Zugriffe gecacht
- `KProcess`: Aufruf anderer Programme
- `KSocket`, `KServerSocket`: Komfortable TCP/IP-Sockets.
- `KWM`: Kommunikation mit dem Windowmanager (nicht unbedingt `kwm`)
- `KConfig`: Zugriff auf Konfigurationsdateien (`rc`-Dateien)

Grafische Klassen / Widgets (KDE-UI):

- `KTopLevelWidget`: Hauptfenster der Applikation
- `KColorDialog`, `KFontDialog`, `KMsgBox`: vordefinierte Dialoge zur Auswahl von Farben, Fonts und einfache Meldungen
- `KDNDWidget`: Widget für Drag & Drop
- `KDatePicker`: Auswahl eines Datums
- `KLEDLamp`: Statuslampe
- `KPanner`, `KNewPanner`: Unterteilung eines Widgets in zwei Teilwidgets mit verschiebbarer Trennlinie
- `KMenuBar`, `KToolBar`, `KStatusBar`: Menü- und Statusleisten
- `KTreeList`: Darstellung hierarchischer Strukturen (Verzeichnisbäume etc.)
- `KEdit`: Einfacher Texteditor, mit Zeilenumbruch, Datei speichern, laden, einfügen, Suchen und Ersetzen, ...

Weitere Widgets (nicht im KDE-Paket enthalten):

- `KFM`: Klasse zur Kommunikation mit einem laufenden kfm, Laden von Dateien im Hintergrund (auch über das Netzwerk mit ftp oder http)

- `KDirDialog, KFileDialog, ...`: Dialog zur Eingabe einer URL, mit Vervollständigung
- `KTextObject`: Widget zur Darstellung von RTF-Dateien (incl. editieren)
- `KHTMLWidget`: Widget zur Darstellung von HTML-Dateien (sehr mächtig)

4.2 Compilieren mit dem configure-Skript

Jeder, der schon einmal eine KDE-Applikation compiliert hat, kennt den Dreisatz:

```
% ./configure
% make
% make install
```

Der Aufruf des Shell-Skripts `configure` bestimmt die compilerrelevanten Daten des Systems, wie z. B. den Compiler, die KDE- und Qt-Pfade. Anschließend werden mit diesen Daten aus den Dateien `Makefile.in` die Dateien `Makefile` in allen Unterverzeichnissen der Applikation erstellt. Der Aufruf von `make` compiliert nun alle Source-Dateien und linkt sie zur einer ausführbaren Datei zusammen. `make install` kopiert schließlich alle Dateien an die richtige Stelle in der Verzeichnisstruktur.

Will man als Programmierer ebenfalls diese Methode übernehmen (was sehr anzuraten ist!), so lädt man sich am besten das Paket `kdesdk` (KDE Software Development Kit) vom KDE-FTP-Server und kopiert die darin enthaltene Beispielapplikation `khexdit` in ein eigenes Verzeichnis. Weiterhin ist es empfehlenswert, sich `automake` und `autoconf` von GNU zu installieren. In das Verzeichnis `khexdit` (das man am besten an den Namen seiner Applikation anpasst) kopiert man nun die eigenen Sourcedateien. Anschließend passt man in allen Unterverzeichnissen die Dateien `Makefile.am` an sowie die Datei `configure.in` im Hauptverzeichnis (hier wird angenommen, dass Ihre Applikation den Namen `kapplication` hat):

`./Makefile.am`: Hier sind nur alle Unterverzeichnisse einzutragen. Solange Sie noch kein `i18n` benutzen (siehe 4.3), lassen Sie einfach das Unterverzeichnis `po` weg.

`./kapplikation/Makefile.am`: Ändern Sie `APPSDIR` auf das entsprechende Unterverzeichnis, um die Kategorie Ihrer Applikation festzulegen. Zur Zeit sind folgende Kategorien vorgesehen: Applications, Games, Graphics, Internet, Multimedia, System und Utilities. Solange Sie noch keine Hilfedateien vorbereitet haben, löschen Sie `doc` von `SUBDIRS`. Im unteren Teil tragen Sie die Namen der Sourcedateien ein, wie angegeben.

./configure.in: In `AM_INIT_AUTOMAKE` tragen Sie den Namen und die Versionsnummer Ihrer Applikation ein. In `AC_OUTPUT` tragen Sie alle `Makefiles` ein, die erzeugt werden sollen, das sind in diesem Fall alle `Makefiles`, zu denen Sie das `Makefile.am` geändert haben, also mindestens `Makefile` und `kapplication/Makefile`.

Anschließend können Sie `make -f Makefile.dist` aufrufen. Dadurch wird das Skript `configure` aus der Datei `configure.in` erzeugt.

Nun sollten Sie noch drei Dateien erzeugen im Verzeichnis `./kapplication`:

- `kapplication.xpm` und `mini-kapplication.xpm`: Icons für den `kwm`, die Ihrer Applikation zugeordnet werden
- `kapplication.kdelnk`: Informationen über die Applikation

Die beiden Icons können Sie am besten vom KDE-Icon-Team erstellen lassen. Schreiben Sie dazu eine E-Mail an Mark Donohoe unter `donohoe@kde.org`, und beschreiben Sie ihm Ihre Applikation und was Sie sich auf dem Icon vorstellen.

Die `kdelnk`-Datei können Sie leicht aus dem Beispiel in `kexample` ableiten.

Danach können Sie mit `./configure`, `make` und `make install` testen, ob das Compilieren und Installieren funktioniert. Eventuell brauchen Sie für das `make install` Superuser-Rechte.

Wenn Ihre Applikation soweit lauffähig ist und Sie sie der Welt präsentieren möchten, so können Sie ganz einfach ein Source-Paket im .tar.gz-Format erzeugen lassen, indem Sie im Hauptverzeichnis Ihrer Applikation `make dist` eingeben. Es wird dann das File `kapplication-version.tar.gz` erzeugt. Sie sollten jedoch vor der Weitergabe einmal testen, ob in dem Archiv auch alle benötigten Dateien enthalten sind. Sollten einige Dateien fehlen (häufig zum Beispiel Header-Dateien), so fügen Sie diese im File `./kapplication/Makefile.am` in `noinst_HEADERS` ein.

4.3 Vielsprachige Programme mit KLocale und i18n

KDE bietet eine komfortable Methode für Programmierer, die Programme in vielen verschiedenen Sprachen verfügbar zu machen. Die zentrale Rolle übernimmt dabei das Makro `i18n`, das bereits im "Hello, World"-Programm in Kapitel 2 verwendet wurde. Wenn sie das `configure`-Skript aus Kapitel 4.2 benutzen, wird ihnen nochmals viel Verwaltungsarbeit abgenommen.

Gehen Sie folgendermaßen vor, um Ihr Programm zu internationalisieren:

1. Umgeben Sie alle Zeichenketten, die in Ihrem Programm benutzt werden und die übersetzt werden sollen, mit dem Makro `i18n ()`. Das kann dann z. B. so aussehen:

vorher:	`printf (Result of calculation: %d\n", res);`
nachher:	`printf (i18n (Result of calculation:%d\n"), res);`

   Die Namen von Einträgen und Gruppen in Konfigurationsdateien sollten **nicht** übersetzt werden, da es sonst Probleme beim Einlesen der Konfigurationsdateien gibt, falls die Landessprache gewechselt wird.

2. Rufen Sie im Verzeichnis Ihrer Applikation `make messages` auf. Dadurch werden alle Zeichenketten, die übersetzt werden sollen, in die Datei `messages.po` geschrieben. Achtung: In der aktuellen Version von `kexample` (KDE Beta-4) wird noch ein älteres Makro `translate` statt `i18n` herausgefiltert. Ändern Sie bitte in der Datei `Makefile.am` im Applikationsverzeichnis folgende Zeile:

   ```
   $(XGETTEXT) -C -ktranslate $(khexdit_SOURCES)
   ```

   um in:

   ```
   $(XGETTEXT) -C -ki18n $(khexdit_SOURCES)
   ```

   Am Zeilenanfang **muss** ein Tabulator stehen!

3. Kopieren Sie die Datei `messages.po` in das Verzeichnis `po` und benennen Sie es um in `kapplication.pot`, wobei `kapplication` wieder der Name ihrer Applikation ist.

4. Falls Sie die Übersetzungen zum ersten mal vornehmen, so kopieren Sie die Datei außerdem noch in das gleiche Verzeichnis mit den Namen *sprache*`.po`, wobei *sprache* das Länderkürzel der unterstützen Sprache darstellt. Wollen Sie beispielsweise Übersetzungen ins Deutsche und Norwegische anbieten, so kopieren Sie die Datei noch mit den Namen `de.po` und `no.po`. Falls Sie bereits einmal Übersetzungen gemacht haben und nun lediglich die Übersetzungstabelle aktualisieren wollen (da Sie beispielsweise neue Buttons oder Menüpunkte in Ihr Programm eingefügt haben), so führen Sie statt dessen im Verzeichnis `po` ein `make merge` aus. Dadurch werden neue zu übersetzende Texte in die Dateien eingefügt.

5. Übersetzen Sie nun die Texte in den `.po`-Dateien. Die Texte stehen dabei in Paaren von Text und Übersetzung. Im "Hello, World"-Programm aus Kapitel 2 sieht die `messages.po`-Datei zum Beispiel so aus:

   ```
   # SOME DESCRIPTIVE TITLE.
   # Copyright (C) YEAR Free Software Foundation, Inc.
   # FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
   #
   #, fuzzy
   msgid ""
   msgstr ""
   "Project-Id-Version: PACKAGE VERSION\n"
   ```

```
"POT-Creation-Date: 1998-05-18 21:49+0200\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: ENCODING\n"

#: khelloworld.cpp:18
msgid "&Quit"
msgstr ""

#: khelloworld.cpp:21
msgid "&File"
msgstr ""

#: khelloworld.cpp:23
msgid "&Help"
msgstr ""

#: khelloworld.cpp:27
msgid "Hello, World"
msgstr ""
```

Tragen Sie zunächst im oberen Teil die Informationen zum Programm ein. Anschließend schreiben Sie zwischen die Anführungsstriche hinter einem `msgstr`-Eintrag die Übersetzung des vorangegangenen `msgid`-Eintrags. Achten Sie dabei darauf, dass Sie das Format möglichst genau beibehalten. Im Beispiel von oben sähe eine Übersetzung dann z. B. so aus:

```
msgid "Result of calculation: %d\n"
msgstr "Ergebnis der Berechnung: %d\n"
```

Einträge mit der Ergänzung `#, fuzzy` (außer dem Header-Eintrag) sind neu hinzugekommene Texte. Es wurde dabei versucht, die Übersetzung aus einem älteren Übersetzungstext zu erraten, was jedoch oftmals nicht ganz richtig ist. Wenn Sie auf einen solchen Eintrag stoßen, so korrigieren Sie evtl. den Eintrag in `msgstr` und löschen Sie die Zeile `#, fuzzy`. (Aber nicht im Header der `.po`-Datei!)

6. Nachdem Sie alle `.po`-Dateien entsprechend bearbeitet haben, sollte Ihr Programm alle angeboteten Landessprachen unterstützen. Nachdem Sie ein `make` und ein `make install` gemacht haben, können Sie es aufrufen und testen, ob alle Texte korrekt übersetzt wurden.

Da nicht jeder Programmierer 20 oder mehr Sprachen sprechen kann, gibt es im KDE-Projekt ein Übersetzungsteam. Um davon Gebrauch zu machen, schicken Sie einfach Ihr `messages.po`-File an Pedro Morais <`pmmm@camoes.rnl.ist.utl.pt`>. Sie werden dann nach und nach die übersetzten Versionen erhalten, die Sie zu Ihrem Programm hinzufügen können.

Bei weiteren Fragen zum Thema Internationalisation (i18n), lesen Sie eine ausführlichere Anleitung unter `http://www.kde.org/i18n.html`, oder wenden Sie sich an die `kde-i18n` Mailing-Liste (`kde-i18n@kde.org`, vorübergehend auch `kde-i18n-doc@kde.org`).

5 Kontaktadressen

5.1 WWW-Seiten

KDE allgemein:	`http://www.kde.org/`
KDE Developers Center:	`http://www.ph.unimelb.edu.au/~ssk/kde/devel/`
KDE i18n:	`http://www.kde.org/i18n.html`
KDE Standards:	`http://www.kde.org/standards.html`
Qt:	`http://www.troll.no/`

5.2 FTP-Server

KDE:	`ftp://ftp.kde.org/`
KDE-upload:	`ftp://upload.kde.org/`

5.3 Mailing-Listen

kde:	Allgemeines Diskussionsforum
kde-announce:	Ankündigungen neuer Versionen und Applikationen
kde-user:	Hilfestellungen für Anwender
kde-devel:	Hilfestellungen für Programmierer (post-Rechte müssen erst beantragt werden bei Martin Konold <konold@kde.org>)
kde-licensing:	Diskussion über rechtliche Lizenz-Fragen
kde-look:	Diskussionen über die äußere Gestalt von KDE

Nähere Informationen zum Subscriben in den Listen finden Sie unter `http://www.kde.org/contact.html`. Alte Artikel finden Sie im Archiv unter `http://lists.kde.org/`.