Programiranje grafičnih vmesnikov s knjižnico QT

Milan Pulko, Borko Bošković, Janez Brest

Inštitut za računalništvo
Fakulteta za elektrotehniko, računalništvo in informatiko
Univerza v Mariboru
E-pošta: milan@mladika.si, borko.boskovic@uni-mb.si, janez.brest@uni-mb.si

GUI programming with QT graphic library

In this paper, we present the Qt C++ graphic library that supports the development of multi-platform GUI applications with "write once, compile anywhere" approach. Using a single source, applications can be written for MS Windows, MS Windows CE, Mac OS X, Embedded Linux, Linux/Unix with X11. Qt has a unique inter-object communication mechanism called "signals and slots". Qt has also support for 2D and 3D graphics, internationalization, XML, etc.

1 Uvod

QT je grafična knjižnica za C++, ki ponuja hitro in enostavno rešitev za programiranje grafičnih uporabniških vmesnikov (GUI).

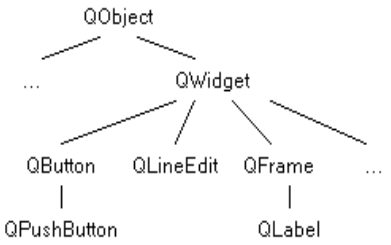
Pomembna značilnost knjižnice QT je popolna prenosljivost izvorne kode med različnimi operacijskimi sistemi MS Windows, MS Windows CE, Mac OS X, Embedded Linux in Linux /Unix z X11.

QT vsebuje tudi podporo za 2D in 3D grafiko (OpenGL), dostop do podatkovnih baz, XML, internacionalizacijo itd.

2 Opis

QT grafični uporabniški vmesnik sestavljajo grafični elementi imenovani widgeti. Ti predstavljajo okna (*QWidget*), gumbe (*QButton*), napise (*QLabel*), vnosna polja (*QLineEdit*), menije (*Qmenu*) itd.

Vsak tip widgeta je objekt, ki je izpeljan iz skupnega razreda *QWidget*. Ta pa je izpeljan iz skupnega razreda *QObject*.



Slika 1. Struktura QT objektov

Primer preproste GUI aplikacije, ki v oknu izpiše "Hello world", prikazuje program 1.

```
#include <QtGui>

int main(int argc, char *argv[]) {
  QApplication app(argc, argv);
  QWidget *okno = new QWidget();  // glavno okno
  okno->setWindowTitle("Main window ");  // naslov okna
  okno->setFixedSize(200, 120);    // dimenzije okna
  QLabel *label1 = new QLabel("Hello world", okno);
  label1->setGeometry(10, 10, 80, 30);
  okno -> show();
  return app.exec();    //  dogodkovna zanka
}
```

Program 1. "Hello world" aplikacija

Vsaka QT grafična aplikacija vsebuje konstruktor *QApplication*, ki ga kličemo s parametri *argc* in *argv* (argumenti ukazne vrstice iz funkcije *main*).

V našem primeru smo ustvarili widget, ki predstavlja glavno okno (*QWidget*). Če v konstruktorju widgetu ne določimo starša, se ta obnaša kot glavno okno. Widgeti, ki pa imajo starše določene, bodo prikazani znotraj območja staršev.

Ustvarili smo še widget za napis (*QLabel*), kateremu smo za starša določili widget *okno*. To pomeni, da bo prikazan znotraj tega widgeta. Z metodo *setGeometry* določimo x in y koordinati položaja widgeta znotraj območja starša ter njegovo širino in višino.

Novo ustvarjeni widgeti so ob nastanku vedno skriti, zato z metodo *show* naredimo widget okno in njegove potomce vidne.

Z *app.exec* zaženemo glavno zanko aplikacije (t.i. Event Loop). V tej zanki program čaka na dogodke (klik z miško, pritisk na tipko itd.).

Ker smo za ustvarjanje objektov (widgetov) uporabili ukaz *new* bi pričakovali, da na koncu programa sprostimo pomnilnik z *delete*. Vendar to ni potrebno, saj QT sam sprosti pomnilnik. Princip delovanja QT objektov je namreč takšen, da na koncu starši počistijo pomnilnik za svojimi potomci.

Postopek prevajanja:

1. Ustvarimo mapo z imenom projekta ter izvorne datoteke.

2. Ustvarimo od operacijskega sistema neodvisno projektno datoteko (.pro): *qmake -project*

Ustvarjena projektna datoteka vsebuje seznam vseh izvornih datotek programa ter dodatne nastavitve za prevajalnik.

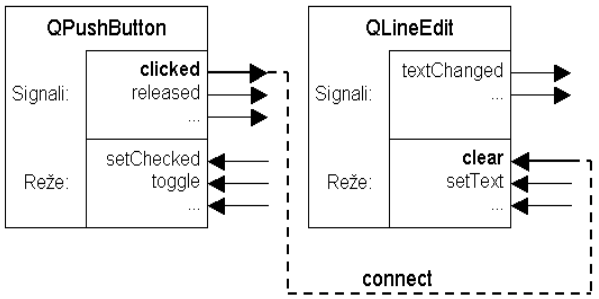
3. Ustvarimo od operacijskega sistema odvisno datoteko makefile: *qmake*

4. Prevedemo program v razhroščevalnem načinu ali kot končno verzijo: *make debug* oziroma *make release*

2.1 Signali in reže

Komunikacija med widgeti (odzivi na razne dogodke) poteka v QT-ju preko posebnega mehanizma imenovanega signali in reže (*signals and slots*).

Signali se prožijo ob posameznih dogodkih (npr. klik z miško). Reže pa so podobne običajnim C++ funkcijam, ki se kličejo kot odgovor na proženje z njimi povezanih signalov. Signale in reže povezujemo med seboj z ukazom connect: *connect(oddajnik, SIGNAL(signal), sprejemnik, SLOT(slot))*.



Slika 2. Mehanizem signalov in rež

Poleg povezovanja signalov in rež, lahko povezujemo tudi en signal z drugim. S tem dosežemo, da istočasno sprožimo dva signala: *connect(oddajnik_1, SIGNAL(sig_1), oddajnik_2, SIGNAL(sig_2))*.

Pri povezovanju signala z režo (oziroma z drugim signalom) morata imeti oba parametre istega tipa ter v istem zaporedju. Če ima signal slučajno več parametrov, kot jih zahteva z njim povezana reža, se dodatni parametri preprosto ignorirajo.

Vsak standardni widget ima že vnaprej definiran nabor signalov in rež. V večini primerov pa je ta nabor preozek. Zato običajno ustvarimo lastne (ang. *custom*) widgete ter jim definiramo reže (signale), ki se odzivajo na dogodke tako, kot smo si zamislili.

V nadaljevanju je predstavljen primer take aplikacije (program 2). Prikazani program ob vsakem kliku na *gumb1*, poveča vrednost števca za 1, ter to vrednost prikaže na widgetu *label1*.

Slika 3. Aplikacija *stevnik*

stevnik.h:

```
#include <QtGui>

class Stevnik : public QWidget {
  Q_OBJECT
public:
  Stevnik(QWidget *parent = 0);  // glavno okno
private slots:
  void napisi();
private:
  QPushButton *gumb1;
  QLabel *label1;
  int i;
};
```

Razred *Stevnik* dedujemo od razreda *QWidget*. Sledi makro *Q_OBJECT*, ki ga navedemo na začetku definicije razreda, v katerem bomo uporabljali mehanizem signalov in rež. Z rezervirano besedo *slots* definiramo režo *napisi*, ki jo v nadaljevanju implementiramo kot običajno metodo.

stevnik.cpp:

```
#include "stevnik.h"

Stevnik::Stevnik(QWidget *parent) {
  i=0;
  setFixedSize(200, 120);
  setWindowTitle("Stevec");
  gumb1= new QPushButton("OK", this);
  gumb1->setGeometry(50, 50, 80, 30);
  label1= new QLabel("0", this);
  label1->setGeometry(10, 10, 80, 30);
  connect(gumb1, SIGNAL(clicked()), this, SLOT(napisi()));
}

void Stevnik::napisi() {
  i++;
  label1->setText(QString::number(i));
}
```

V konstruktorju smo določili začetne vrednosti spremenljivk in geometrijo grafičnega vmesnika. Z ukazom *connect* smo nato povezali standardni signal *clicked(gumb1)* z novo ustvarjeno režo *napisi*. Ta povezava omogoči, da se ob kliku na *gumb1* kliče reža *napisi*.

Reža *napisi* ob vsakem klicu poveča vrednost števca in jo prikaže na widgetu *label1*. Metoda za izpis *setText*

zahteva parameter tipa *QString*, zato celoštevilčno vrednost števca pretvorimo v *QString* z ukazom *QString::number(i).*

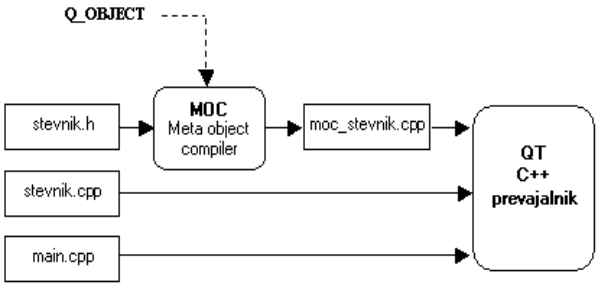
main.cpp:

```
#include <QtGui>
#include "stevnik.h"

int main(int argc, char *argv[]) {
  QApplication app(argc, argv);
  Stevnik st;
  st.show();
  return app.exec();
}
```

Program 2. Custom widget

Ker QT program uporablja nestandardne razširitve jezika C++ (makroje *Q_OBJECT*, *signals*, *slots*), se pred prevajanjem izvede poseben predprocesor MOC (Meta object compiler), ki ustvari standardno kodo primerno za prevajanje. S takim načinom je dosežena večja preglednost in prenosljivost izvorne kode med različnimi operacijskimi sistemi.



Slika 4. Prevajanje QT aplikacije

2.2 Podpora internacionalizaciji

QT vsebuje tudi podporo za prevod aplikacije v različne jezike. To izvedemo tako, da besede oz. fraze (meniji, napisi na gumbih, namigi itd.), katere nameravamo prevesti še v druge jezike, pišemo kot argumente metode *tr*.

Za podporo internacionalizaciji uporabljamo razred *QTranslator*.

Postopek prevajanja je prikazan na primeru programa 3, ki smo mu dodali podporo za slovenski in nemški jezik:

```
#include <QtGui>
#include <QTranslator>

int main(int argc, char *argv[]) {
  QApplication app(argc, argv);
  QTranslator prevod;

  QString jezik = QLocale::languageToString
    (QLocale::system().language());
  if (jezik=="Slovenian") {
    prevod.load("slo");  // naloži datoteko slo.qm
    app.installTranslator(&prevod);
  }
  if (jezik=="German")  {
    prevod.load("nem");  // naloži datoteko nem.qm
    app.installTranslator(&prevod);
  }

  QWidget *okno = new QWidget();
  okno->setWindowTitle(QObject::tr("Main window"));
  okno->setFixedSize(200, 120);
  QLabel *label1 = new QLabel(QObject::tr("Hello world", okno));
  label1->setGeometry(10, 10, 80, 30);

  okno->show();
  return app.exec();
}
```

Program 3. Internacionalizacija aplikacije

Z ukazom *QLocale::system().language()* smo ugotovili privzeti jezik operacijskega sistema (*Slovenian*, *German*, *English* itd.) ter naložili ustrezno .qm datoteko s prevodom. V kolikor datoteka ne obstaja bo uporabljen privzeti jezik.

Postopek samega prevajanja poteka tako, da v projektno datoteko .pro dodamo vrstico TRANSLATIONS, v kateri navedemo imena datotek, ki bodo vsebovala prevode v druge jezike. V našem primeru *slo.ts* (slovenščina) in *nem.ts* (nemščina):

TRANSLATIONS += slo.ts nem.ts

Nato zaženemo program lupdate, ki ustvari XML datoteki *slo.ts* in *nem.ts*. Ti vsebujeta besede za prevajanje:

lupdate primer.pro

Datoteki .ts odpremo v programu Linguist, ki je del orodjarne QT in besede prevedemo v želene jezike. Namesto programa Linguist lahko uporabimo poljuben urejevalnik besedil.

Datoteki .ts nato s programom lrelease prevedemo v obliko primerno za uporabo v aplikaciji (*slo.qm* in *nem.qm*):

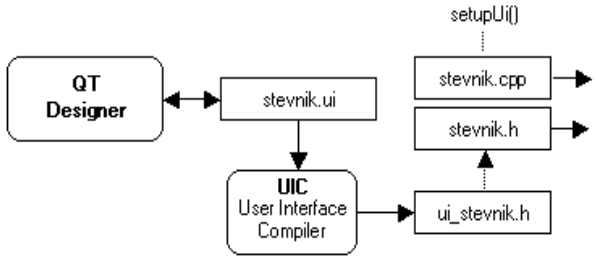
lrelease primer.pro

2.3 QT Designer

Designer je orodje namenjeno oblikovanju grafičnih vmesnikov. Omogoča izbiro in razporejanje različnih widgetov po načinu povleci in spusti ter hiter in enostaven način določanja njihovih lastnosti. Designer omogoča tudi grafični predogled aplikacije na različnih sistemih (MS Windows, Motif, CDE, ...), kar omogoča razvijalcu, da že med delom vidi, kakšen bo končni izgled aplikacije (ang. *What You See Is What You Get*).

Kot rezultat oblikovanja dobimo datoteko s končnico .ui, ki vsebuje XML opis grafičnega vmesnika. To datoteko nato predprocesor UIC (User Interface Compiler) prevede v poseben razred (*POD class*), ki se shrani v zaglavno datoteko *ui_ime.h*. To nato vključimo v izvorno kodo.

Na ta način je dosežena enostavna integracija vmesnika v izvorno kodo ter možnost kasnejšega spreminjanja oblike vmesnika.



Slika 5. Prevajanje QT Designer aplikacije

Primer programa *stevnik* iz programa 2 narejenega z uporabo Designerja prikazuje program 4.

stevnik.h:

```
#include <QtGui>
#include "ui_stevnik.h"

class Stevnik : public QWidget, private Ui::mojStevnik {
  Q_OBJECT
public:
  Stevnik(QWidget *parent = 0);
private slots:
  void napisi();
private:
  int i;
};
```

V kodo smo vključili datoteko *ui_stevnik.h*, ki jo ustvari orodje UIC (User Interface Compiler) iz XML datoteke *stevnik.ui*.

Razred *Stevnik* dedujemo od razredov *QWidget* in *mojStevnik*. Slednjega je ustvaril UIC in predstavlja okno narejeno v Designerju.

stevnik.cpp:

```
#include "stevnik.h"

Stevnik::Stevnik(QWidget *parent) {
  i=0;
  setupUi(this);
  connect(gumb1, SIGNAL(clicked()), this, SLOT(napisi()));
}

void Stevnik::napisi() {
  i++;
  label->setText(QString::number(i));
}
```

Program 4. Program narejen z uporabo Designerja

Z metodo *setupUi* se ustvari GUI vmesnik. Nato smo še povezali signal *clicked* z režo *napisi*. Reža bo izpisala število na widget *label*, ki smo ga definirali v Designerju.

3 Sklep

Knjižnica QT omogoča hiter in enostaven razvoj GUI aplikacij. Čas učenja je kratek, zaradi uporabe makrov in predprocesorjev pa je izvorna koda tudi zelo pregledna. Knjižnica vsebuje obsežno podporo za dostop do baz podatkov, podporo mrežnim aplikacijam, podporo OpenGL grafiki itd. Dobre lastnosti se kažejo tudi v prenosljivosti izvorne kode med različnimi operacijskimi sistemi, dodelani podpori za internacionalizacijo ter orodju Designer.

Prav zaradi tega predstavlja QT osnovo mnogim znanim aplikacijam, kot so namizno okolje KDE, spletni brskalnik Opera, programi Google Earth, Skype, Adobe Photoshop Album, VLC Media Player itd.

Slabe lastnosti QT se kažejo predvsem v počasnosti (glede na podobna orodja WxWidgets, GTK+). Nekateri uporabniki kritizirajo QT tudi zaradi uporabe makrov in predprocesorjev in s tem odklona od "čistega" C++ programiranja.

QT obstaja v dveh različicah LGPL in komercialni. Slednja ima vgrajeno močnejšo podporo za komercialne podatkovne baze ter ni omejena z licencami za distribucijo aplikacij razvitih s QT.

QT deluje poleg okolja C++ tudi v okoljih C# (Qyoto/Kimono), java (Qt Jambi), perl (Perl Qt4), PHP (PHP-Qt), python (PyQt), pascal (FreePascal QT4) ...

Literatura

[1] J. Blanchette, M. Summerfield: C++ GUI programming with Qt4, Trolltech AS, 2006

[2] Reference Documentation, http://doc.trolltech.com/4.4

[3] Linguist, http://doc.trolltech.com/4.4/linguist-hellotr.html

[4] Designer, http://doc.trolltech.com/4.4/designer-manual.html