Introdução ao Desenvolvimento com Qt4 e KDE4

Sandro Santos Andrade [1, 2]
Tomaz Martins dos Santos Canabrava [2]
{sandro.andrade, tomaz.canabrava}@gmail.com

[1] Universidade Federal da Bahia
Departamento de Ciência da Computação (DCC)
Laboratório de Sistemas Distribuídos (LaSiD)

[2] Faculdade Ruy Barbosa
Bacharelado em Ciência da Computação

Objetivos

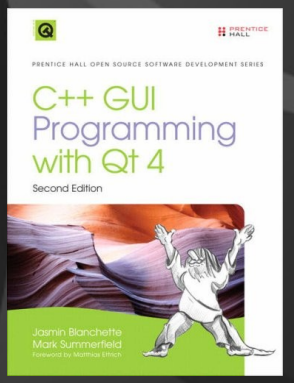
- Apresentar as principais funcionalidades do Qt 4.4.3 e do KDE 4.1.2 para o desenvolvimento produtivo de aplicações *desktop* multi-plataforma modernas;
- Proporcionar uma vivência prática inicial das soluções mais utilizadas nessas plataformas, motivando a formação de novos desenvolvedores Qt/KDE;
- Discutir decisões de projeto, idiomas e ferramentas auxiliares nessas plataformas.

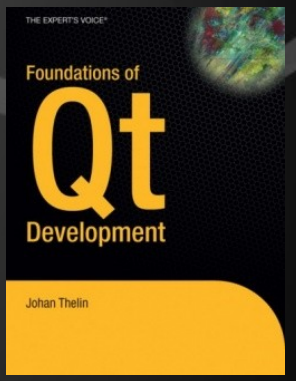
Pré-requisitos

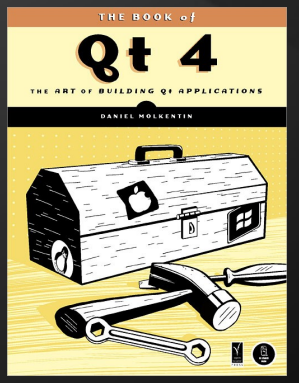
- Necessários:
  - Fundamentos de Orientação a Objetos (OO);
  - Experiência com alguma linguagem OO;
  - Experiência com desenvolvimento de aplicações *desktop*.
- Desejáveis:
  - Fundamentos da linguagem Standard C++.
- Especiais:
  - Padrões de projeto, estilos arquiteturais, *application frameworks* etc.

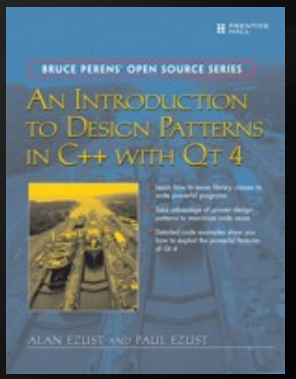
Metodologia

- Duração: 2 *slots* de 3 horas;
- Tópicos expositivos e laboratórios práticos;
- Referências:
  - Livros;
  - *Qt Reference Documentation*;
  - Fóruns (*QtCentre* etc);
  - *Qt Quarterly*.





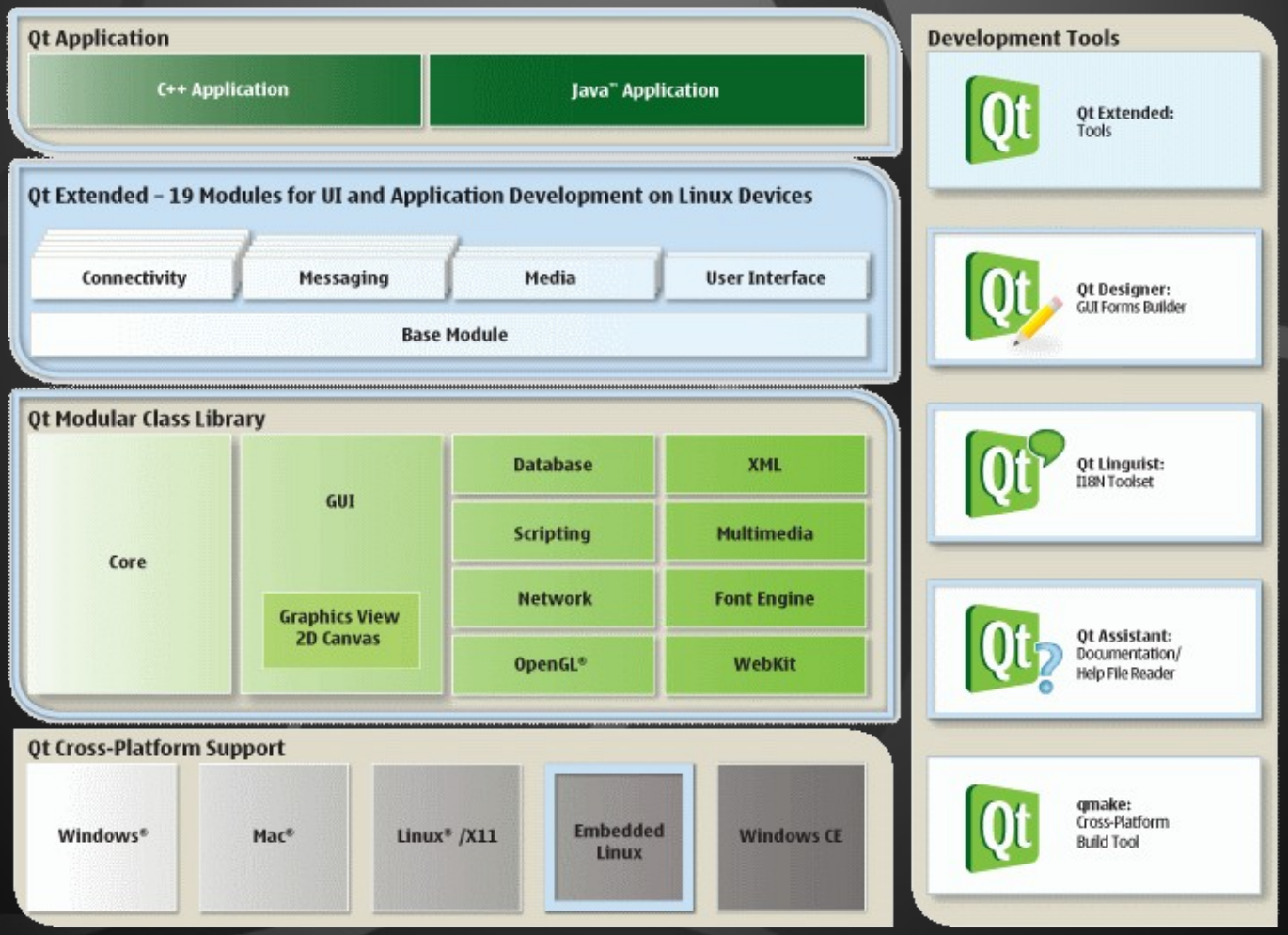




Visão Geral

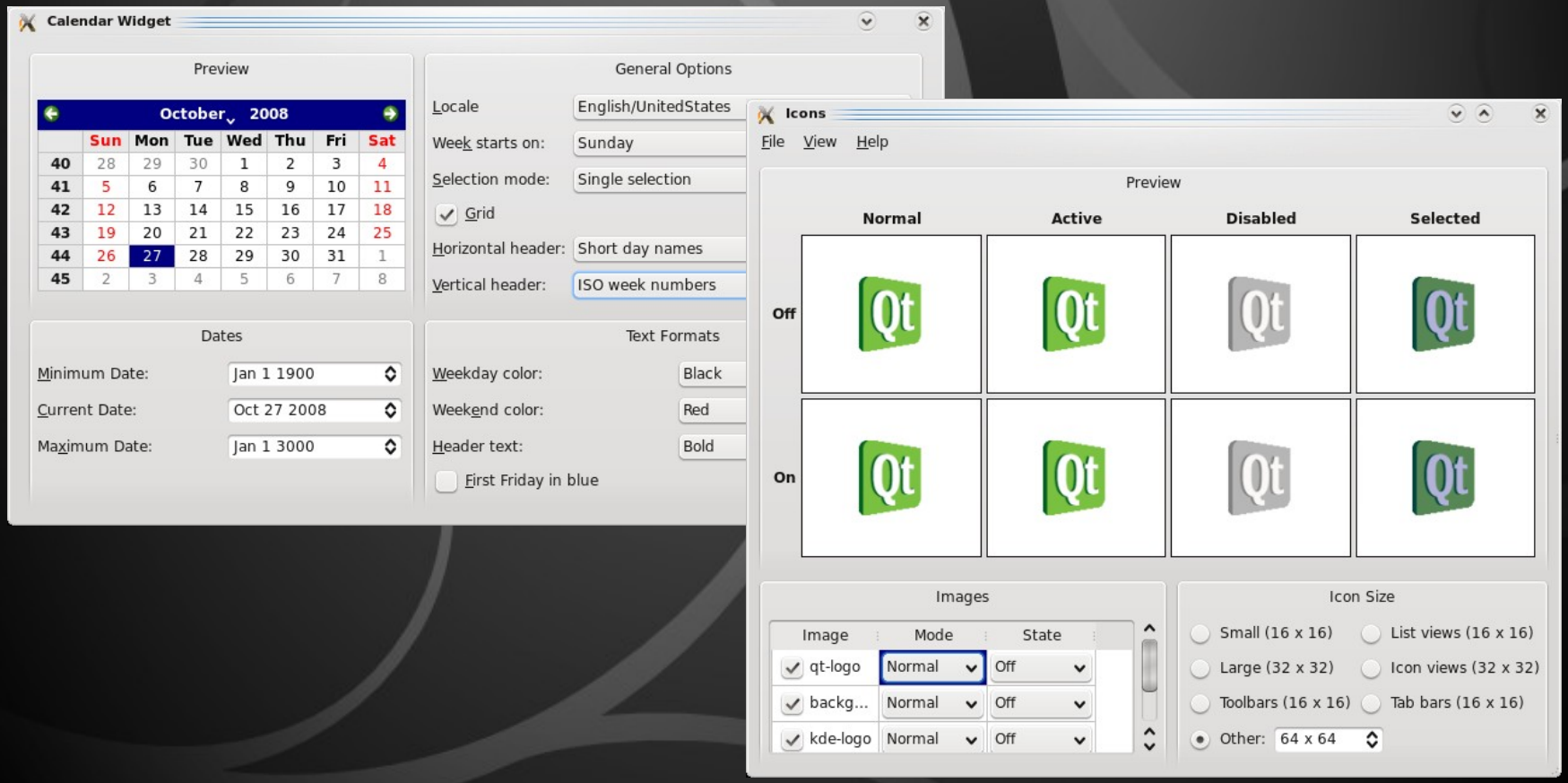
- O Qt é um *toolkit* para o desenvolvimento de aplicações GUI multi-plataforma, com recursos para IPC, *networking*, XML, SVG, banco de dados, *scripting*, OpenGL, multimídia e soluções embarcadas.
- Disponível publicamente desde maio de 1995.
- Possui cerca de 500 classes e 9000 funções.
- Ganhou os prêmios *LinuxWorld* (1999, 2001 e 2002) e *Jolt Productivity Award* (2002).
- Possui licença dual.

Visão Geral


Qt Application
C++ Application
Java™ Application
Qt Extended – 19 Modules for UI and Application Development on Linux Devices
Connectivity
Messaging
Media
User Interface
Base Module
Qt Modular Class Library
Core
GUI
Graphics View
2D Canvas
Database
XML
Scripting
Multimedia
Network
Font Engine
OpenGL®
WebKit
Qt Cross-Platform Support
Windows®
Mac®
Linux® /X11
Embedded Linux
Windows CE
Development Tools
Qt Extended: Tools
Qt Designer: GUI Forms Builder
Qt Linguist: I18N Toolset
Qt Assistant: Documentation/ Help File Reader
qmake: Cross-Platform Build Tool

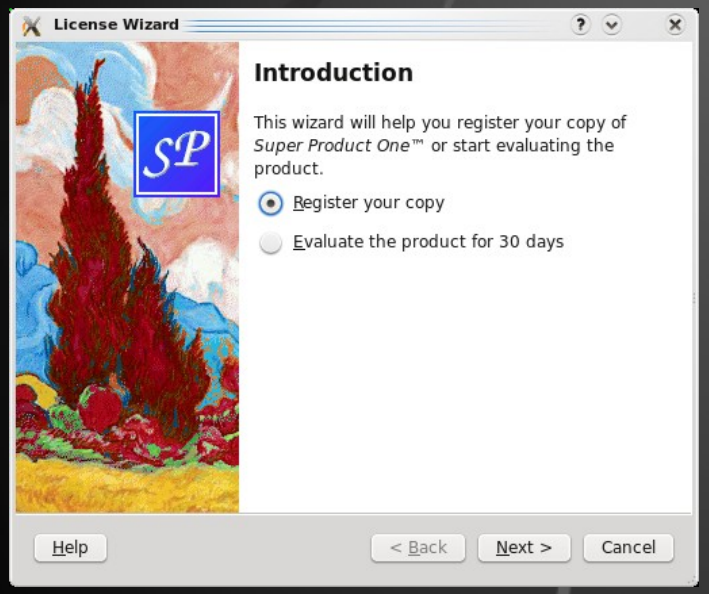
Visão Geral

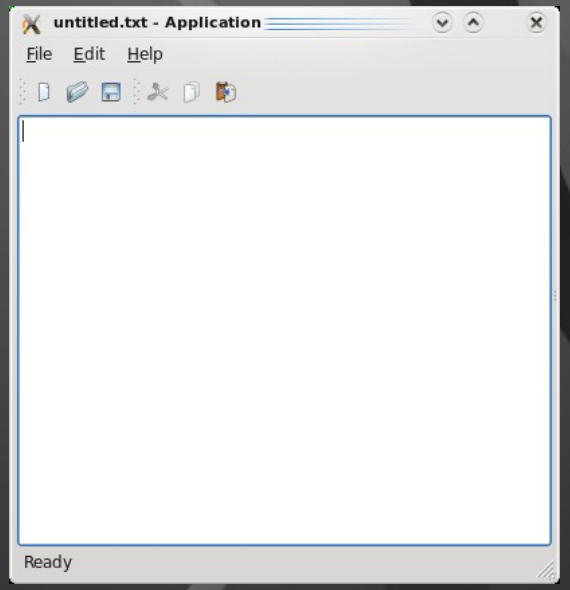
- *Widgets*:

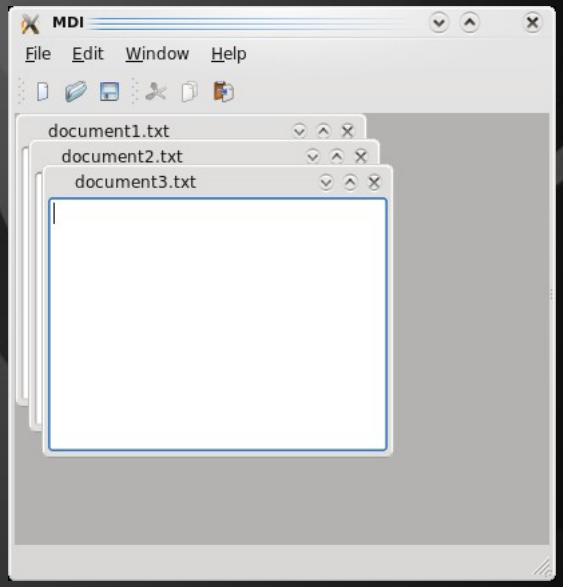


Visão Geral

- *Dialogs* e *MainWindows*:

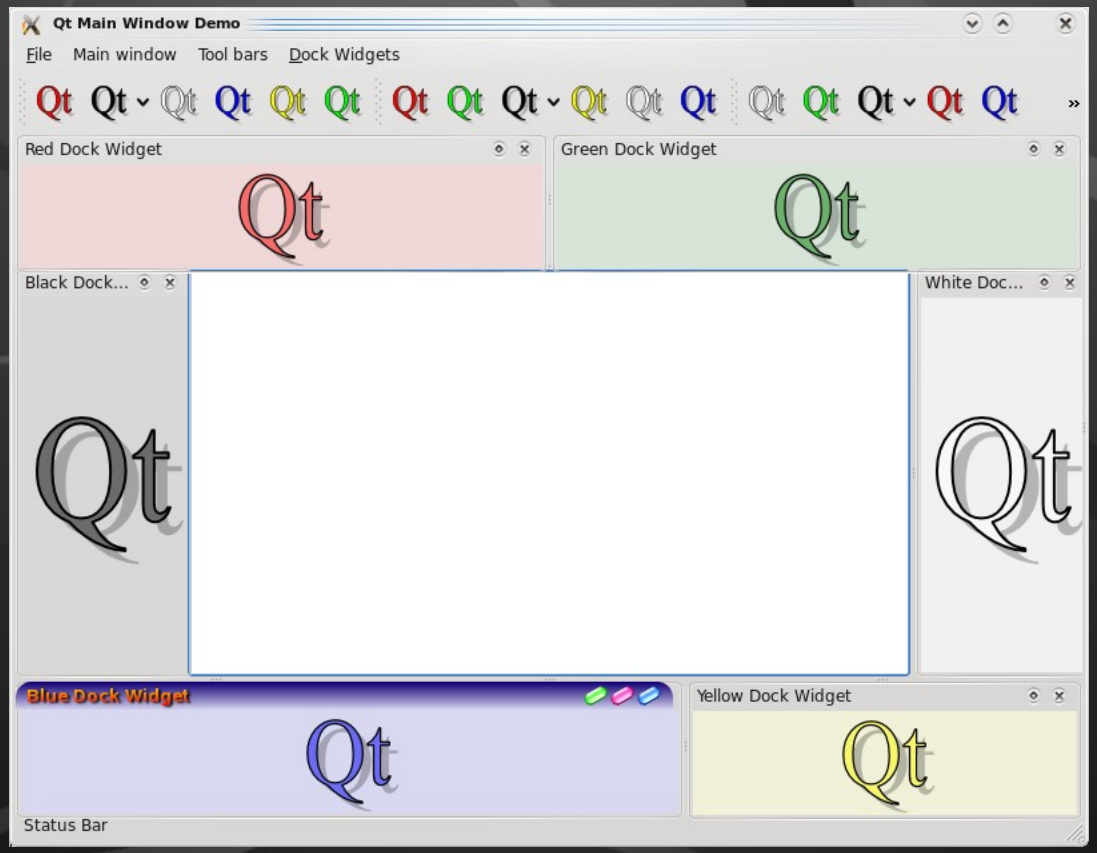






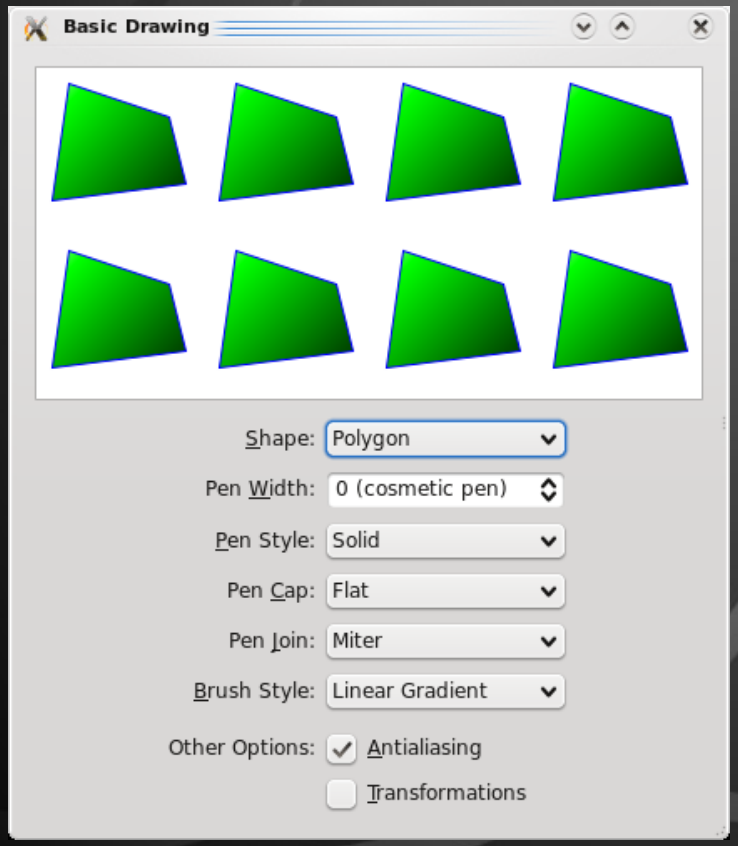
Visão Geral

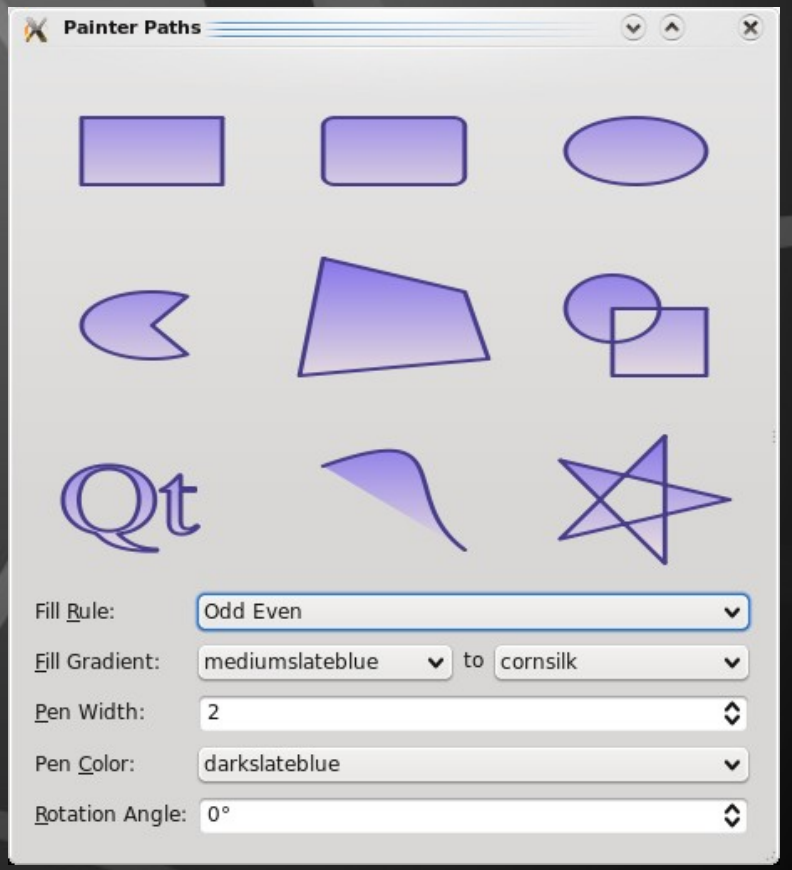
- *Dialogs* e *MainWindows*:



Visão Geral

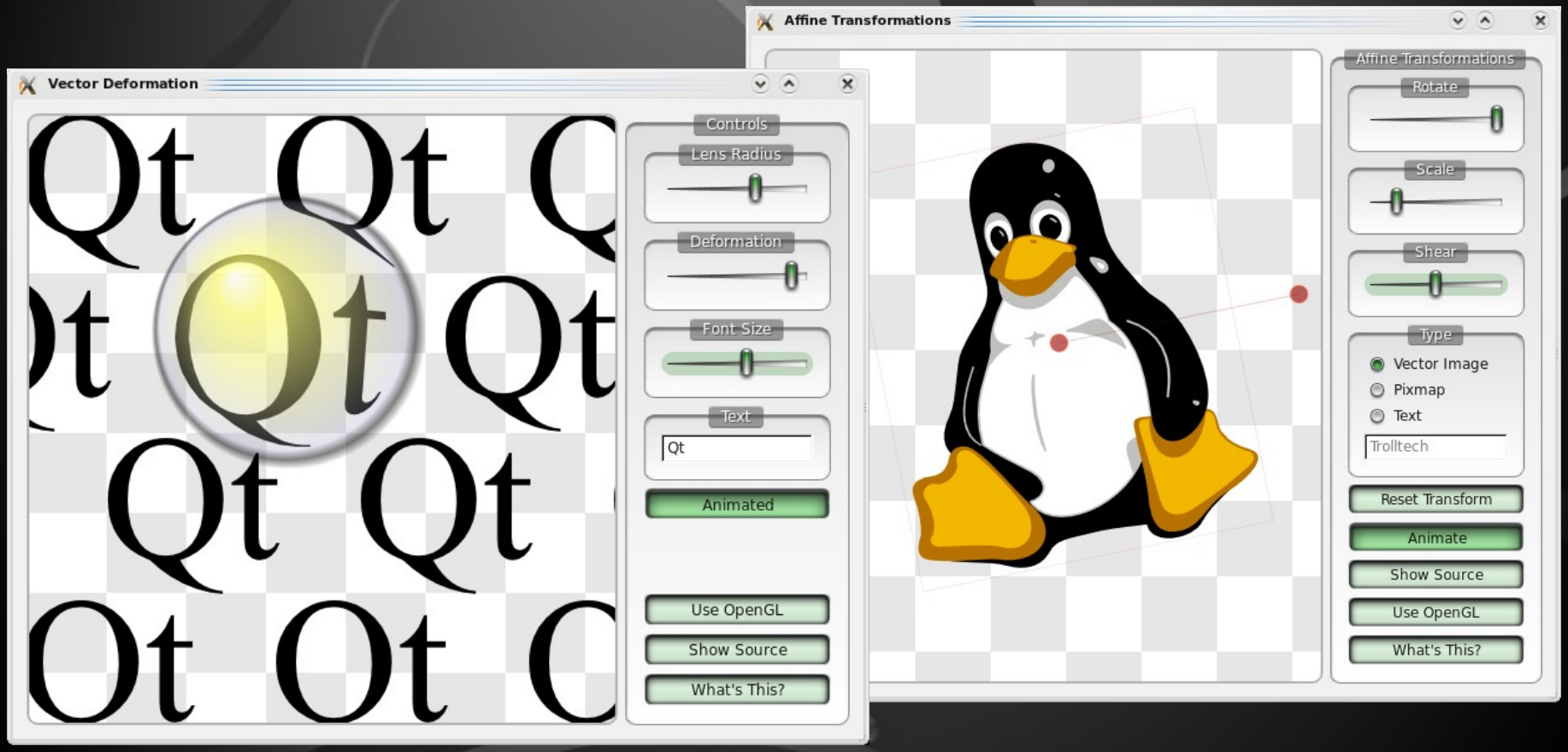
- Gráficos 2D:





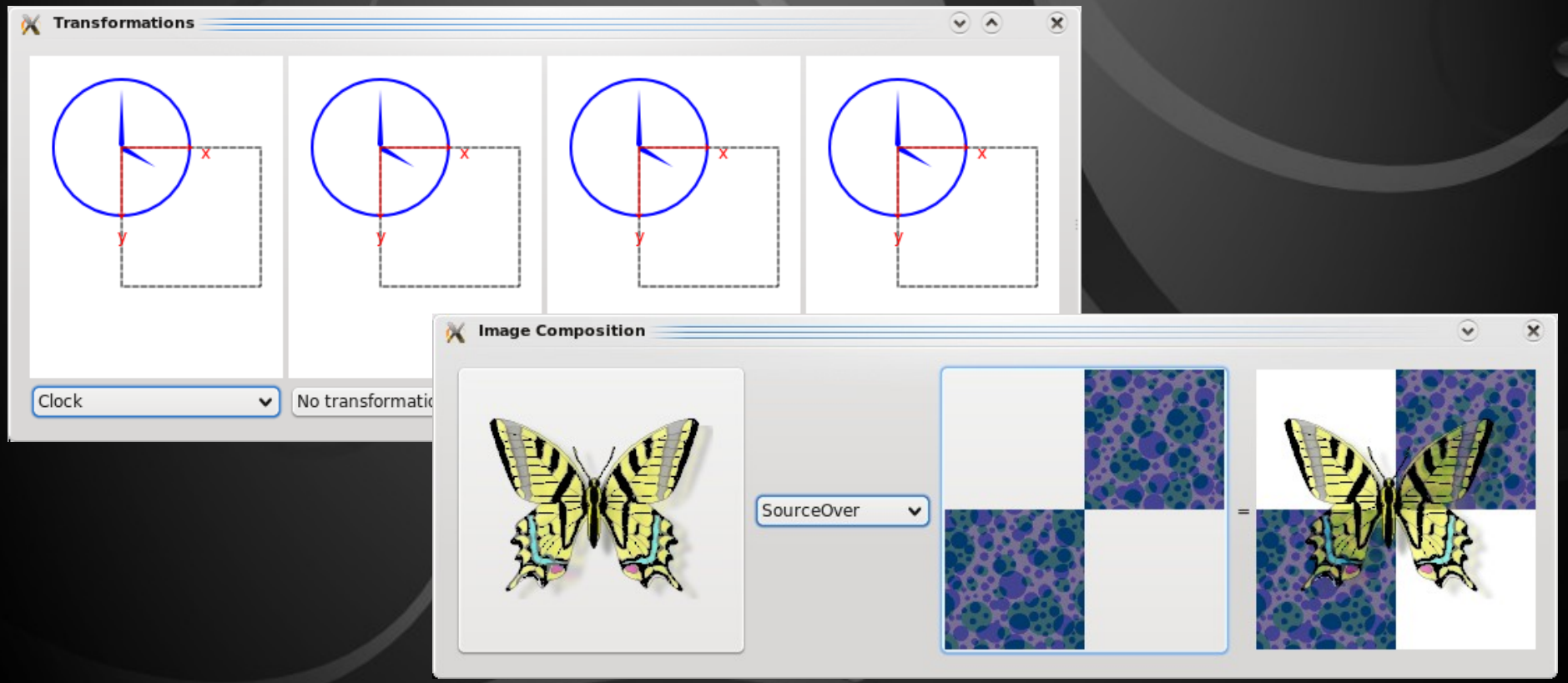
Visão Geral

- Gráficos 2D:



Visão Geral

- Gráficos 2D:

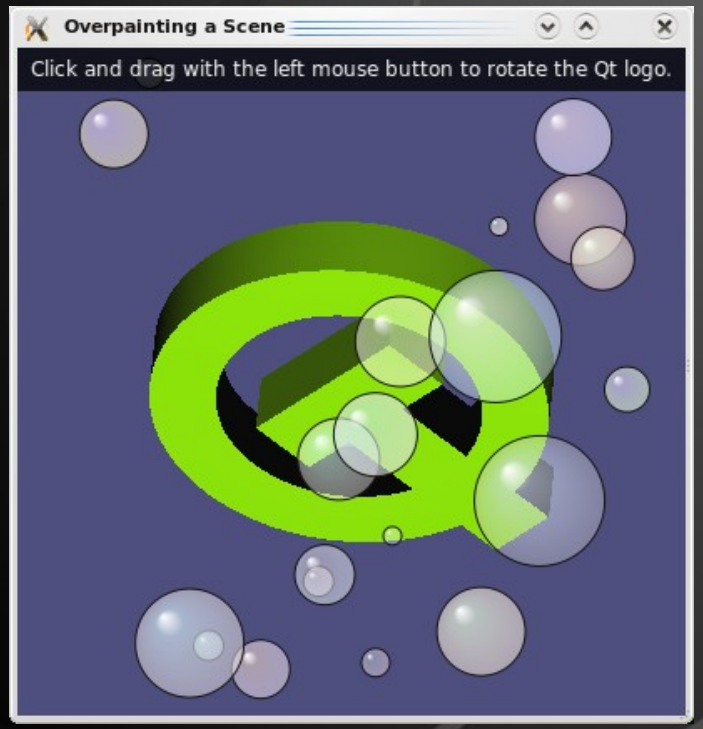


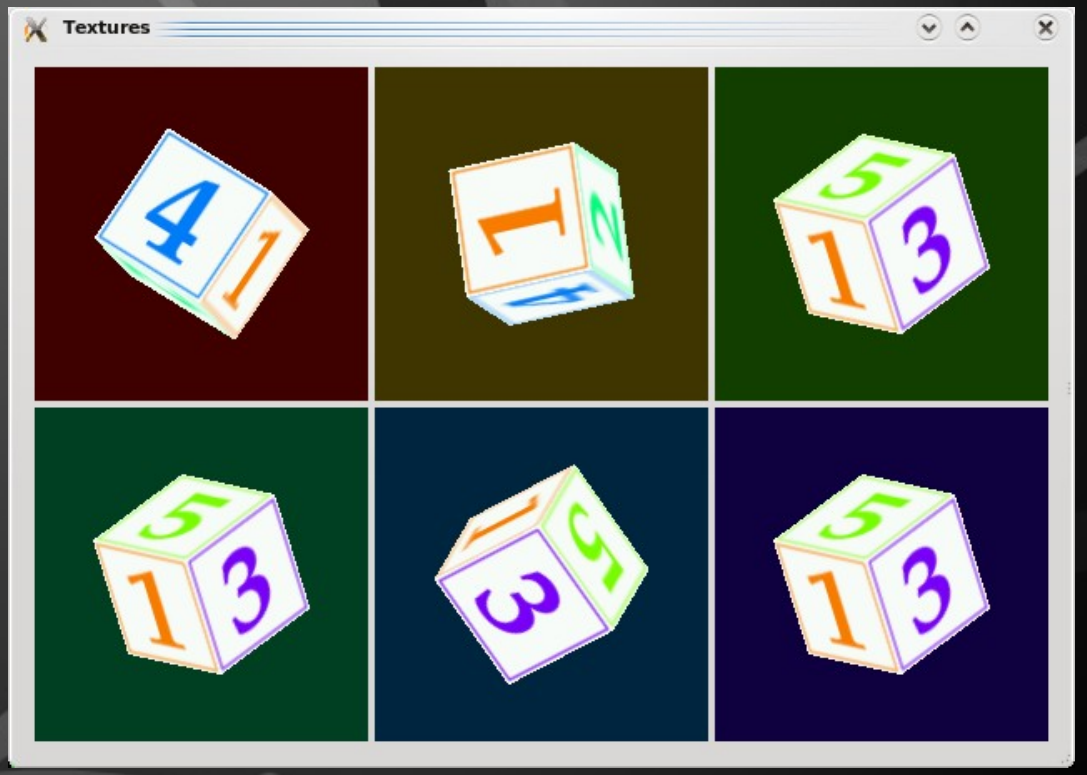
Visão Geral

- Gráficos 2D:

Visão Geral

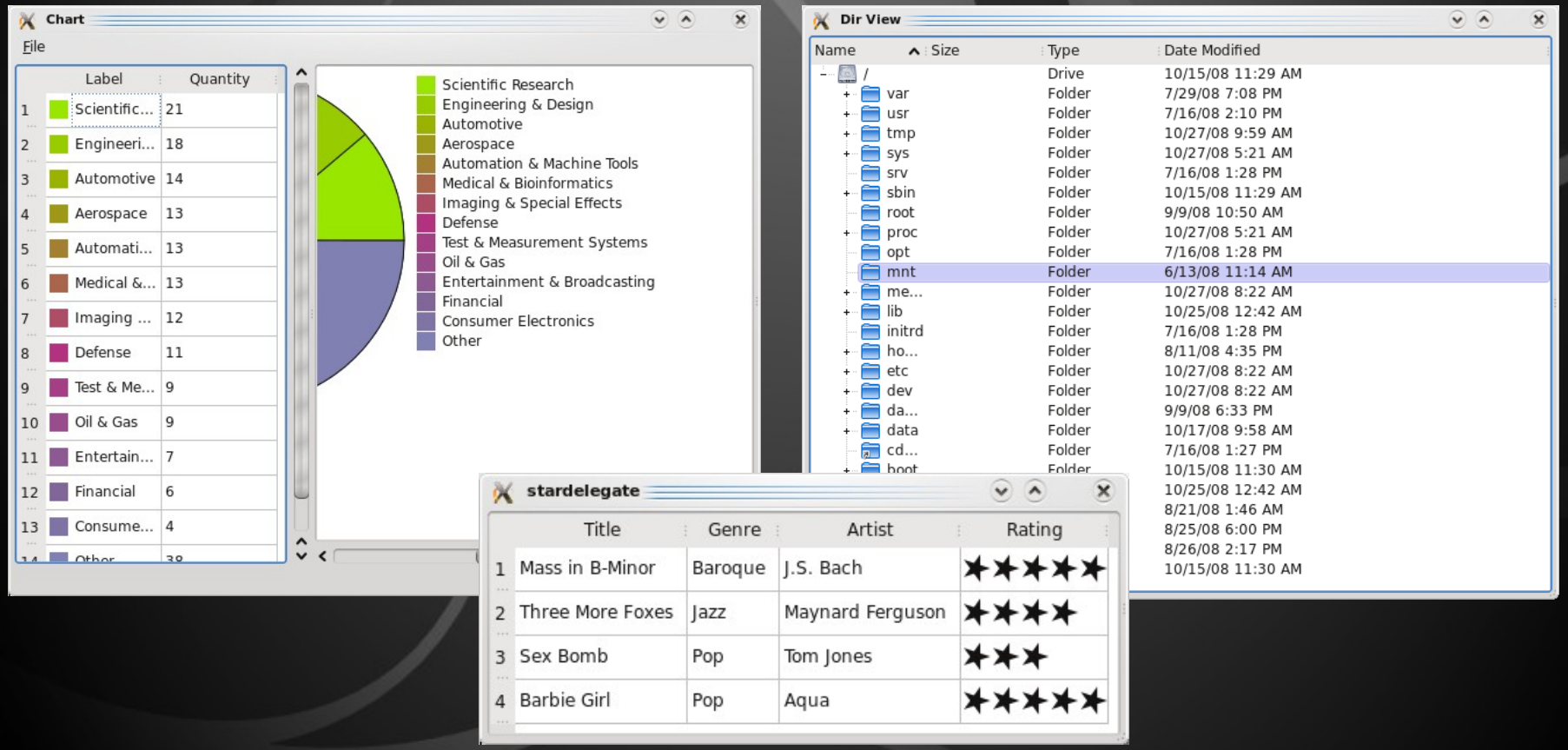
- OpenGL:





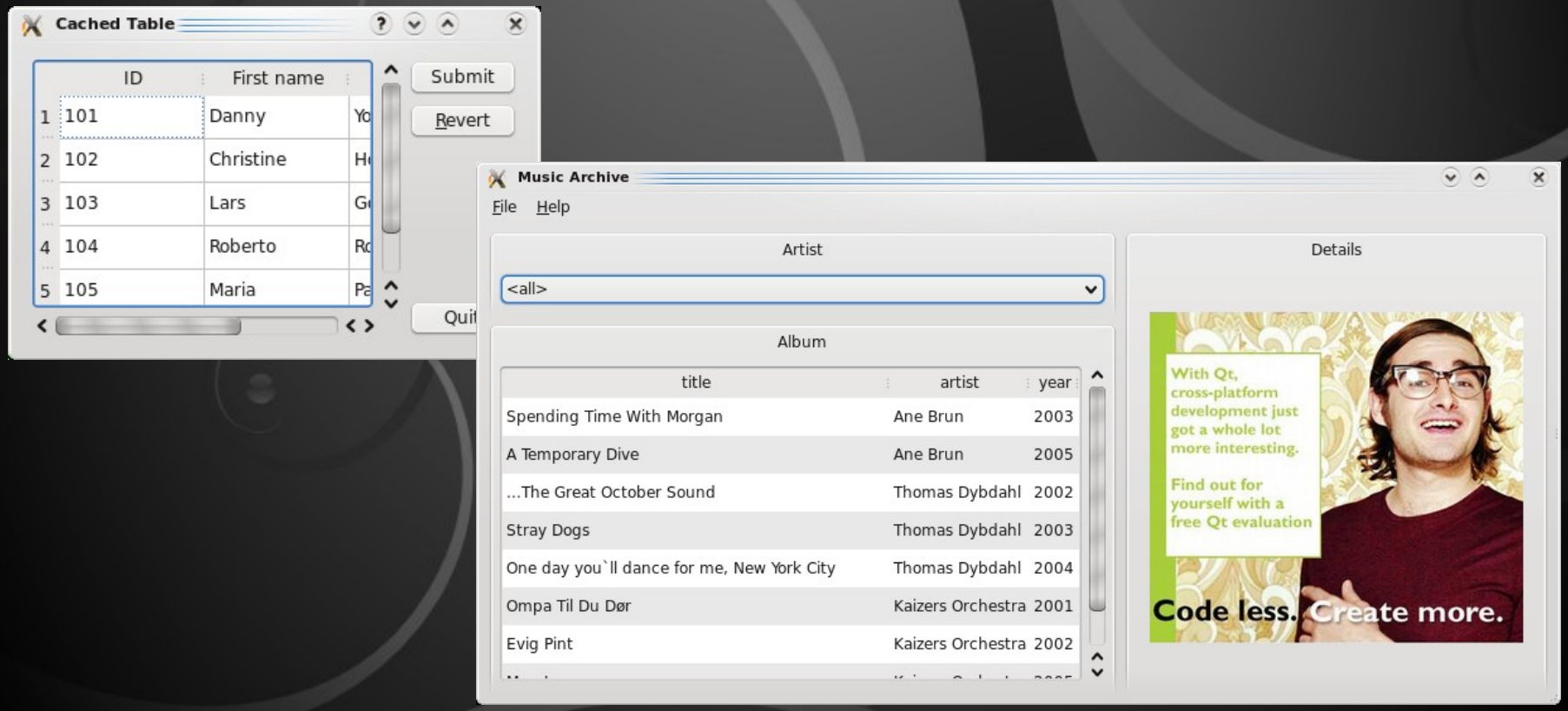
Visão Geral

- *Model-View*:



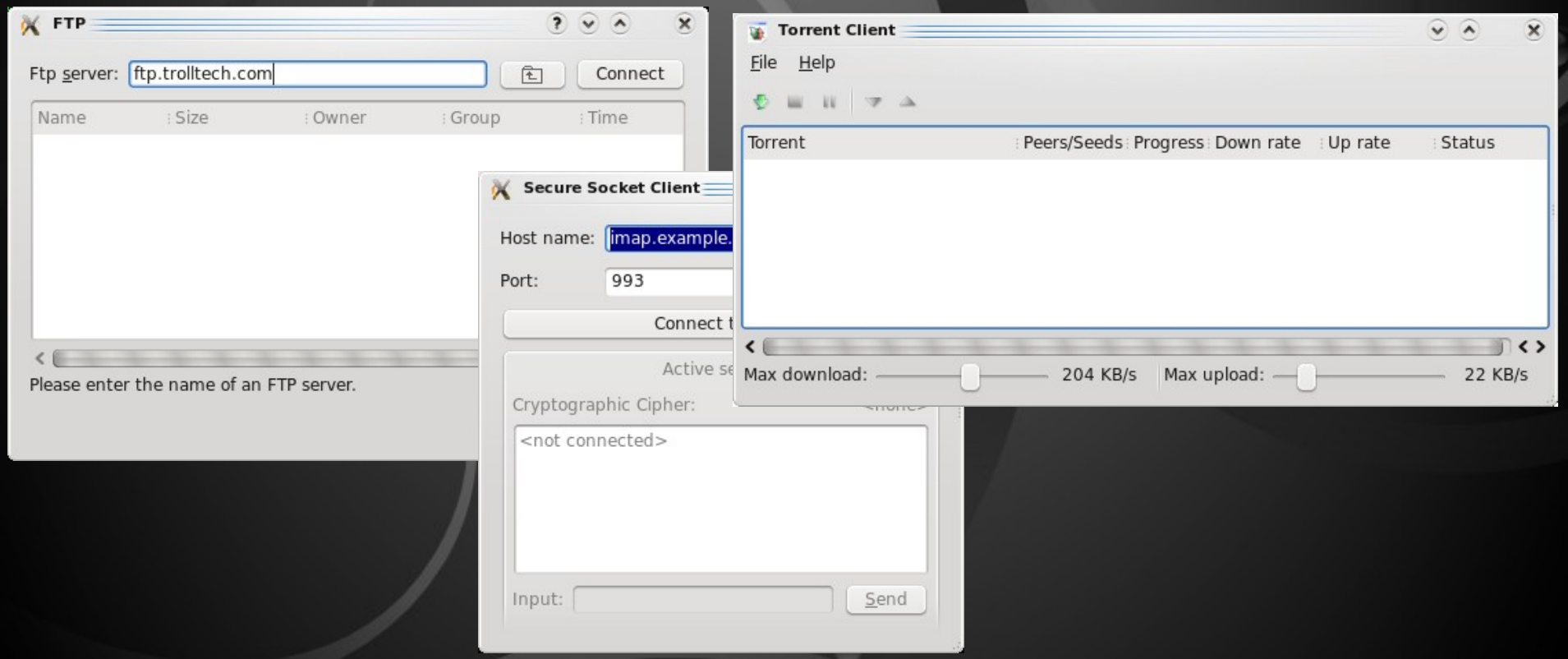
Visão Geral

- Banco de dados:



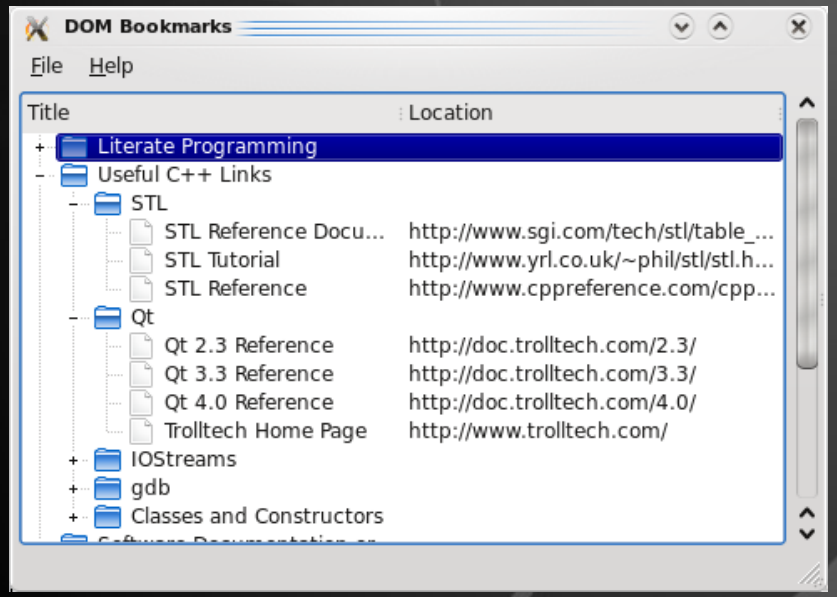
Visão Geral

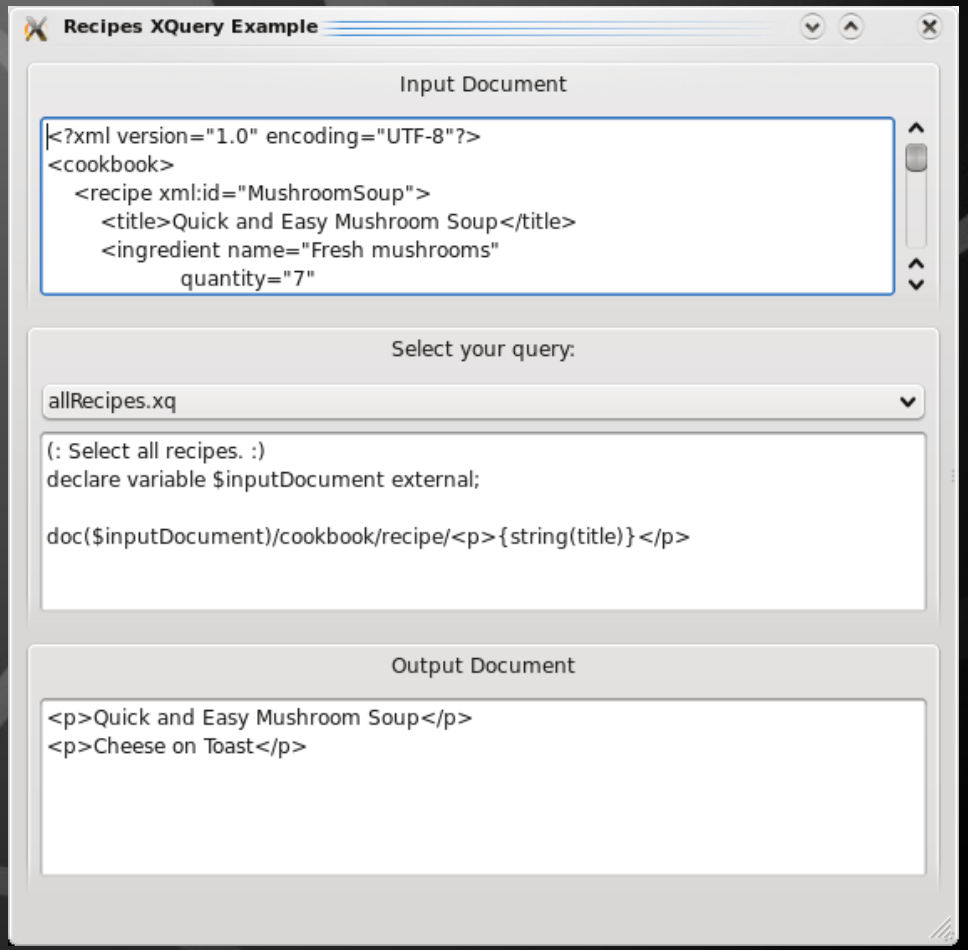
- *Networking*:



Visão Geral

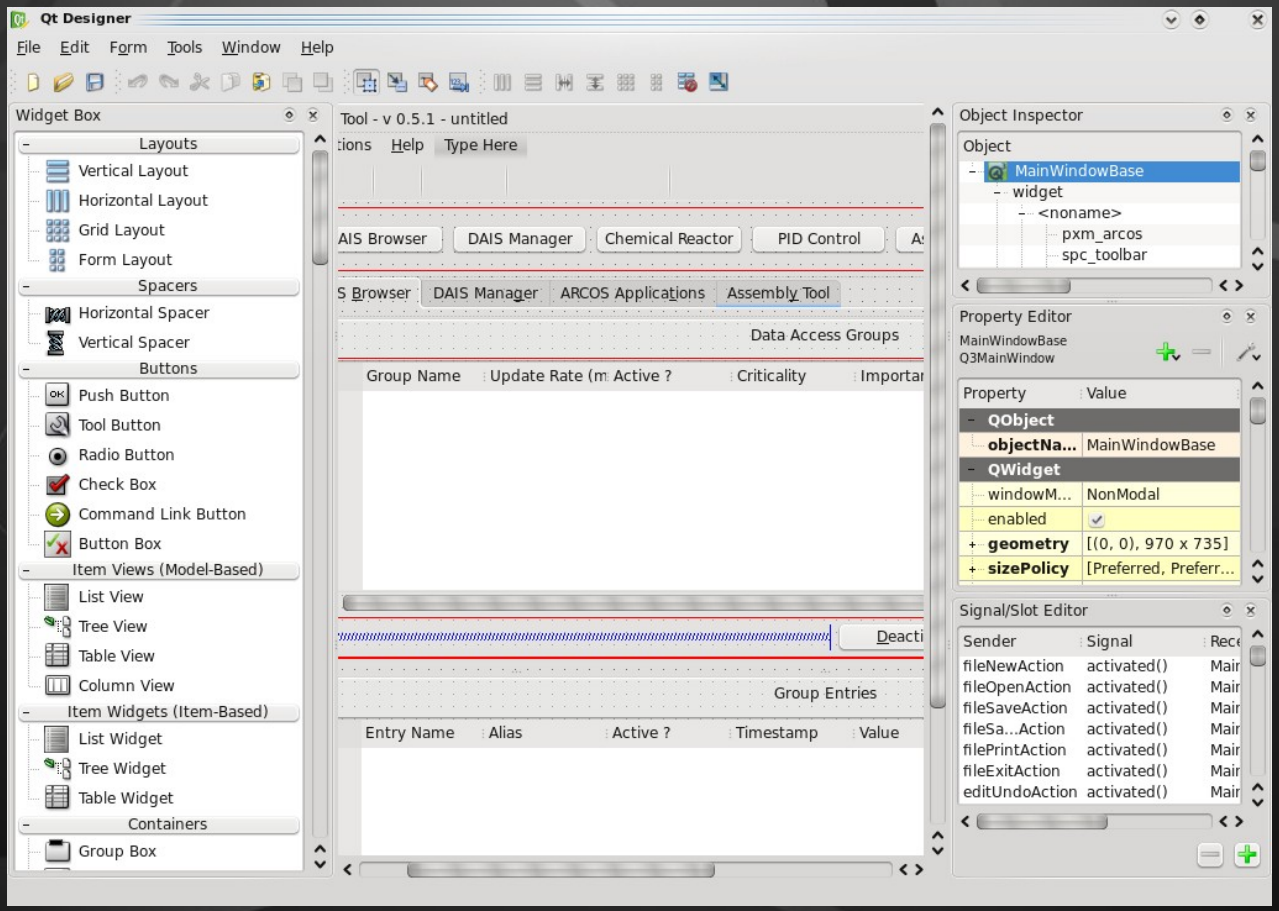
- XML:





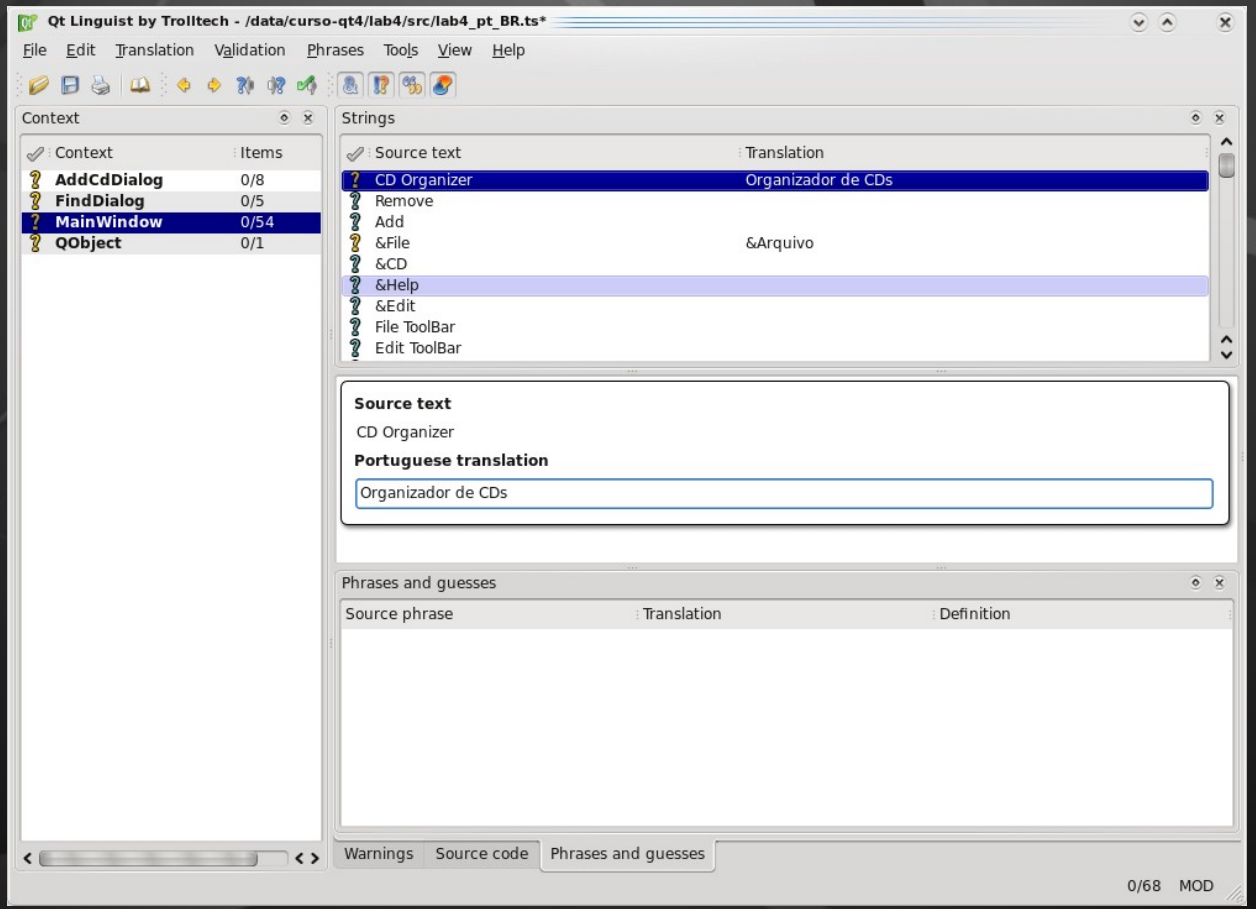
Visão Geral

- Ferramentas (*Qt Designer*):



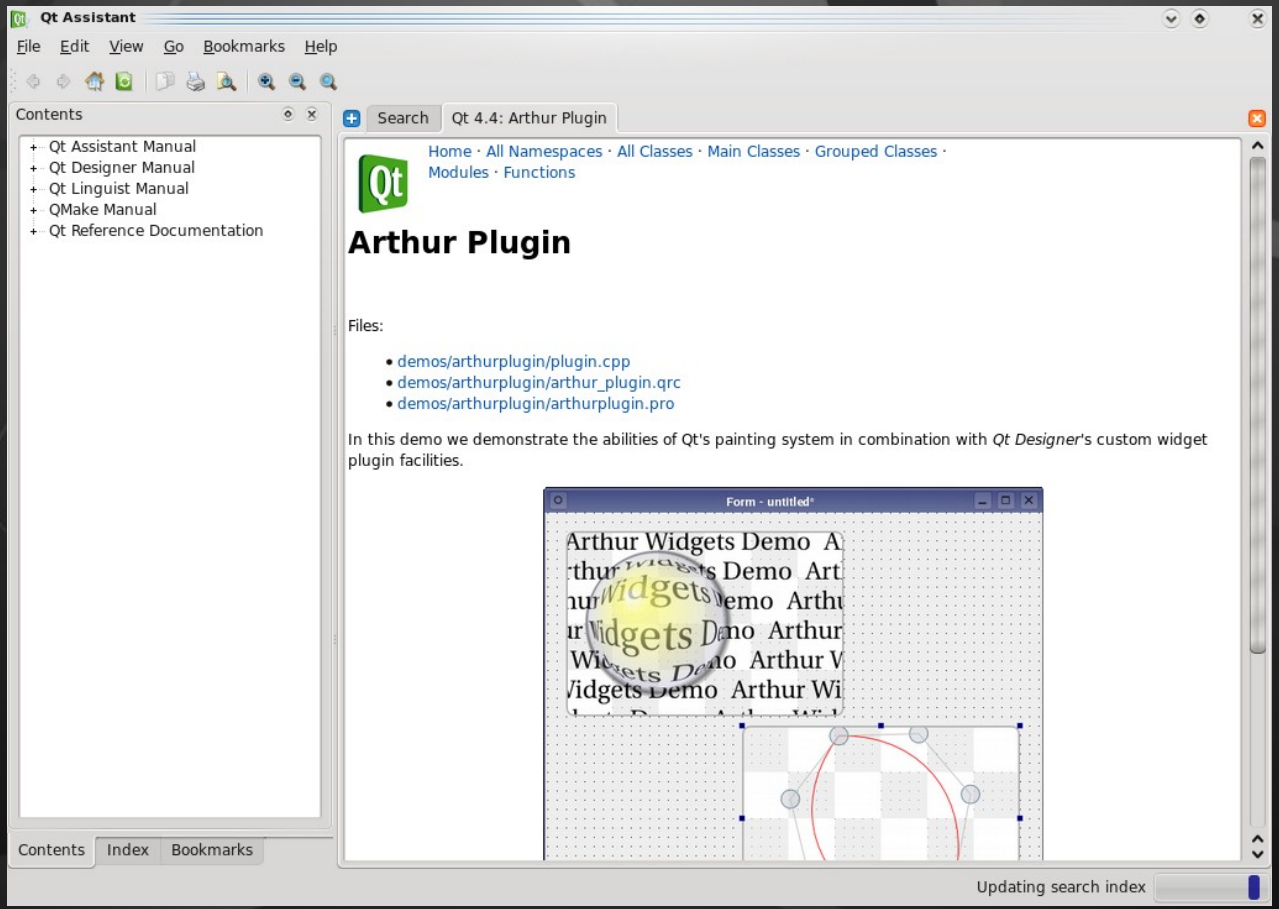
Visão Geral

- Ferramentas (*Qt Linguist*):



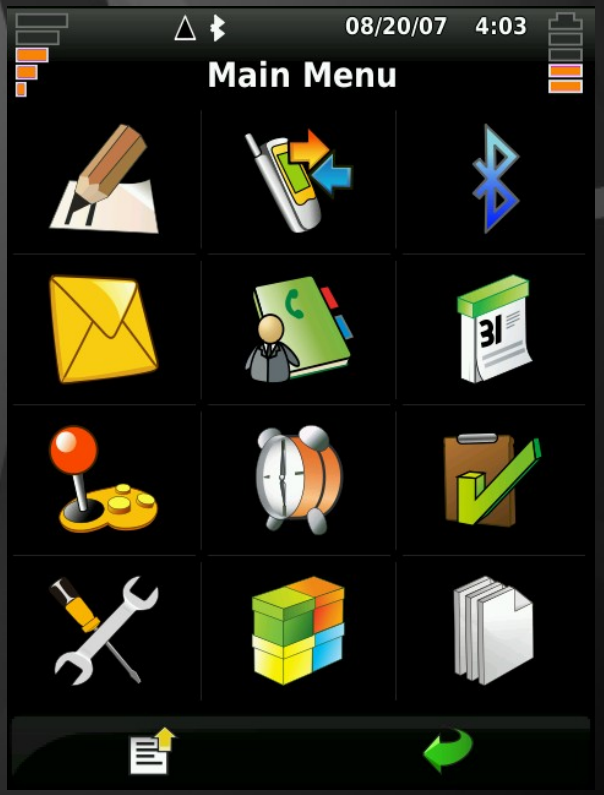
Visão Geral

- Ferramentas (*Qt Assistant*):



Visão Geral

- Qtopia (*Embedded Qt*):



Visão Geral

- E mais:
  - Extensões do Qt: *signals*/*slots*, meta-objetos, *containers*;
  - *Style Sheets* e *Scripting* (*QtScript*);
  - *Web Browser Engine* (*WebKit*) e *Plugins*;
  - Multimídia (*Phonon*) e Manipulação de SVG;
  - *Bindings* para outras linguagens (*Qt Jambi*, *PyQt*, *QtRuby*, *Perl*, *Qt#*, *Qymono*);
  - Qt na web ? Wt !!!;
  - Extensões do KDE e de terceiros (ex: *qwt*, *qanava*, *quazip*, *QLALR* ...).

História do Qt

- Primeira versão disponibilizada em 1995, por *Haavard Nord e Eirik Chambe-Eng*;
- Seu desenvolvimento se iniciou em 1991 e em 1993 já existia um núcleo que suportava *widgets*;
- A letra 'Q' foi escolhida porque ela aparecia de forma bonita no editor *emacs* de *Haavard* :);
- O “t” vem da palavra *toolkit*;
- Em 1994 foi fundada a *Trolltech*, antes *Troll Tech* e ainda antes *Quasar Technologies*;

História do Qt

- Em 1996 foi lançado o Qt 1.1 e a *Trolltech* tinha 8 clientes;
- Também em 1996 o projeto KDE (*The K Desktop Environment*) foi fundado por *Matthias Ettrich*;
- O 'K' (do KDE) era simplesmente a letra que vinha antes do 'L' (do Linux) :);
- Em 1997 o Qt passa a ser utilizado no desenvolvimento do KDE e a versão 1.3 é lançada;
- Em 1999, o Qt 2 passa a ser licenciado pela QPL;

História do Qt

- Em 2000 é lançado o Qtopia (Qt para ambientes embarcados);
- Neste mesmo, o Qt passa a ser licenciado pela GPL;
- Em 2001 é lançado o Qt3;
- Em 2005 é lançado o Qt4: primeira versão *open-source* em todas as plataformas;
- Em janeiro de 2008 a Trolltech é comprada pela Nokia.

Porque o Qt ?

- Multi-plataforma com código-fonte único;
- Riqueza de características e funcionalidades;
- Bom desempenho;
- Disponibilidade de código-fonte;
- Otima documentação;
- Disponibiliza soluções corretas sob o ponto de vista da engenharia de *software*;
- *Write once, compile anywhere*.

Instalando o Qt 4.4.3

- Linux:
  - Pacotes .deb: (essenciais)

libqt4-core
libqt4-dev
libqt4-gui
libqt4-opengl-dev
libqt4-assistant
libqt4-dbus
libqt4-designer
libqt4-help
libqt4-network

libqt4-opengl
libqt4-script
libqt4-sql
libqt4-svg
libqt4-test
libqt4-webkit
libqt4-xml
libqt4-xmlpatterns

Instalando o Qt 4.4.3

- Linux:
  - Pacotes .deb: (recomendados)

libqt4-qt3support
libqt4-sql-mysql
libqt4-sql-odbc
libqt4-sql-psql
libqt4-sql-sqlite
libqt4-sql-sqlite2
qt4-demos
qt4-designer

python-qt4
python-qt4-common
python-qt4-dbg
python-qt4-dbus
qt4-dev-tools
qt4-doc
qt4-doc-html
qt4-qtconfig

Instalando o Qt 4.4.3

- Windows:
  - Fazer o *download* do pacote binário para Windows;
  - Executar o programa de instalação;
  - O programa de instalação irá fazer o *download* do MinGW (*Minimalist GNU for Windows*);
  - Pode ser utilizado com o Microsoft Visual C++.

Hello Qt

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *label = new QLabel("Hello Qt!");
    label->show();
    return app.exec();
}
```

lab1 / main.cpp

- Executar:

  $ *qmake -project*

  $ *qmake*

  $ *make*

Hello Qt

Toda classe do Qt4 está declarada em um arquivo *header* com o mesmo nome

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *label = new QLabel("Hello Qt!");
    label->show();
    return app.exec();
}
```

lab1 / main.cpp

- Executar:

  $ *qmake -project*

  $ *qmake*

  $ *make*

Hello Qt

Toda classe do Qt4 está declarada em um arquivo *header* com o mesmo nome

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *label = new QLabel("Hello Qt!");
    label->show();
    return app.exec();
}
```

Cria um objeto *application* para gerenciar recursos da aplicação e um *label* contendo "Hello Qt!"

lab1 / main.cpp

- Executar:

  $ *qmake -project*

  $ *qmake*

  $ *make*

Hello Qt

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *label = new QLabel("Hello Qt!");
    label->show();
    return app.exec();
}
```

lab1 / main.cpp

Toda classe do Qt4 está declarada em um arquivo *header* com o mesmo nome

Cria um objeto *application* para gerenciar recursos da aplicação e um *label* contendo "Hello Qt!"

Torna o *label* visível.

- Executar:

  $ *qmake -project*

  $ *qmake*

  $ *make*

Hello Qt

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *label = new QLabel("Hello Qt!");
    label->show();
    return app.exec();
}
```

lab1 / main.cpp

Toda classe do Qt4 está declarada em um arquivo *header* com o mesmo nome

Cria um objeto *application* para gerenciar recursos da aplicação e um *label* contendo "Hello Qt!"

Torna o *label* visível.

Passa o controle da aplicação para o Qt (entra no *loop* de eventos)

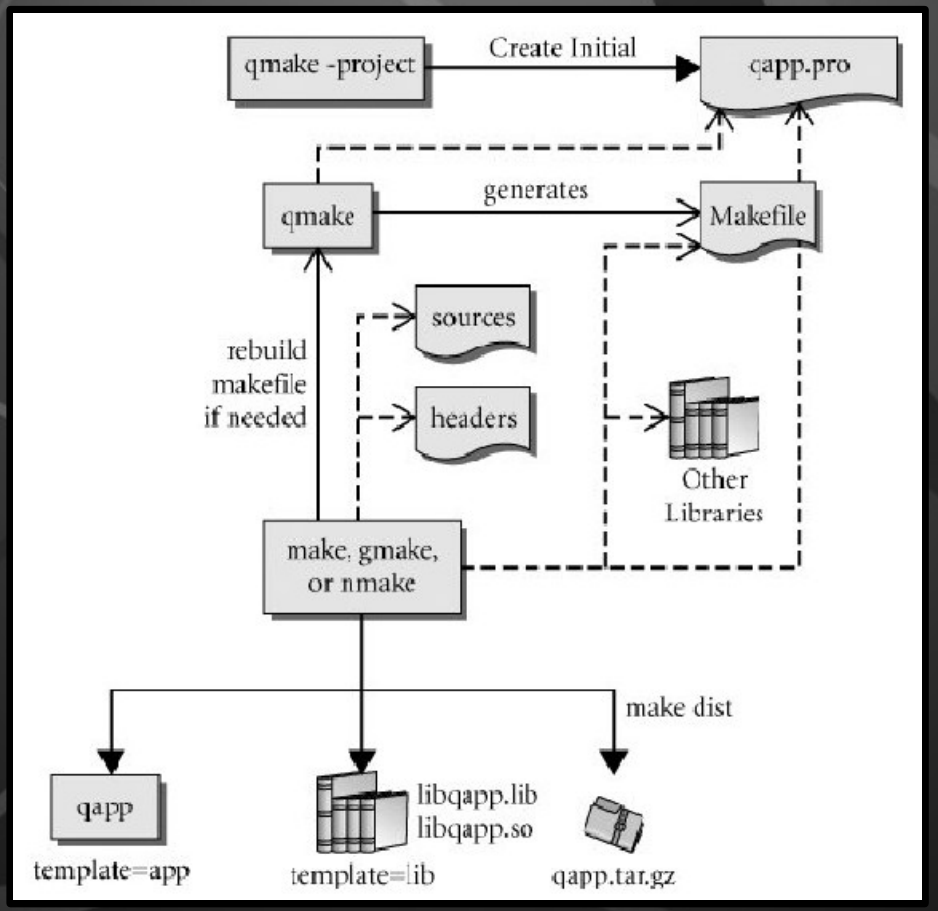
- Executar:
  - $ *qmake -project*
  - $ *qmake*
  - $ *make*

Hello Qt

- O QMake:
  - Ferramenta que automatiza o processo de compilação, *linking* e instalação em diferentes plataformas;
  - Realiza a geração automática de *Makefiles* a partir de arquivos de projeto de fácil criação;
  - O arquivo de projeto pode ser criado pelo desenvolvedor ou automaticamente pelo *qmake* (opção *-project*);
  - Os módulos *QtCore* e *QtGui* são automaticamente incluídos no processo de *linking*.

Hello Qt

- O QMake:



Hello Qt

- O QMake:
  - Arquivo de projeto automaticamente gerado neste exemplo:

```
#############################################################
# Automatically generated by qmake (2.01a) Wed Oct 29 18:01:15 2008
#############################################################

TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
SOURCES += main.cpp
```

Hello Qt

- Algumas considerações:
  - O *QLabel* é um dos muitos *widgets* (*windows gadgets*) do Qt;
  - *Widgets* podem conter outros *widgets*;
  - A janela principal de um programa Qt geralmente é um *QMainWindow* ou *QDialog* contendo outros *widgets*;
  - Entretanto, qualquer *widget* pode representar a janela principal (neste exemplo, um *QLabel*);
  - Por *default*, todos os *widgets* são criados ocultos.

Modelo de Objetos do Qt

- O Qt estende o modelo de objetos do C++ com as seguintes funcionalidades:
  - *Signals/Slots*: mecanismo desacoplado para comunicação (um-muitos) entre objetos;
  - *Object properties*: atributos dinâmicos;
  - *Meta-Objects*: para operações RTTI (*run-time type information*);
  - Eventos e filtros de eventos;
  - Tradução contextual de *strings* para internacionalização.

Signals / Slots

- Representam um mecanismo central do Qt;
- *Signals* e *slots* são processados a partir do momento que a aplicação inicia o *loop* de eventos (*app->exec()*);
- Um *signal* é uma mensagem que está presente em uma classe como uma declaração de uma função-membro *void*. *Signals* não são invocados, mas emitidos (via *emit*) por um objeto da classe;
- Um *slot* é uma função-membro *void* e pode ser normalmente invocada.

Signals / Slots

- Um *signal* de um objeto pode ser conectado a *slots* de um ou mais outros objetos, desde que os parâmetros sejam compatíveis;
- Sintaxe de conexão:

```
bool QObject::connect(senderqobjptr,
                      SIGNAL(signalname(argtypelist)),
                      receiverqobjptr,
                      SLOT(slotname(argtypelist))
                      optionalConnectionType);
```

Signals / Slots

- Um *signal* de um objeto pode ser conectado a *slots* de um ou mais outros objetos, desde que os parâmetros sejam compatíveis;
- Sintaxe de conexão:

```
bool QObject::connect(senderqobjptr,
                      SIGNAL(signalname(argtypelist)),
                      receiverqobjptr,
                      SLOT(slotname(argtypelist))
                      optionalConnectionType);
```

Classe base de todos os objetos Qt

Signals / Slots

- Um *signal* de um objeto pode ser conectado a *slots* de um ou mais outros objetos, desde que os parâmetros sejam compatíveis;
- Sintaxe de conexão:

```
bool QObject::connect(senderqobjptr,
                      SIGNAL(signalname(argtypelist)),
                      receiverqobjptr,
                      SLOT(slotname(argtypelist))
                      optionalConnectionType);
```

Emissor e *signal* emitido

Classe base de todos os objetos Qt

Signals / Slots

- Um *signal* de um objeto pode ser conectado a *slots* de um ou mais outros objetos, desde que os parâmetros sejam compatíveis;
- Sintaxe de conexão:

```
bool QObject::connect(senderqobjptr,
                      SIGNAL(signalname(argtypelist)),
                      receiverqobjptr,
                      SLOT(slotname(argtypelist))
                      optionalConnectionType);
```

Emissor e *signal* emitido

Receptor e *slot* a ser invocado

Classe base de todos os objetos Qt

Signals / Slots

- Um *signal* de um objeto pode ser conectado a *slots* de um ou mais outros objetos, desde que os parâmetros sejam compatíveis;
- Sintaxe de conexão:

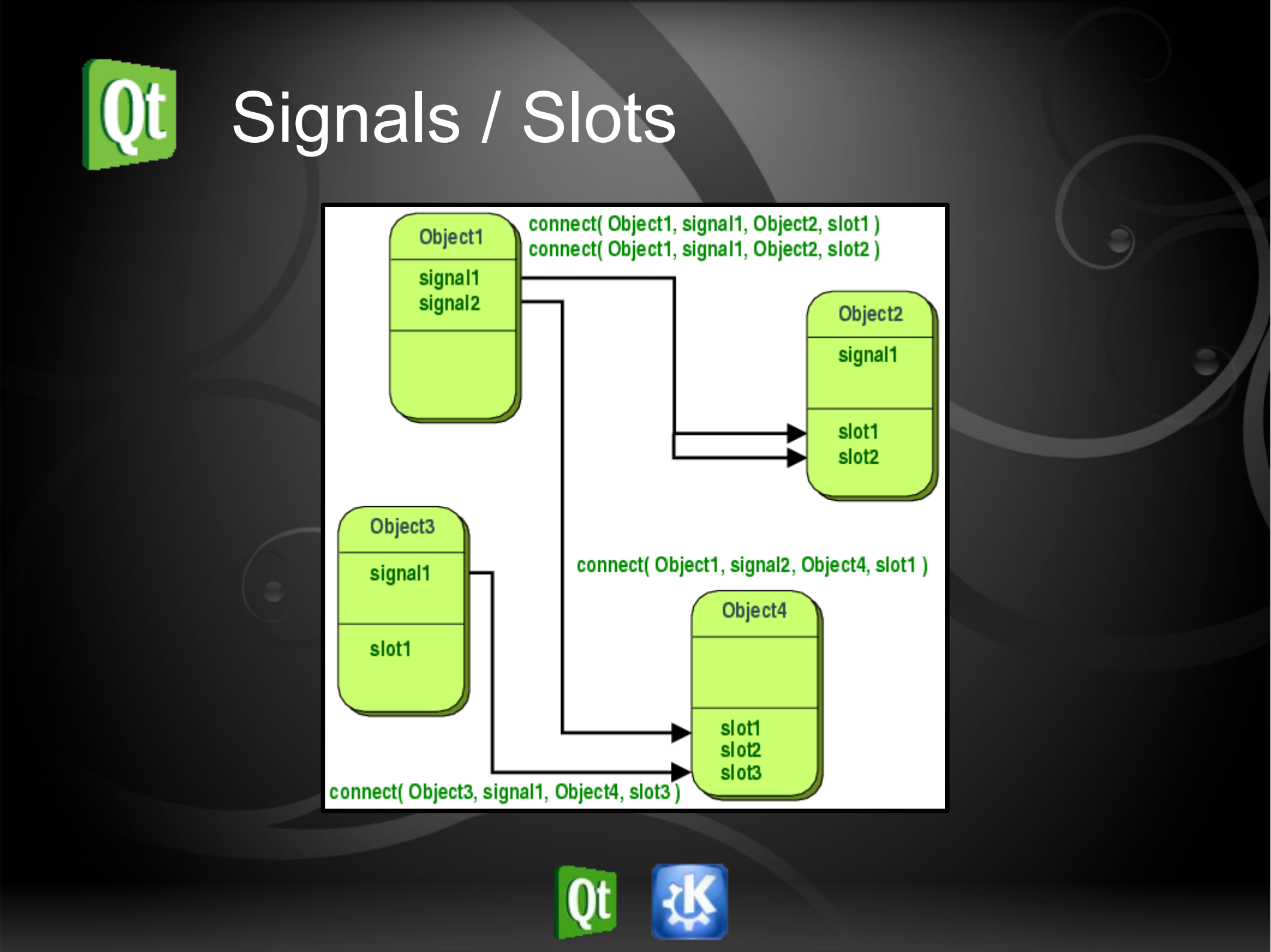
```
bool QObject::connect(senderqobjptr,
                      SIGNAL(signalname(argtypelist)),
                      receiverqobjptr,
                      SLOT(slotname(argtypelist))
                      optionalConnectionType);
```

Emissor e *signal* emitido

Receptor e *slot* a ser invocado

Classe base de todos os objetos Qt

Tipo de conexão: síncrona / assíncrona



Signals / Slots

Signals / Slots

- Um exemplo simples:

Counter.h

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT
public:
    Counter() { m_value = 0; }
    int value() const { return m_value; }
public slots:
    void setValue(int value);
signals:
    void valueChanged(int newValue);
private:
    int m_value;
};
```

Signals / Slots

- Um exemplo simples:

Se deseja-se usar alguma extensão do Qt a classe DEVE derivar de QObject ...

Counter.h

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT
public:
    Counter() { m_value = 0; }
    int value() const { return m_value; }
public slots:
    void setValue(int value);
signals:
    void valueChanged(int newValue);
private:
    int m_value;
};
```

Signals / Slots

- Um exemplo simples:

Counter.h

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT
public:
    Counter() { m_value = 0; }
    int value() const { return m_value; }
public slots:
    void setValue(int value);
signals:
    void valueChanged(int newValue);
private:
    int m_value;
};
```

Se deseja-se usar alguma extensão do Qt a classe DEVE derivar de QObject ...

... e acrescentar a macro Q_OBJECT

Signals / Slots

- Um exemplo simples:

Counter.h

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT
public:
    Counter() { m_value = 0; }
    int value() const { return m_value; }
public slots:
    void setValue(int value);
signals:
    void valueChanged(int newValue);
private:
    int m_value;
};
```

Se deseja-se usar alguma extensão do Qt a classe DEVE derivar de QObject ...

... e acrescentar a macro Q_OBJECT

Daí *slots* e *signals* podem ser declarados

Signals / Slots

- Um exemplo simples:

Counter.cpp

```
void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
```

main.cpp

```
Counter a, b;
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));

a.setValue(12);     // a.value() == 12, b.value() == 12
b.setValue(48);     // a.value() == 12, b.value() == 48
```

Signals / Slots

- Um exemplo simples:

Counter.cpp

```
void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
```

A palavra reservada *emit*, criada pelo Qt, dispara um novo *signal*

main.cpp

```
Counter a, b;
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));

a.setValue(12);     // a.value() == 12, b.value() == 12
b.setValue(48);     // a.value() == 12, b.value() == 48
```

Signals / Slots

- Um exemplo simples:

Counter.cpp

```
void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
```

A palavra reservada *emit*, criada pelo Qt, dispara um novo *signal*

main.cpp

```
Counter a, b;
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));

a.setValue(12);     // a.value() == 12, b.value() == 12
b.setValue(48);     // a.value() == 12, b.value() == 48
```

Conexão entre o *signal* e o *slot*

Signals / Slots

- Considerações importantes:
  - *Slots* são funções comuns do C++: podem ser invocadas diretamente, sobrecarregadas, públicas ou privadas;
  - Um *signal* pode ser conectado a vários *slots*;
  - Mais de um *signal* pode ser conectado ao mesmo *slot* (o emissor pode ser descoberto com *QObject::sender*);
  - Um *signal* pode ser conectado a outro *signal*;
  - Conexões podem ser removidas com *QObject::disconnect*;

Signals / Slots

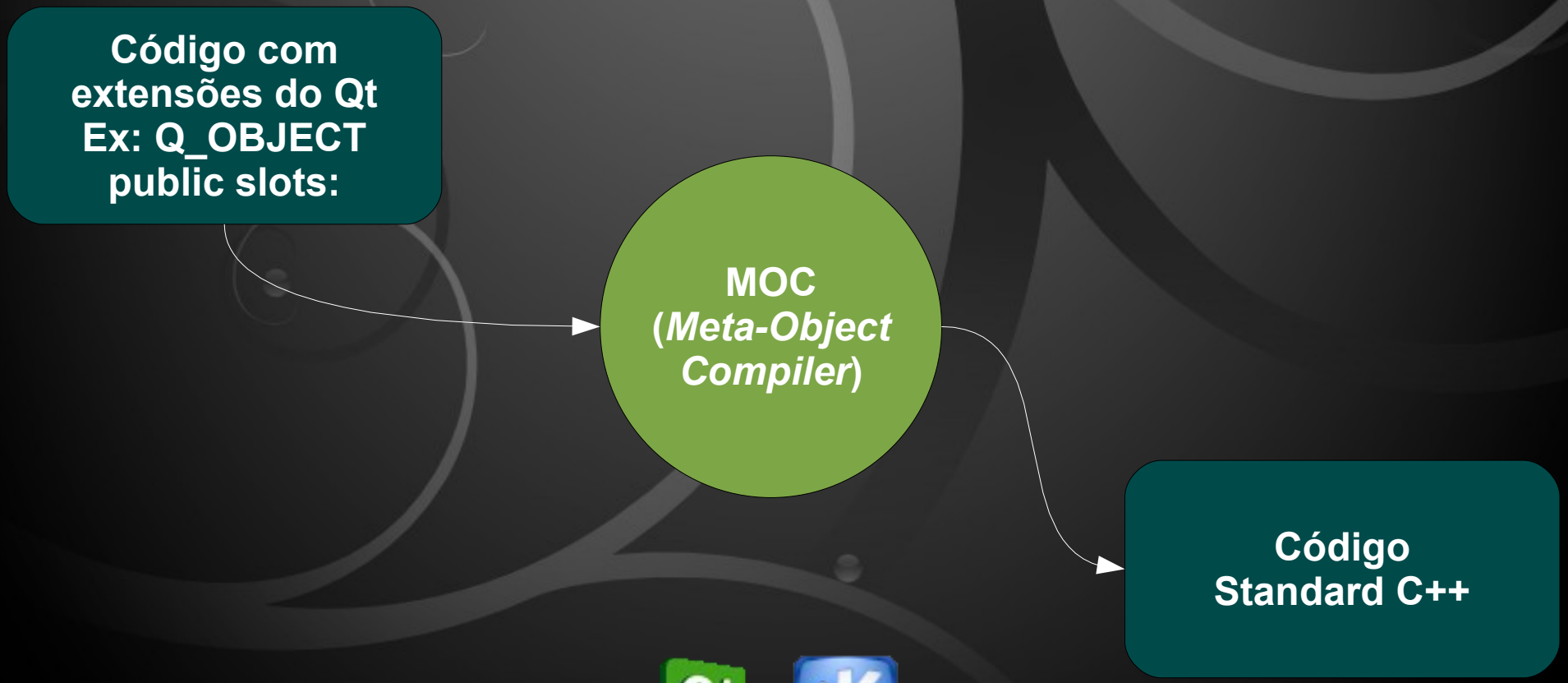
- Considerações importantes:
  - Um *signal* pode ter um número de parâmetros maior ou igual ao número de parâmetros do *slot* conectado;
  - *Signals* e *slots* podem ser utilizados em qualquer classe derivada de *QObject*, não somente *widgets*;
  - Conexões, em *QDialogs*, podem ser automaticamente realizadas (sem requerer o *QObject::connect*).
  - Se as palavras reservadas *signals*, *slots* e *emit* são utilizadas por outra biblioteca (ex. *boost*) pode-se desabilitá-las e usar as macros *Q_SIGNALS*, *Q_SLOTS* e *Q_EMIT*.

Signals / Slots

- Como *signals* e *slots* são implementados ?
  - As palavras reservadas *signals*, *slots, emit* (bem como *foreach* e recursos de meta-objetos, propriedades etc) não estão presentes no Standard C++;
  - Essas extensões são tratadas pelo MOC (*Meta-Object Compiler*);
  - O *QMake* verifica quais classes, declaradas na variável *HEADER*, utilizam a macro *Q_OBJECT* e automaticamente inclui a invocação do *moc* no arquivos *Makefile* gerados.

Signals / Slots

- Como *signals* e *slots* são implementados ?
  - O MOC (*Meta-Object Compiler*):



Layout e Parentesco

- *Layouts* gerenciam a geometria dos *widgets* de uma janela;
- Um *widget* pode ter uma relação de parentesco com outro *widget*;
- Exemplo:

Layout e Parentesco

- *Layouts* gerenciam a geometria dos *widgets* de uma janela;
- Um *widget* pode ter uma relação de parentesco com outro *widget*;
- Exemplo:

Layout e Parentesco

- *Layouts* gerenciam a geometria dos *widgets* de uma janela;
- Um *widget* pode ter uma relação de parentesco com outro *widget*;
- Exemplo:

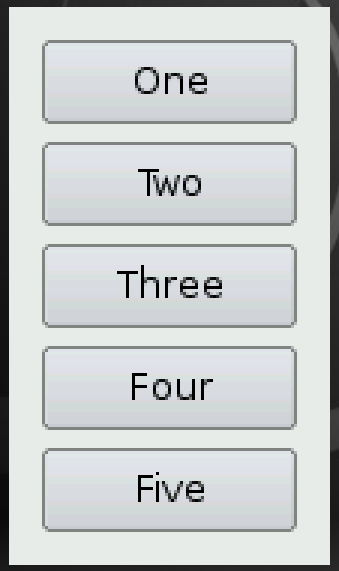
Layout e Parentesco

- *Layouts* gerenciam a geometria dos *widgets* de uma janela;
- Um *widget* pode ter uma relação de parentesco com outro *widget*;
- Exemplo:

Layout e Parentesco

- Tipos de *layouts*:
  - *QHBoxLayout*:

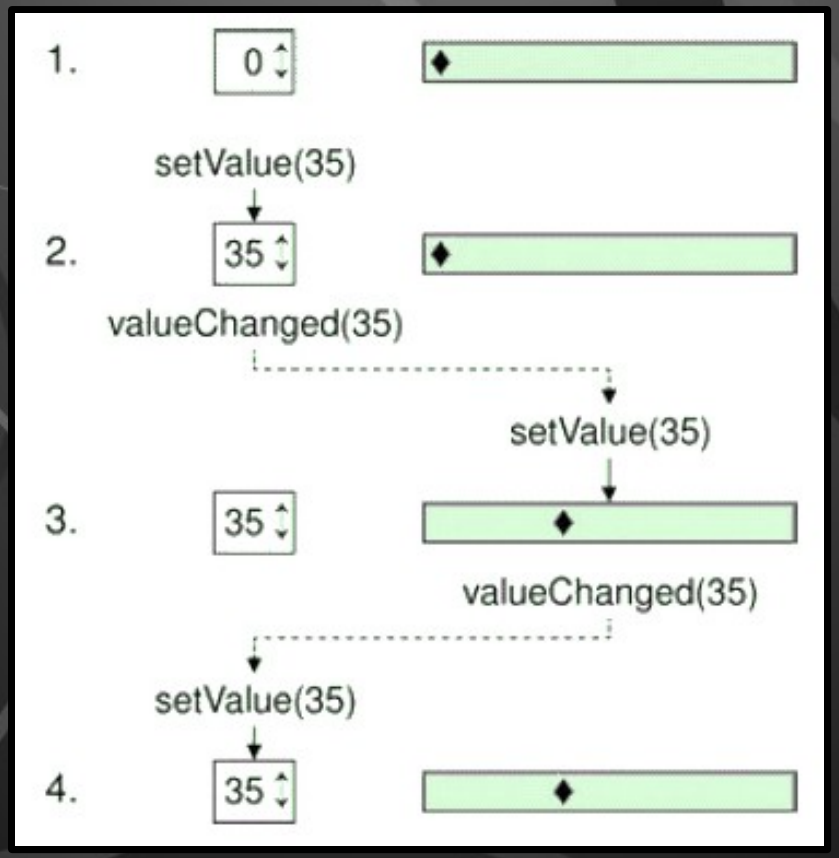
  - *QVBoxLayout*:



QGridLayout:

Layout e Parentesco

- Conexões de *signals* e *slots*:



Layout e Parentesco

```
#include <QApplication>
#include <QSlider>
#include <QHBoxLayout>
#include <QSpinBox>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *window = new QWidget;
    window->setWindowTitle("Enter Your Age");
    QSpinBox *spinBox = new QSpinBox;
    QSlider *slider = new QSlider(Qt::Horizontal);
    spinBox->setRange(0, 130);
    slider->setRange(0, 130);
    QObject::connect(spinBox, SIGNAL(valueChanged(int)),
                     slider, SLOT(setValue(int)));
    QObject::connect(slider, SIGNAL(valueChanged(int)),
                     spinBox, SLOT(setValue(int)));
    spinBox->setValue(35);
    QHBoxLayout *layout = new QHBoxLayout;
    layout->addWidget(spinBox);
    layout->addWidget(slider);
    window->setLayout(layout);
    window->show();
    return app.exec();
}
```

lab2 / main.cpp

Layout e Parentesco

```
#include <QApplication>
#include <QSlider>
#include <QHBoxLayout>
#include <QSpinBox>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *window = new QWidget;
    window->setWindowTitle("Enter Your Age");
    QSpinBox *spinBox = new QSpinBox;
    QSlider *slider = new QSlider(Qt::Horizontal);
    spinBox->setRange(0, 130);
    slider->setRange(0, 130);
    QObject::connect(spinBox, SIGNAL(valueChanged(int)),
                     slider, SLOT(setValue(int)));
    QObject::connect(slider, SIGNAL(valueChanged(int)),
                     spinBox, SLOT(setValue(int)));
    spinBox->setValue(35);
    QHBoxLayout *layout = new QHBoxLayout;
    layout->addWidget(spinBox);
    layout->addWidget(slider);
    window->setLayout(layout);
    window->show();
    return app.exec();
}
```

QWidget (*top-level window*)

lab2 / main.cpp

Layout e Parentesco

```
#include <QApplication>
#include <QSlider>
#include <QHBoxLayout>
#include <QSpinBox>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *window = new QWidget;
    window->setWindowTitle("Enter Your Age");
    QSpinBox *spinBox = new QSpinBox;
    QSlider *slider = new QSlider(Qt::Horizontal);
    spinBox->setRange(0, 130);
    slider->setRange(0, 130);
    QObject::connect(spinBox, SIGNAL(valueChanged(int)),
                     slider, SLOT(setValue(int)));
    QObject::connect(slider, SIGNAL(valueChanged(int)),
                     spinBox, SLOT(setValue(int)));
    spinBox->setValue(35);
    QHBoxLayout *layout = new QHBoxLayout;
    layout->addWidget(spinBox);
    layout->addWidget(slider);
    window->setLayout(layout);
    window->show();
    return app.exec();
}
```

QWidget (*top-level window*)

Realização das conexões

lab2 / main.cpp

Layout e Parentesco

```
#include <QApplication>
#include <QSlider>
#include <QHBoxLayout>
#include <QSpinBox>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *window = new QWidget;
    window->setWindowTitle("Enter Your Age");
    QSpinBox *spinBox = new QSpinBox;
    QSlider *slider = new QSlider(Qt::Horizontal);
    spinBox->setRange(0, 130);
    slider->setRange(0, 130);
    QObject::connect(spinBox, SIGNAL(valueChanged(int)),
                     slider, SLOT(setValue(int)));
    QObject::connect(slider, SIGNAL(valueChanged(int)),
                     spinBox, SLOT(setValue(int)));
    spinBox->setValue(35);
    QHBoxLayout *layout = new QHBoxLayout;
    layout->addWidget(spinBox);
    layout->addWidget(slider);
    window->setLayout(layout);
    window->show();
    return app.exec();
}
```

QWidget (*top-level window*)

Realização das conexões

Criação do *layout* e inclusão dos *widgets*

lab2 / main.cpp

Layout e Parentesco

```
#include <QApplication>
#include <QSlider>
#include <QHBoxLayout>
#include <QSpinBox>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *window = new QWidget;
    window->setWindowTitle("Enter Your Age");
    QSpinBox *spinBox = new QSpinBox;
    QSlider *slider = new QSlider(Qt::Horizontal);
    spinBox->setRange(0, 130);
    slider->setRange(0, 130);
    QObject::connect(spinBox, SIGNAL(valueChanged(int)),
                     slider, SLOT(setValue(int)));
    QObject::connect(slider, SIGNAL(valueChanged(int)),
                     spinBox, SLOT(setValue(int)));
    spinBox->setValue(35);
    QHBoxLayout *layout = new QHBoxLayout;
    layout->addWidget(spinBox);
    layout->addWidget(slider);
    window->setLayout(layout);
    window->show();
    return app.exec();
}
```

QWidget (*top-level window*)

Realização das conexões

Criação do *layout* e inclusão dos *widgets*

Neste ponto os *widgets* do *layout* são adotados por *window*

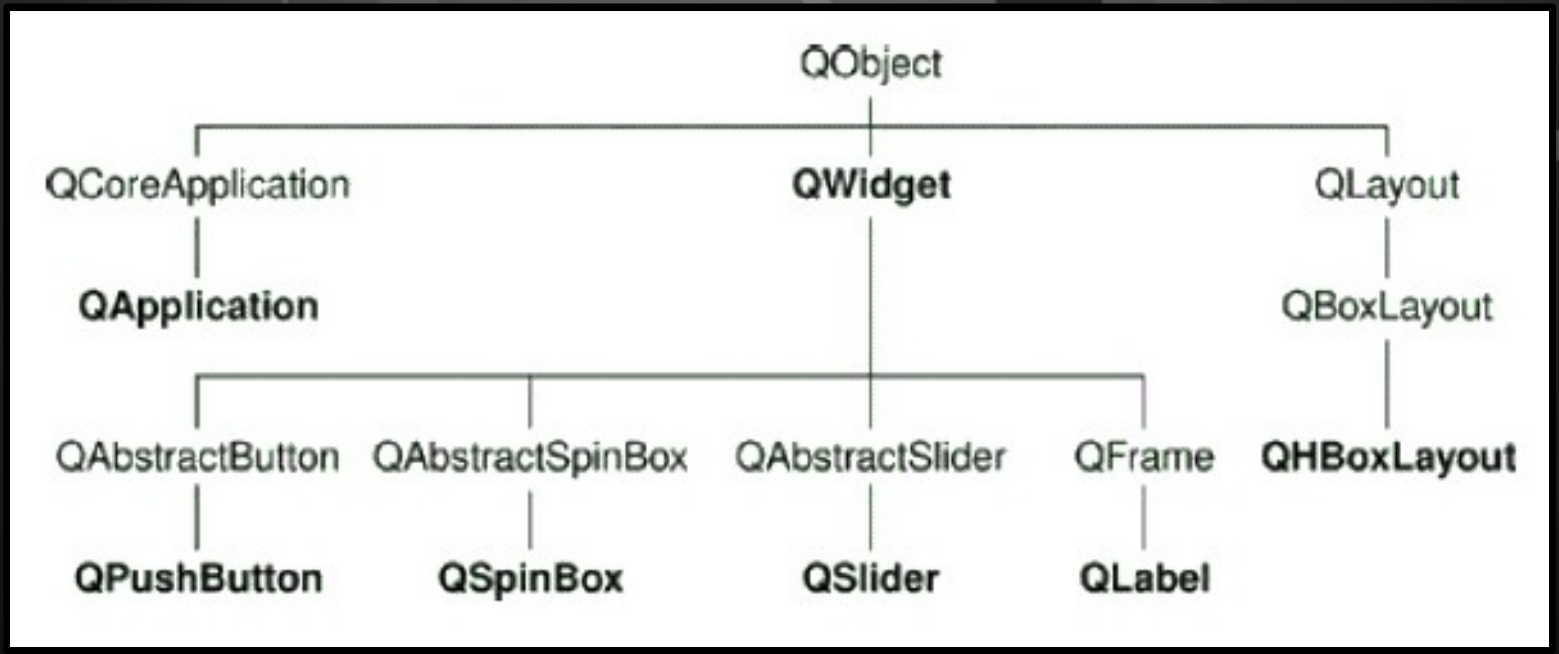
lab2 / main.cpp

Layout e Parentesco

- Considerações sobre relações de parentesco:
  - O pai automaticamente assume a responsabilidade de liberação de memória de todos os filhos;
  - Os filhos são liberados quando o pai sair do escopo;
  - Os únicos objetos a serem deletados manualmente são os criados com *new* e que não possuem pai;
  - Se um filho é deletado antes do pai ele é automaticamente excluído da lista de filhos do pai;
  - *Widgets* filhos são vistos dentro da área do pai;
  - A execução do *show()* no pai automaticamente exibe todos os filhos.

Layout e Parentesco

- Hierarquia das classes utilizadas:

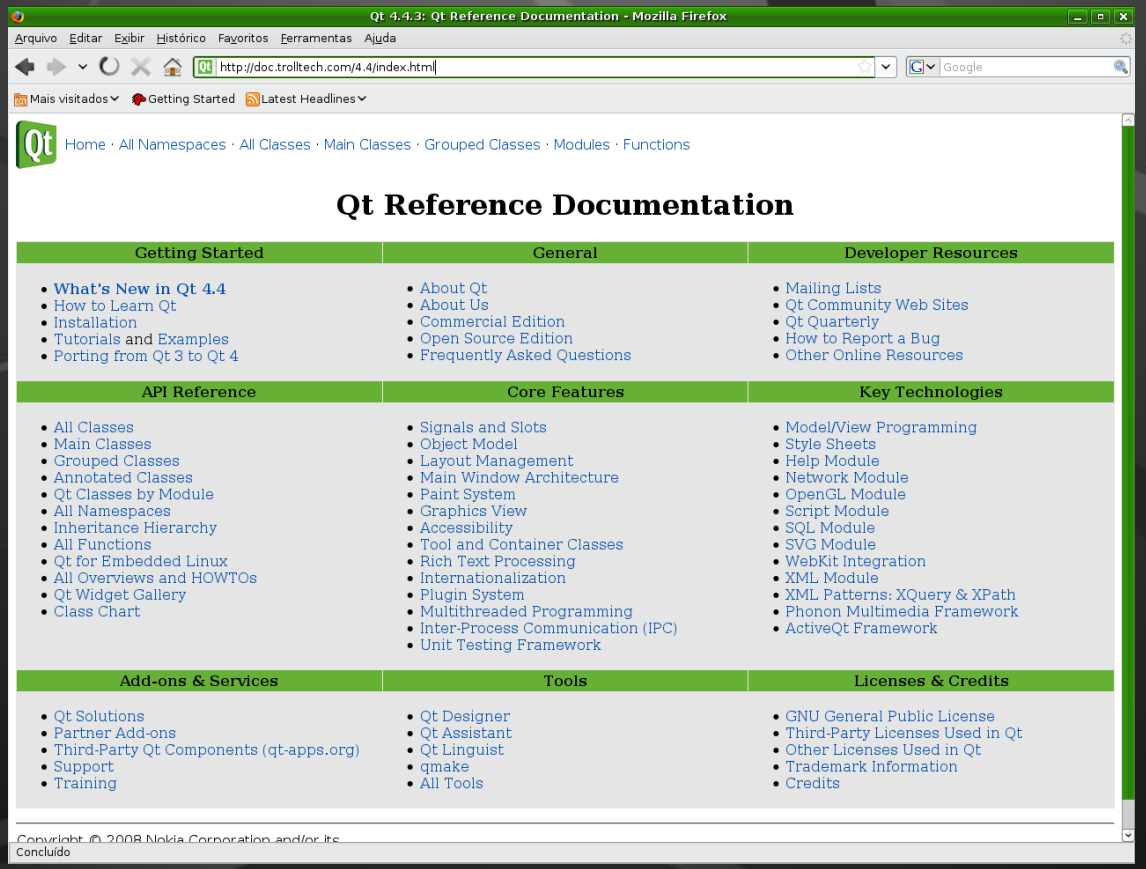


Qt e IDE's

- Principais IDE's multi-plataforma:
  - KDevelop (http://www.kdevelop.org);
  - XCode (http://developer.apple.com/tools/xcode/);
  - Visual Studio (http://msdn.microsoft.com/en-us/vstudio);
  - Eclipse (http://www.eclipse.org/)
  - Edyuk (http://www.edyuk.org)
  - Greenhouse (http://trolltech.com/developer/greenhouse)
  - QDevelop (http://www.qdevelop.org/)

Qt Reference Documentation

- http://doc.trolltech.com/4.4/

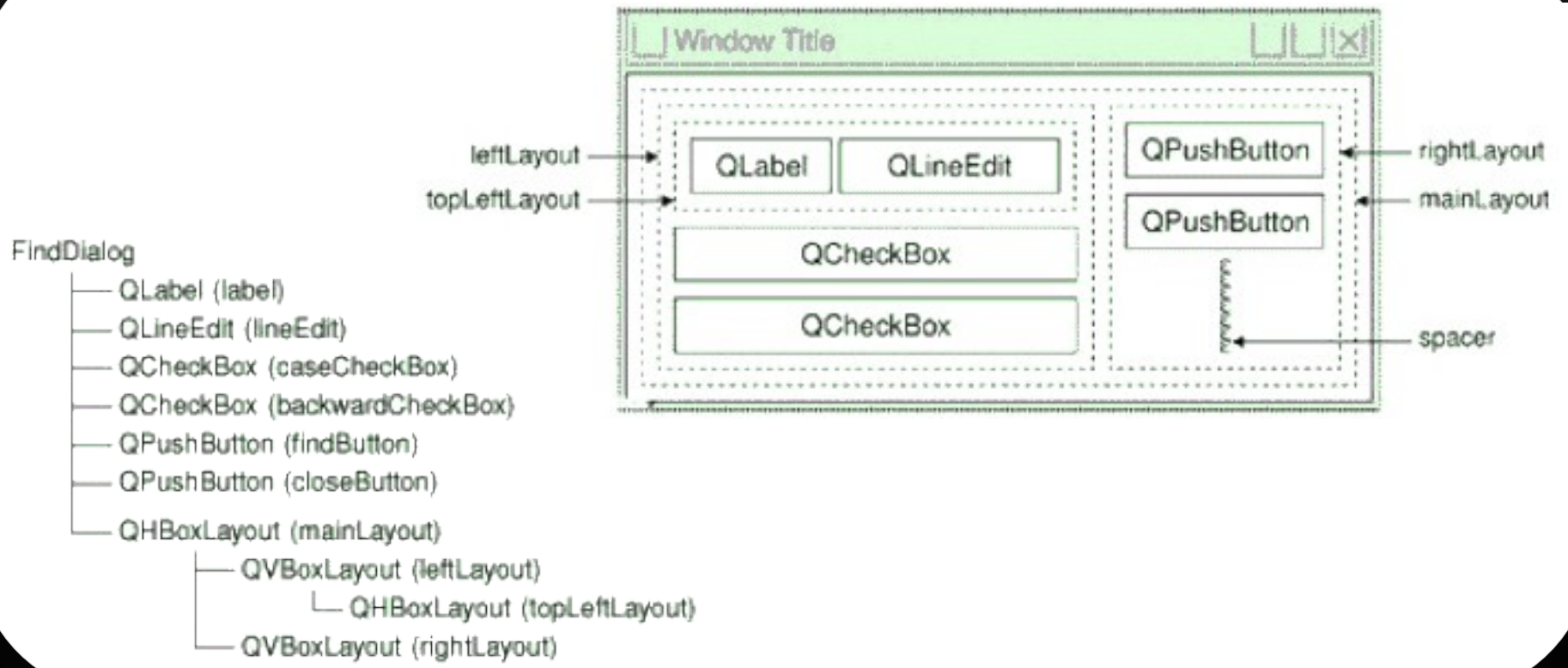


Main Window e Dialogs

- Uma aplicação é geralmente formada por uma tela principal (*QMainWindow*) e por várias outras telas (*QDialog*);
- Essas telas podem ser criadas manualmente ou através do *Qt Designer*;
- Exemplo:

Main Window e Dialogs

- *Layout* e relações de parentesco:



Main Window e Dialogs

```
#include <QDialog>

class QCheckBox;          class QLabel;
class QLineEdit;          class QPushButton;

class FindDialog : public QDialog
{
        Q_OBJECT
public:
        FindDialog(QWidget *parent = 0);
signals:
        void findNext(const QString &str, Qt::CaseSensitivity cs);
        void findPrevious(const QString &str, Qt::CaseSensitivity cs);
private slots:
        void findClicked();
        void enableFindButton(const QString &text);
private:
        QLabel *label;
        QLineEdit *lineEdit;
        QCheckBox *caseCheckBox;
        QCheckBox *backwardCheckBox;
        QPushButton *findButton;
        QPushButton *closeButton;
};
```

lab3 / finddialog.h

Main Window e Dialogs

```
#include <QDialog>

class QCheckBox;          class QLabel;
class QLineEdit;          class QPushButton;

class FindDialog : public QDialog
{
    Q_OBJECT
public:
    FindDialog(QWidget *parent = 0);
signals:
    void findNext(const QString &str, Qt::CaseSensitivity cs);
    void findPrevious(const QString &str, Qt::CaseSensitivity cs);
private slots:
    void findClicked();
    void enableFindButton(const QString &text);
private:
    QLabel *label;
    QLineEdit *lineEdit;
    QCheckBox *caseCheckBox;
    QCheckBox *backwardCheckBox;
    QPushButton *findButton;
    QPushButton *closeButton;
};
```

Toda nova tela deriva de *QDialog*

lab3 / finddialog.h

Main Window e Dialogs

```
#include <QDialog>

class QCheckBox;          class QLabel;
class QLineEdit;          class QPushButton;

class FindDialog : public QDialog
{
    Q_OBJECT
public:
    FindDialog(QWidget *parent = 0);
signals:
    void findNext(const QString &str, Qt::CaseSensitivity cs);
    void findPrevious(const QString &str, Qt::CaseSensitivity cs);
private slots:
    void findClicked();
    void enableFindButton(const QString &text);
private:
    QLabel *label;
    QLineEdit *lineEdit;
    QCheckBox *caseCheckBox;
    QCheckBox *backwardCheckBox;
    QPushButton *findButton;
    QPushButton *closeButton;
};
```

Toda nova tela deriva de *QDialog*

Construtor típico de *widgets*, aceitando o *parent* como parâmetro

lab3 / finddialog.h

Main Window e Dialogs

```
#include <QDialog>

class QCheckBox;          class QLabel;
class QLineEdit;          class QPushButton;

class FindDialog : public QDialog
{
    Q_OBJECT
public:
    FindDialog(QWidget *parent = 0);
signals:
    void findNext(const QString &str, Qt::CaseSensitivity cs);
    void findPrevious(const QString &str, Qt::CaseSensitivity cs);
private slots:
    void findClicked();
    void enableFindButton(const QString &text);
private:
    QLabel *label;
    QLineEdit *lineEdit;
    QCheckBox *caseCheckBox;
    QCheckBox *backwardCheckBox;
    QPushButton *findButton;
    QPushButton *closeButton;
};
```

Toda nova tela deriva de *QDialog*

Construtor típico de *widgets*, aceitando o *parent* como parâmetro

Signals que a tela poderá emitir

lab3 / finddialog.h

Main Window e Dialogs

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

FindDialog::FindDialog(QWidget *parent) : QDialog(parent)
{
        label = new QLabel(tr("Find &what:"));
        lineEdit = new QLineEdit;
        label->setBuddy(lineEdit);

        caseCheckBox = new QCheckBox(tr("Match &case"));
        backwardCheckBox = new QCheckBox(tr("Search &backward"));

        findButton = new QPushButton(tr("&Find"));
        findButton->setDefault(true);
        findButton->setEnabled(false);

        closeButton = new QPushButton(tr("Close"));

        connect(lineEdit, SIGNAL(textChanged(const QString &)),
                this, SLOT(enableFindButton(const QString &)));
        connect(findButton, SIGNAL(clicked()),
                this, SLOT(findClicked()));
        connect(closeButton, SIGNAL(clicked()),
                this, SLOT(close()));
```

lab3 / finddialog.cpp

Main Window e Dialogs

```
    QHBoxLayout *topLeftLayout = new QHBoxLayout;
    topLeftLayout->addWidget(label);
    topLeftLayout->addWidget(lineEdit);

    QVBoxLayout *leftLayout = new QVBoxLayout;
    leftLayout->addLayout(topLeftLayout);
    leftLayout->addWidget(caseCheckBox);
    leftLayout->addWidget(backwardCheckBox);

    QVBoxLayout *rightLayout = new QVBoxLayout;
    rightLayout->addWidget(findButton);
    rightLayout->addWidget(closeButton);
    rightLayout->addStretch();

    QHBoxLayout *mainLayout = new QHBoxLayout;
    mainLayout->addLayout(leftLayout);
    mainLayout->addLayout(rightLayout);
    setLayout(mainLayout);

    setWindowTitle(tr("Find"));
    setFixedHeight(sizeHint().height());
}
```

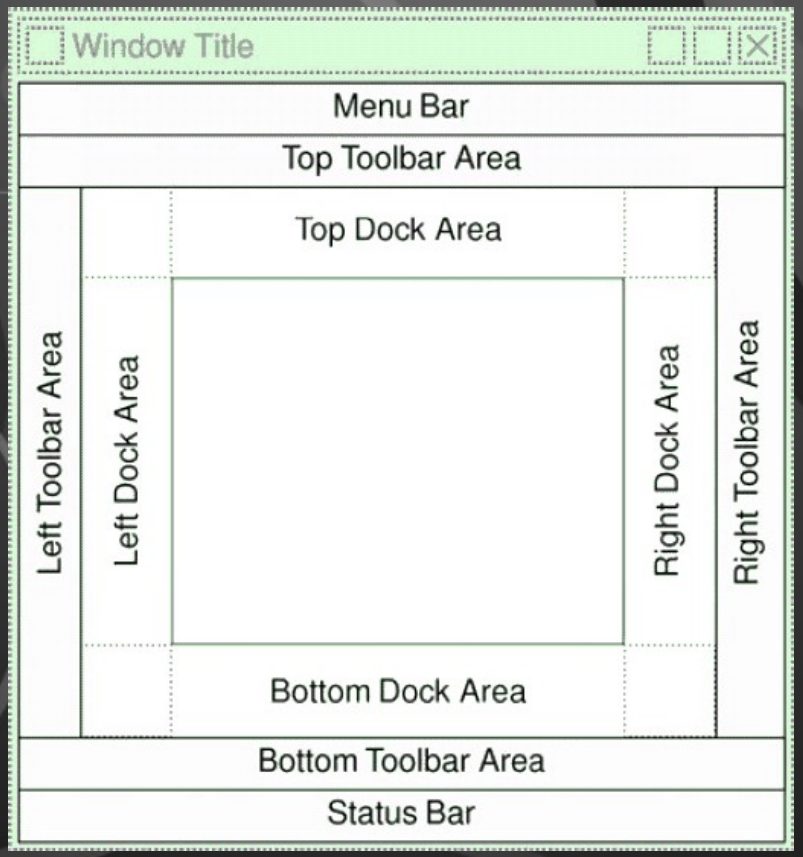
lab3 / finddialog.cpp

Main Window e Dialogs

- *MainWindows* são *widgets* que podem conter menus, *toolbars* e barra de status;
- *MainWindows* são criadas através da derivação da classe *QMainWindow*;
- O *Qt Designer* facilita sobremaneira a criação de *MainWindows*

Main Window e Dialogs

- Áreas definidas pela *QMainWindow*:



Main Window e Dialogs

- Menus e *toolbars* baseiam-se no conceito de *action*:
  - *Um action* é um item que pode ser adicionado a qualquer número de menus e *toolbars*.
- Criar menus e *toolbars* requer três passos:
  - Criação e configuração dos *actions*;
  - Criação do menu e utilização dos *actions*;
  - Criação dos *toolbars* e utilização dos *actions*.
- Todo *action* possui o *signal triggered()*.

Main Window e Dialogs

- Criando um *action*:

```
QAction *newAction = new QAction(tr("&New"), this);
newAction->setIcon(QIcon(":/images/new.png"));
newAction->setShortcut(tr("Ctrl+N"));
newAction->setStatusTip(tr("Create a new spreadsheet file"));
connect(newAction, SIGNAL(triggered()), this, SLOT(newFile()));
```

Main Window e Dialogs

- Inserindo *actions* em *menus* e *toolbars*:

```
QMenu *fileMenu = menuBar()->addMenu(tr("&File"));
fileMenu->addAction(newAction);
fileMenu->addAction(openAction);
fileMenu->addAction(saveAction);
fileMenu->addAction(saveAsAction);
```

Main Window e Dialogs

- Inserindo *actions* em *menus* e *toolbars*:

```
QToolBar *fileToolBar = addToolBar(tr("&File"));
fileToolBar->addAction(newAction);
fileToolBar->addAction(openAction);
fileToolBar->addAction(saveAction);
QToolBar *editToolBar = addToolBar(tr("&Edit"));
editToolBar->addAction(cutAction);
editToolBar->addSeparator();
editToolBar->addAction(findAction);
```

Main Window e Dialogs

- Criando a barra de *status*:

```
QLabel *locationLabel = new QLabel(" W999 ");
QLabel *lormulaLabel = new QLabel;
statusBar()->addWidget(locationLabel);
statusBar()->addWidget(formulaLabel, 1);
connect(spreadsheet, SIGNAL(currentCellChanged(int, int, int, int)),
            this, SLOT(updateStatusBar()));
connect(spreadsheet, SIGNAL(modified()),
            this, SLOT(spreadsheetModified()));
updateStatusBar();
```

Main Window e Dialogs

- Usando um *message box*:

```
int r = QMessageBox::warning(this, tr("Spreadsheet"),
                             tr("The document has been modified.\n"
                             "Do you want to save your changes?"),
                             QMessageBox::Yes | QMessageBox::Default,
                             QMessageBox::No,
                             QMessageBox::Cancel | QMessageBox::Escape);
if (r == QMessageBox::Yes) return save();
else if (r == QMessageBox::Cancel) return false;
```

Main Window e Dialogs

- As imagens utilizadas pela aplicação (ícones etc) podem ser carregadas de três formas:
  - Através da leitura, em tempo de execução, dos arquivos das imagens;
  - Através da inclusão de imagens XPM no código-fonte;
  - Através do uso do mecanismo de *resources* do Qt.

Main Window e Dialogs

- *Resources*:
  - O arquivo de recursos de uma aplicação Qt indica quais arquivos serão diretamente incluídos no executável a ser gerado. Dessa forma, ícones não são perdidos;
  - Na verdade, qualquer tipo de arquivo (não somente imagens) pode ser incluído no executável/
  - Arquivos de recursos podem ser organizados em categorias;
  - Recursos são utilizados com o prefixo "*:/*".

Main Window e Dialogs

- Lab4: criando um *MainWindow* no *Qt Designer*;
- Dicas sobre *actions:*
  - Pode-se especificar aceleradores colocando o caracter '&' antes do caracter acelerador;
  - Pode-se especificar teclas de atalho na propriedade *shorcut*;
  - Pode-se especificar *tips* de *toolbar* e mensagens de status nas propriedades *toolTip* e *statusTip*.

Main Window e Dialogs

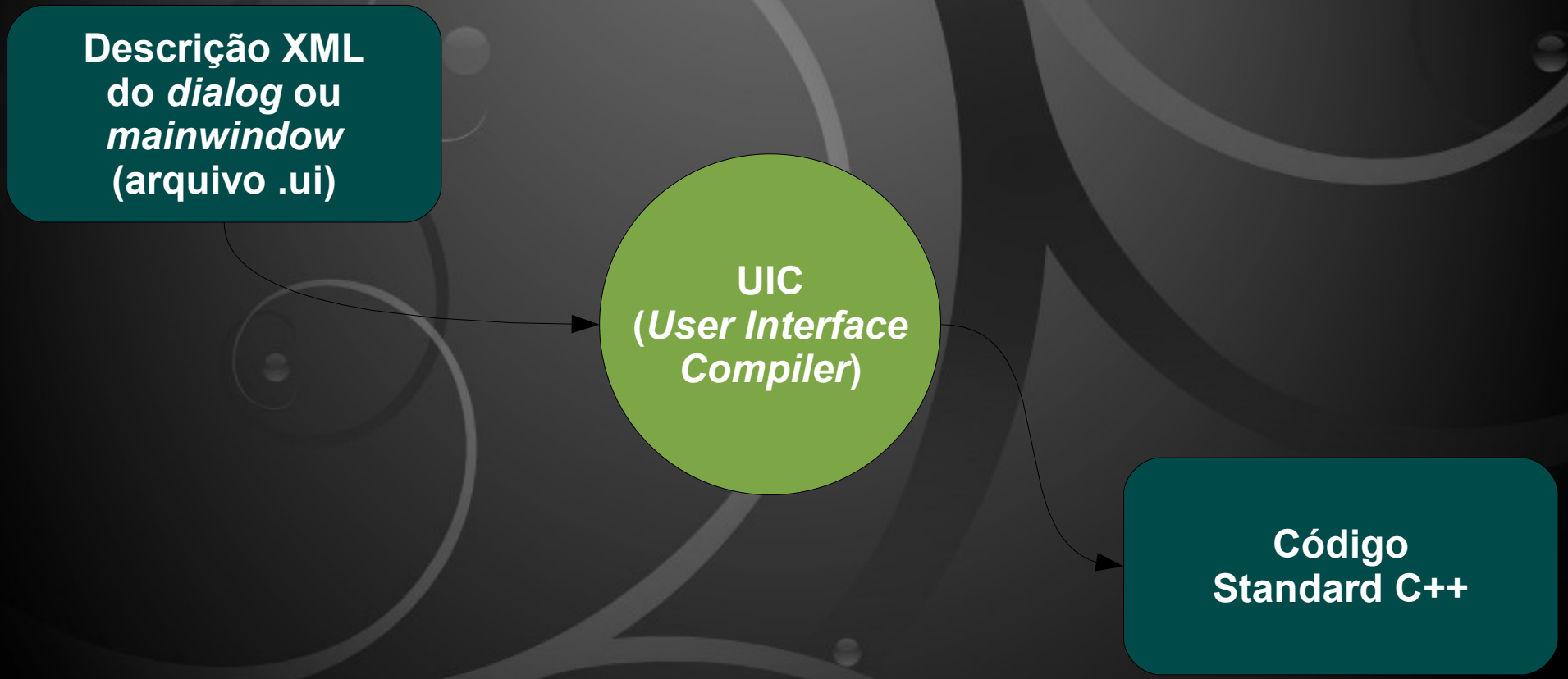
- Lab4: criando um *MainWindow* no *Qt Designer*:

```
#include <QApplication>
#include <QDialog>
#include "ui_cdmainwindow.h"
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    Ui::CdMainWindow mw;
    mw.setupUi();
    mw.show();
    return app.exec();
}
```

lab4 / main.cpp

Main Window e Dialogs

- Entendendo o *Qt Designer*:



Main Window e Dialogs

- Implementando funcionalidade:

lab5 / cdmainwindow.h

```
#include <QDialog>
#include "ui_cdmainwindow.h"
class CdMainWindow : public QMainWindow, public Ui::CdMainWindow
{
    Q_OBJECT
public:
    CdMainWindow(QWidget *parent = 0);
private slots:
    void on_lineEdit_textChanged();
};
```

Main Window e Dialogs

- Implementando funcionalidade:

Estende-se QmainWindow e a classe gerada pelo *Qt Designer*

```
#include <QDialog>
#include "ui_cdmainwindow.h"
class CdMainWindow : public QMainWindow, public Ui::CdMainWindow
{
    Q_OBJECT
public:
    CdMainWindow(QWidget *parent = 0);
private slots:
    void on_lineEdit_textChanged();
};
```

lab5 / cdmainwindow.h

Main Window e Dialogs

- Implementando funcionalidade:

Estende-se QmainWindow e a classe gerada pelo *Qt Designer*

```
#include <QDialog>
#include "ui_cdmainwindow.h"
class CdMainWindow : public QMainWindow, public Ui::CdMainWindow
{
    Q_OBJECT
public:
    CdMainWindow(QWidget *parent = 0);
private slots:
    void on_lineEdit_textChanged();
};
```

Conexões automáticas de *signals* e *slots*

lab5 / cdmainwindow.h

Main Window e Dialogs

- Implementando funcionalidade:

lab5 / cdmainwindow.cpp

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

CdMainWindow::CdMainWindow(QWidget *parent) : QMainWindow(parent) {
        setupUi(this);
        QRegExp regExp("[A-Za-z][1-9][0-9]{0,2}");
        lineEdit->setValidator(new QRegExpValidator(regExp, this));
}
void CdMainWindow::on_lineEdit_textChanged() {
        okButton->setEnabled(lineEdit->hasAcceptableInput());
}
```

Main Window e Dialogs

- Implementando funcionalidade:

setupUi() deve sempre ser invocado no construtor

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

CdMainWindow::CdMainWindow(QWidget *parent) : QMainWindow(parent) {
    setupUi(this);
    QRegExp regExp("[A-Za-z][1-9][0-9]{0,2}");
    lineEdit->setValidator(new QRegExpValidator(regExp, this));
}
void CdMainWindow::on_lineEdit_textChanged() {
    okButton->setEnabled(lineEdit->hasAcceptableInput());
}
```

lab5 / cdmainwindow.cpp

Main Window e Dialogs

- Implementando funcionalidade:

setupUi() deve sempre ser invocado no construtor

cdmainwindow.cpp

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

CdMainWindow::CdMainWindow(QWidget *parent) : QMainWindow(parent) {
    setupUi(this);
    QRegExp regExp("[A-Za-z][1-9][0-9]{0,2}");
    lineEdit->setValidator(new QRegExpValidator(regExp, this));
}
void CdMainWindow::on_lineEdit_textChanged() {
    okButton->setEnabled(lineEdit->hasAcceptableInput());
}
```

Implementação do *slot*

Main Window e Dialogs

- Implementando funcionalidade:

```
#include <QApplication>
#include "cdmainwindow.h"
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    CdMainWindow *mw = new CdMainWindow;
    mw->show();
    return app.exec();
}
```

lab5 / main.cpp

Main Window e Dialogs

- Usando *modeless dialogs*:

```
MainWindow::MainWindow(...) {
    ...
    findDialog = 0;
    ...
}
void MainWindow::on_findAction_triggered() {
    if (!findDialog) {
        findDialog = new FindDialog(this);
    }
    findDialog->show();
    findDialog->activateWindow();
}
```

Main Window e Dialogs

- Usando *modal dialogs*:

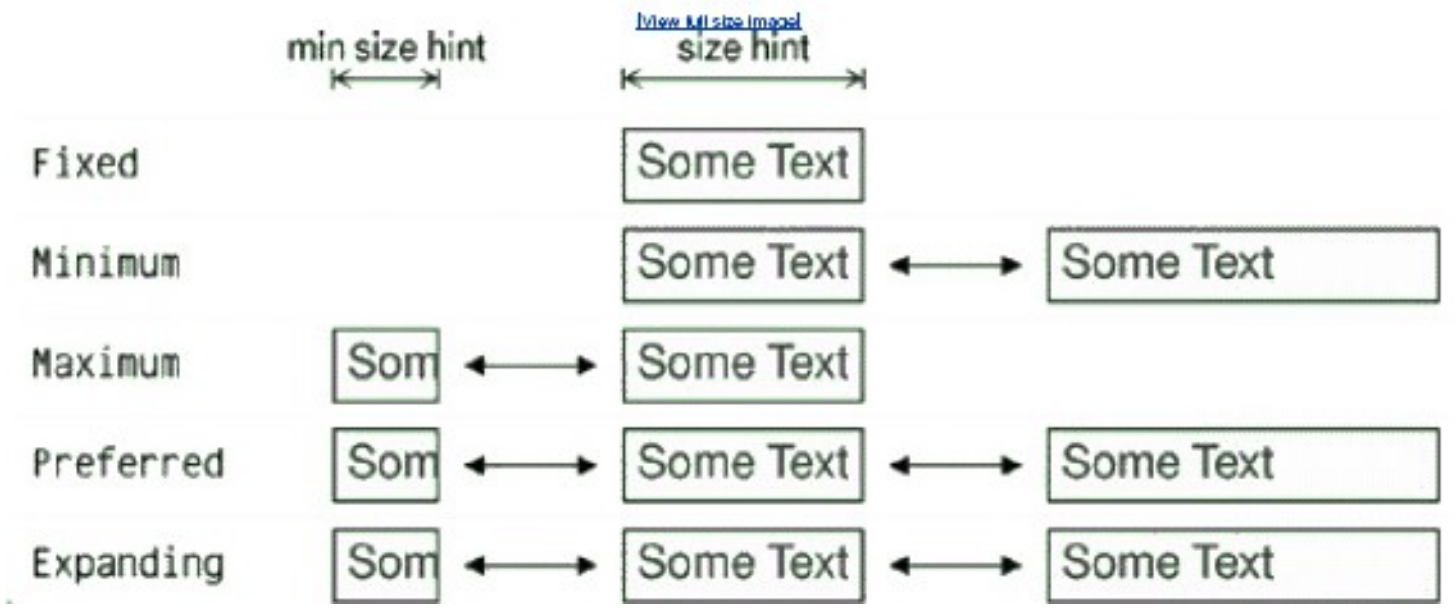
```
void MainWindow::addCd()
{
    AddCdDialog addCdDialog(this);
    if (addCdDialog.exec()) {
        // Retorna QDialog::Accepted ou QDialog::Reject
        ...
    }
}
```

Main Window e Dialogs

- *Size policies*: especificam como o *widget* será alterado pelo *layout manager*:
  - Fixed: o *widget* não pode aumentar ou diminuir. Será sempre mantido no seu tamanho atual;
  - Minimum: o tamanho atual é o tamanho mínimo do *widget*;
  - Maximum: o tamanho atual é o tamanho máximo do *widget*;
  - Preferred: o tamanho atual é o preferido, mas ele podem aumentar ou diminuir;
  - Expanding: o *widget* é propenso a aumentar.

Main Window e Dialogs

- *Size policies*:



Model-View

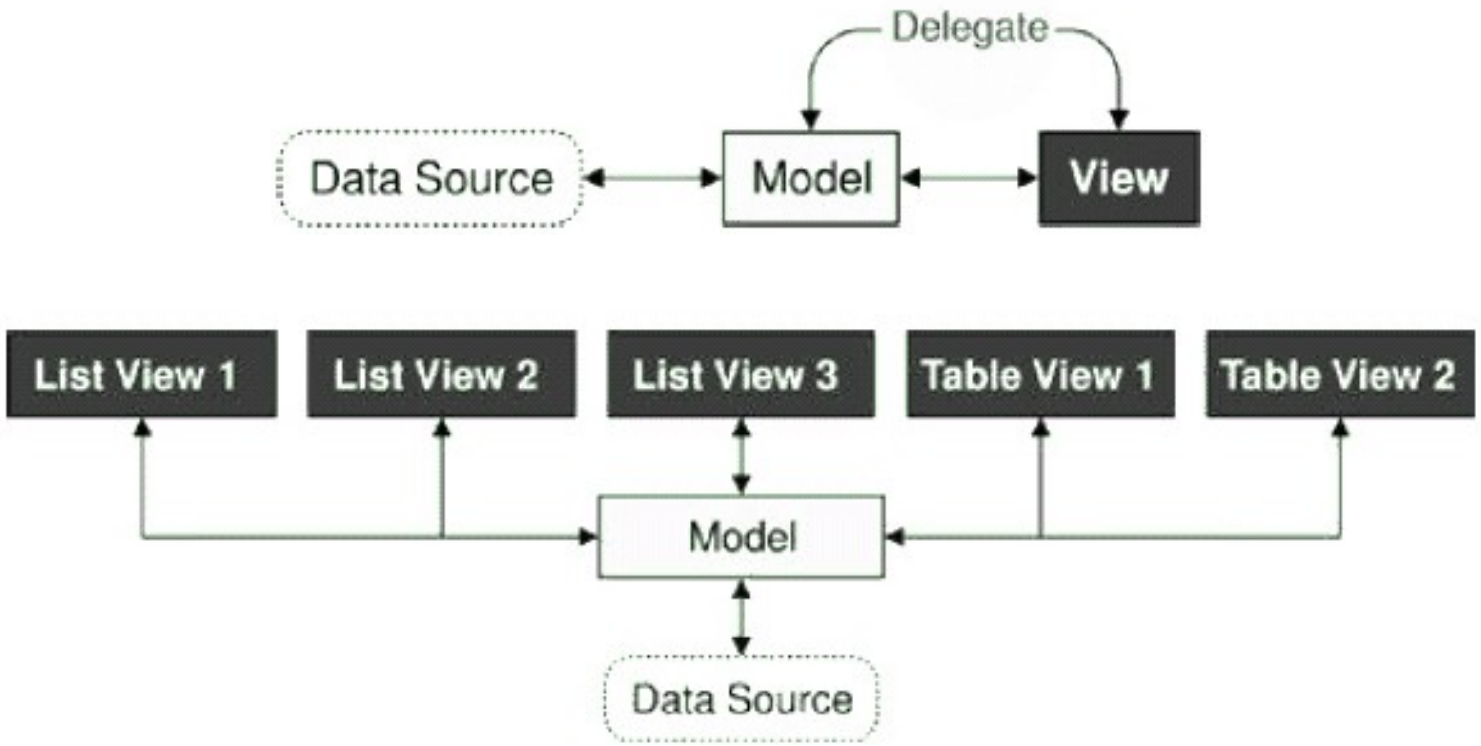
- Muitas aplicações permitem a busca, visualização e edição de itens individuais de um conjunto de dados.
- Em versões anteriores do Qt, os *widgets* eram populados com todo o conteúdo do conjunto de dados, estes eram alterados e posteriormente atualizados na sua origem.
- Entretanto, esta abordagem não é escalável para grandes conjuntos e não resolve o problema da visualização do mesmo conjunto ou dois ou mais *widgets* diferentes.

Model-View

- O Qt4 usa o padrão *Model-View*, uma variação do *model-view-controller*, para resolver esses problemas:
  - *Model*: representa o conjunto de dados e é responsável pela aquisição e atualização de dados a partir da origem. Cada tipo de dados tem seu próprio modelo, porém a API provida pelos modelos aos *views* é padrão;
  - *View*: apresenta os dados para o usuário. Para grandes conjuntos, exibe somente o necessário (*lazy-load*);
  - *Controller*: mediador entre o usuário e o *view*. Converte ações do usuário em requisições para navegar e editar dados, os quais são enviados pelo *view* para o *model*.

Model-View

- *Qt4 Model-View*:



Model-View

- *Qt4 Model-View*:
  - O *delegate* é usado para ter total controle sobre como os itens são apresentados e editados;
  - O Qt disponibiliza um *delegate* padrão para cada tipo de *view*;
  - Os modelos podem obter somente os dados que são realmente necessários;
  - Cada item de dados em um modelo é representado por um *index*;
  - O Qt traz um conjunto de modelos e outros podem ser criados.

Model-View

- Os *widgets* são divididos em dois grupos:
  - As classes *QListWidget*, *QTableWidget* e *QTreeWidget* devem ser utilizadas para apresentar poucos itens;
  - Para grandes conjuntos devem ser utilizadas as classes *QListView*, *QTableView* e *QTreeView*, em conjunto com um modelo de dados (do Qt ou customizado).

Model-View

- Modelos predefinidos do Qt:
  - *QStringListModel*: armazena uma lista de *strings*;
  - *QStandardItemModel*: armazena dados hierárquicos arbitrários;
  - *QDirModel*: encapsula o sistema de arquivos local;
  - *QSqlQueryModel*: encapsula um *resultset* SQL;
  - *QSqlTableModel*: encapsula uma tabela SQL;
  - *QSqlRelationalTableModel*: encapsula uma tabela SQL com chaves estrangeiras;
  - *QSortFilterProxyModel*: ordena ou filtra outro modelo.

Model-View

- Visualizando o sistema de arquivos:

```
model = new QDirModel;
model->setReadOnly(false);
model->setSorting(QDir::DirsFirst | QDir::IgnoreCase | QDir::Name);
treeView = new QTreeView;                    treeView->setModel(model);
treeView->header()->setStretchLastSection(true);
treeView->header()->setSortIndicator(0, Qt::AscendingOrder);
treeView->header()->setSortIndicatorShown(true);
treeView->header()->setClickable(true);
QModelIndex index = model->index(QDir::currentPath());
treeView->expand(index);
treeView->scrollTo(index);
treeView->resizeColumnToContents(0);
```

lab6 / dirtableview.cpp

Classes Container

- São implicitamente compartilhados;
- Possuem *iterators*;
- Podem ser serializados com o *QDataStream*;
- Resultam em menos código executável que os correspondentes STL;
  - *Containers* sequenciais. Ex: *QVector*, *QLinkedList*, *QList*.
  - *Containers* associativos. Ex: *QMap*, *QHash*.
  - Algoritmos genéricos. Ex: *qSort()*, *qBinaryFind()*.

Banco de Dados

- O módulo *QtSql* disponibiliza uma interface independente de plataforma e de banco de dados;
- Essa interface é suportada por um conjunto de classes que usam a arquitetura *model/view* do Qt4;
- Uma conexão de banco é representada pelo objeto *QSqlDatabase*.

Banco de Dados

- *Drivers* disponíveis:
  - QDB2: IBM DB2 (version 7.1 and above)
  - QIBASE: Borland InterBase
  - QMYSQL: MySQL
  - QOCI: Oracle Call Interface Driver
  - QODBC: Open Database Connectivity (ODBC)
  - QPSQL: PostgreSQL (versions 7.3 and above)
  - QSQLITE2: SQLite version 2
  - QSQLITE: SQLite version 3
  - QTDS: Sybase Adaptive Server

Banco de Dados

- O banco pode ser utilizado de duas formas:
  - Com a classe *QSqlQuery*: possibilita a execução direta de sentenças SQL;
  - Com classes de mais alto nível como *QSqlTableModel* e *QsqlRelationalTableModel*;
  - As classes de mais alto nível podem ser anexadas a *widgets* para visualização automática dos dados do banco;
  - O Qt4 torna fácil a programação de recursos como *masterdetail* e *drill-down*.

Banco de Dados

- Criando a conexão *default*:

```
inline bool createConnection()
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("cddatabase.db");
    if (!db.open()) {
        QMessageBox::warning(0, QObject::tr("Database Error"),
                             db.lastError().text());
    return false;
    }
    return true;
}
```

Banco de Dados

- Criando a conexão *default*:

Cria uma nova conexão como conexão *default*

```
inline bool createConnection()
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("cddatabase.db");
    if (!db.open()) {
        QMessageBox::warning(0, QObject::tr("Database Error"),
                             db.lastError().text());
    return false;
    }
    return true;
}
```


Banco de Dados

- Criando a conexão *default*:

```
inline bool createConnection()
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("cddatabase.db");
    if (!db.open()) {
        QMessageBox::warning(0, QObject::tr("Database Error"),
                             db.lastError().text());
    return false;
    }
    return true;
}
```

Cria uma nova conexão como conexão *default*

Informa qual o banco a ser aberto

Banco de Dados

- Criando a conexão *default*:

```
inline bool createConnection()
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("cddatabase.db");
    if (!db.open()) {
        QMessageBox::warning(0, QObject::tr("Database Error"),
                             db.lastError().text());
    return false;
    }
    return true;
}
```

Cria uma nova conexão como conexão *default*

Abre o banco de dados

Informa qual o banco a ser aberto

Banco de Dados

- Consultando com *QSqlQuery*:

```
QSqlQuery query;
query.exec("SELECT title, year FROM cd WHERE year >= 1998");
while (query.next()) {
      QString title = query.value(0).toString();
      int year = query.value(1).toInt();
      cerr << qPrintable(title) << ": " << year << endl;
}
```

  - Outros métodos (geralmente custosos) da classe *QSqlQuery*: *first()*, *last()*, *previous()*, *seek()*;
  - Pode otimizar invocando *QSqlQuery::setForwardOnly(true)* antes do *exec()* e então utiliza-se somente o *next()*.

Banco de Dados

- *QSqlQuery* podeser utilizado com comandos DDL e *inserts*:

```
QSqlQuery query;

query.exec("CREATE TABLE artist (id INTEGER PRIMARY KEY, "
           "name VARCHAR(40) NOT NULL, country VARCHAR(40))");

query.exec("CREATE TABLE cd (id INTEGER PRIMARY KEY, "
           "title VARCHAR(40) NOT NULL, artistid INTEGER NOT NULL, "
           "year INTEGER NOT NULL, "
           "FOREIGN KEY (artistid) REFERENCES artist)");

query.exec("CREATE TABLE track (id INTEGER PRIMARY KEY, "
           "title VARCHAR(40) NOT NULL, duration INTEGER NOT NULL, "
           "cdid INTEGER NOT NULL)");
```

Banco de Dados

- Consultando com *QSqlTableModel*:

```
QSqlTableModel model;
model.setTable("cd");
model.setFilter("year >= 1998");
model.select();
connect(model, SIGNAL(beforeInsert(QSqlRecord &)),
            this, SLOT(beforeInsertArtist(QSqlRecord &)));
for (int i = 0; i < model.rowCount(); ++i) {
      QSqlRecord record = model.record(i);
      QString title = record.value("title").toString();
      int year = record.value("year").toInt();
      cerr << qPrintable(title) << ": " << year << endl;
}
```

Banco de Dados

- Ligando o *model* a um *view*:

```
model = new QSqlTableModel(this);
model->setTable("artist");
model->setSort(Artist_Name, Qt::AscendingOrder);
model->setHeaderData(Artist_Id, Qt::Horizontal, tr("Id"));
model->setHeaderData(Artist_Name, Qt::Horizontal, tr("Name"));
model->setHeaderData(Artist_Country, Qt::Horizontal, tr("Country"));
model->select();
artistTableView = new QTableView;
artistTableView->setModel(model);
artistTableView->setSelectionBehavior(QAbstractItemView::SelectRows);
artistTableView->resizeColumnsToContents();
```

Banco de Dados

- Incluindo registros:

```
...
connect(model, SIGNAL(beforeInsert(QSqlRecord &)),
        this, SLOT(beforeInsertArtist(QSqlRecord &)));
...

int row = model->rowCount();
model->insertRow(row);
QModelIndex index = model->index(row, Artist_Name);
tableView->setCurrentIndex(index);
tableView->edit(index);

void ArtistForm::beforeInsertArtist(QSqlRecord &record)
{
    record.setValue("id", generateId("artist"));
}
```

Banco de Dados

- Incluindo registros:

```
inline int generateId(const QString &table)
{
	QSqlQuery query;
	query.exec("SELECT MAX(id) FROM " + table);
	int id = 0;
	if (query.next())
		id = query.value(0).toInt() + 1;
	return id;
}
```

Banco de Dados

- Excluindo registros:

```
tableView->setFocus();
QModelIndex index = tableView->currentIndex();
if (!index.isValid())
        return;
QSqlRecord record = model->record(index.row());
QSqlTableModel cdModel;
cdModel.setTable("cd");
cdModel.setFilter("artistid = " + record.value("id").toString());
cdModel.select();
if (cdModel.rowCount() == 0) {
        model->removeRow(tableView->currentIndex().row());
} else {
        QMessageBox::information(this,
                        tr("Delete Artist"),
                        tr("Cannot delete %1 because there are CDs associated "
                        "with this artist in the collection.")
                        .arg(record.value("name").toString()));
}
```


Banco de Dados

- Relacionamentos e *master-detail*:
  - *QSqlRelationalTableModel* para exibir CDs com o nome do autor no lugar do *id*;

```
cdModel = new QSqlRelationalTableModel(this);
cdModel->setTable("cd");
cdModel->setRelation(Cd_ArtistId,
                     QSqlRelation("artist", "id", "name"));
```

Banco de Dados

- Relacionamentos e *master-detail*:
  - Para o *detail* usa-se o *QSqlTableModel* original com um filtro;
  - Capturar o *signal currentRowChanged* do modelo *master* para atualizar o *detail*.

```
connect(cdTableView->selectionModel(),
        SIGNAL(currentRowChanged(const QModelIndex &,
                                 const QModelIndex &)),
        this, SLOT(currentCdChanged(const QModelIndex &)));
```

Painting

- A principal classe para desenho 2D é *QPainter*:
  - Desenha formas geométricas (pontos, retas, retângulos, elipses, arcos, polígonos e curvas de Bézier);
  - Desenha *pixmaps*, imagens e texto;
  - Possui mecanismos para *anti-aliasing*, canal alfa, gradientes e vetores;
  - Suporta transformações geométricas (translações, rotações e escala (*zoom*)).

Painting

- A principal classe para desenho 2D é *QPainter*:
  - O *QPainter* desenha em um "*paint device*" como um *QWidget*, *QPixmap* ou *Qimage*;
  - Pode ser utilizado em conjunto com *QPrinter* para impressão e geração de arquivos PDF. Ou seja, um mesmo código imprime na tela ou na impressora;
  - Uma alternativa ao *QPainter* é o uso das classes do módulo de *OpenGL* do Qt4.

Painting

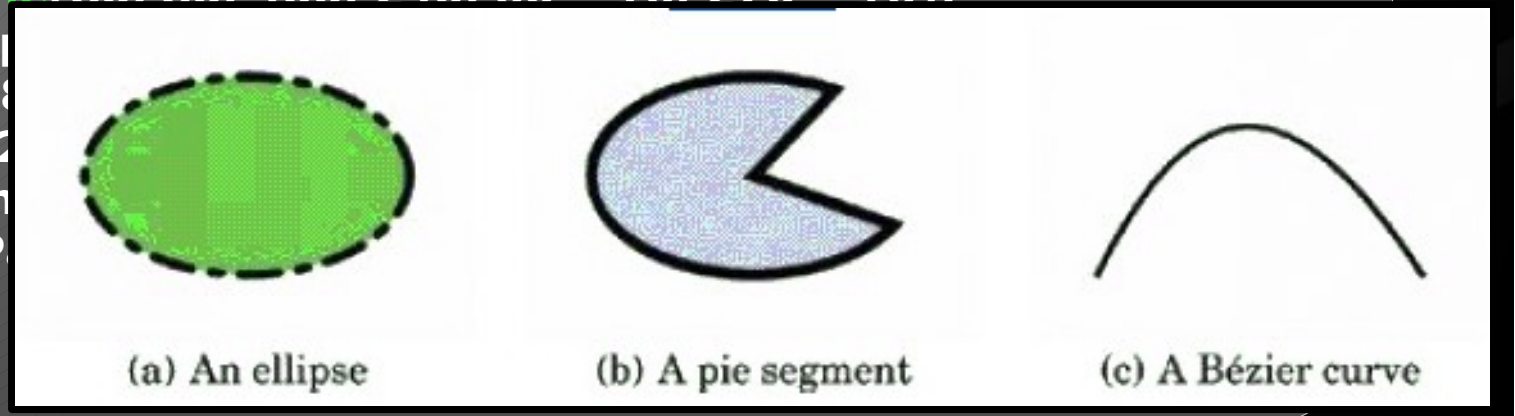
- Exemplo de uso do *QPainter*:

```
QPainter painter(this);
painter.setRenderHint(QPainter::Antialiasing, true);
painter.setPen(QPen(Qt::black, 12, Qt::DashDotLine, Qt::RoundCap));
painter.setBrush(QBrush(Qt::green, Qt::SolidPattern));
painter.drawEllipse(80, 80, 400, 240);
painter.drawPie(80, 80, 400, 240, 60 * 16, 270 * 16);
QPainterPath path;
path.moveTo(80, 320);
path.cubicTo(200, 80, 320, 80, 480, 320);
painter.setPen(QPen(Qt::black, 8));
painter.drawPath(path);
```

Painting

- Exemplo de uso do *QPainter*:

```
QPainter painter(this);
painter.setRenderHint(QPainter::Antialiasing, true);
painter.setPen(QPen(Qt::black, 12, Qt::DashDotLine, Qt::RoundCap));
painter.setBrush(QBrush(Qt::green, Qt::SolidPattern));
painter.drawEllipse(80, 80, 400, 240);
painter.drawPie(80, 80, 400, 240, 60 * 16, 270 * 16);
QPainterPath
path.moveTo(
path.cubicTo(
painter.setPen
painter.drawP
```




XML

- O Módulo *QtXml* disponibiliza duas interfaces para manipulação de XML:
  - SAX (*Simple API for XML*): reporta eventos de *parsing* diretamente para a aplicação;
  - DOM (*Document Object Model*): converte um documento XML em uma árvore de objetos, na qual a aplicação pode navegar.

XML

- Lendo XML com o SAX:
  - O Qt disponibiliza a classe *QXmlSimpleReader* para a leitura não validada de documentos XML;
  - Funções virtuais de objetos *handler* registrados são invocadas para sinalizar os eventos;
  - Ex:

```
<doc>
    <quote>Ars longa vita brevis</quote>
</doc>
```

```
startDocument()
startElement("doc")
startElement("quote")
characters("Ars longa vita brevis")
endElement("quote")
endElement("doc")
endDocument()
```

XML

- Exemplo de classe *handle* para o SAX:

```
class SaxHandler : public QXmlDefaultHandler
{
public:
SaxHandler(QTreeWidget *tree);
bool startElement(const QString &namespaceURI, const QString &localName,
                  const QString &qName, const QXmlAttributes &attributes);
bool endElement(const QString &namespaceURI, const QString &localName,
                  const QString &qName);
bool characters(const QString &str);
bool fatalError(const QXmlParseException &exception);
private:
QTreeWidget *treeWidget;
};
```

XML

- Processando o XML:

```
bool parseFile(const QString &fileName)
{
    QFile file(fileName);
    QXmlInputSource inputSource(&file);
    QXmlSimpleReader reader;
    SaxHandler handler(treeWidget);
    reader.setContentHandler(&handler);
    reader.setErrorHandler(&handler);
    return reader.parse(inputSource);
}
```


XML

- A partir da versão 4.3, as classes *QXmlStreamReader* e *QXmlStreamWriter* simplificam este processo;
- O módulo *QtXmlPatterns* inclui o uso de *XQuery* e *XPath* para realizar buscas facilitadas em arquivos XML.

Networking

- O Qt disponibiliza as classes *QFtp* e *QHttp*, que podem ser utilizadas para fazer *download* e *upload* de arquivos, bem como solicitar e receber páginas de servidores web;
- O Qt também provê classes de nível mais baixo, como *QTcpSocket* e *QUdpSocket*;
- Para tratar conexões em servidores pode-se utilizar a classe *QtcpServer*;
- Disponibiliza recursos para *sockets* seguros, certificados, chaves etc.

Internacionalização

- O Qt suporta *unicode*, linguas não derivadas do Latin e linguas escritas da direita para a esquerda;
- O Qt torna fácil traduzir todo um sistema de uma lingua para outra:
  - Envolva toda string visível pelo usuário na macro *tr()*;
  - Execute o comando *lupdate* para gerar o arquivo a ser traduzido;
  - Utilize o *Qt Linguist* para traduzir o arquivo;
  - Execute o comando *lrelease* para gerar o arquivo traduzido.

Internacionalização

- Grande parte das aplicações carrega o arquivo de traduções no momento em que é iniciada, baseado nas configurações de *locale* do sistema;
- O Qt também possibilita a troca, em tempo de execução, da lingua utilizada pelo sistema;
- O Qt também permite a seleção de imagens (ícones, *splashs* etc) a depender do *locale* do sistema.

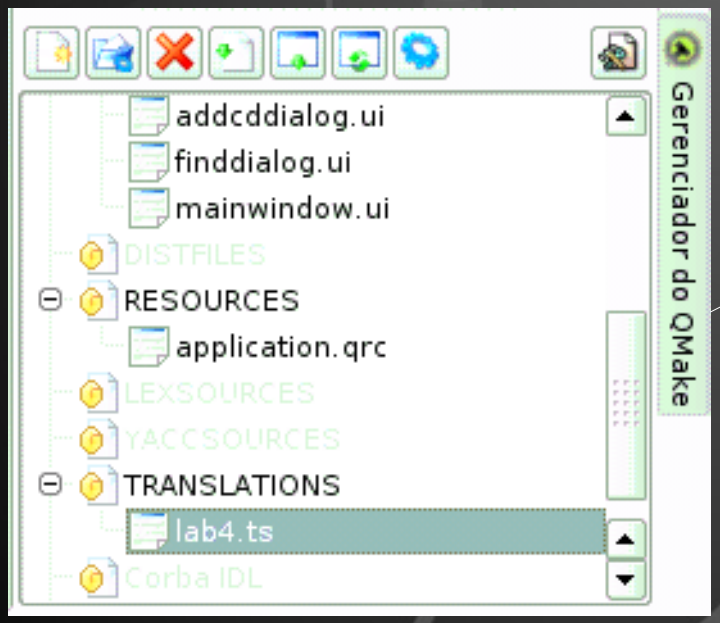
Internacionalização

- Carregando a tradução ao iniciar a aplicação:

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QTranslator appTranslator;
    appTranslator.load("myapp_" + QLocale::system().name(),
                       qApp->applicationDirPath());
    app.installTranslator(&appTranslator);
    ...
    return app.exec();
}
```

Internacionalização

- Criando e traduzindo o arquivo:



Qt Linguist

lrelease as
(no diretório do executável)

Plugins

- O que é um *Plugin* ?
  - “É uma biblioteca dinâmica que implementa uma interface particular, com o objetivo de prover funcionalidades extras.”; **(C++ GUI Programming with Qt4)**
  - Não requer que seja previamente compilada e linkada à aplicação.
- O Qt4 possui um conjunto próprio de interfaces de *plugins* para domínios tais como:
  - Formato de imagens, *drivers* de banco de dados, estilos de *widgets*, codificação de texto etc.

Plugins

- É possível criar novos domínios de *plugins*, específicos para uma aplicação em particular;
- O *framework* para *plugins* do Qt adiciona recursos de conveniência e segurança à classe *Qlibrary*;
- Para criar um *plugin* para os domínios já existentes do Qt4 precisa-se implementar duas classes:
  - *Plugin Wrapper*
  - *Plugin Handler*

Plugins

- É possível criar novos domínios de *plugins*, específicos para uma aplicação em particular;
- O *framework* para *plugins* do Qt adiciona recursos de conveniência e segurança à classe *Qlibrary*;
- Para criar um *plugin* para os domínios já existentes do Qt4 precisa-se implementar duas classes:
  - *Plugin Wrapper*
  - *Plugin Handler*

Classe *factory* para criação de novos *handlers*

Plugins

- É possível criar novos domínios de *plugins*, específicos para uma aplicação em particular;
- O *framework* para *plugins* do Qt adiciona recursos de conveniência e segurança à classe *Qlibrary*;
- Para criar um *plugin* para os domínios já existentes do Qt4 precisa-se implementar duas classes:
  - *Plugin Wrapper*
  - *Plugin Handler*

Classe *factory* para criação de novos *handlers*

Classe que efetivamente implementa a funcionalidade

Plugins

- Estas duas classes deverão ser filhas de classes disponibilizadas pelo Qt:

Wrapper	Handler
QAccessibleBridgePlugin	QAccessibleBridge
QAccessiblePlugin	QAccessibleInterface
QIconEnginePlugin	QIconEngine
QImageIOPlugin	QImageIOHandler
QInputContextPlugin	QInputContext
QPictureFormatPlugin	-
QSqlDriverPlugin	QSqlDriver
QStylePlugin	QStyle
QTextCodecPlugin	QTextCodec

Plugins

- Os *plugins* para os domínios do Qt são invocados pelos próprios objetos do Qt (ex: *QImageReader*, *QImageWriter*);
- *Plugins* instalados globalmente devem estar localizados no diretório $QTDIR/plugins/;
- *Plugins* instalados localmente devem estar no subdiretório *plugins/* do diretório que contém o executável da aplicação.

Plugins

- A arquitetura de *plugins* do Qt é uma abordagem interessante para a extensão facilitada de aplicações em C++;
- As classes *QLibrary* e *QPluginLoader* do Qt realizam o processo de carga dinâmica de novas classes;
- Se as interfaces dos *plugins* forem bem projetadas, pode-se estender aplicações sem requerer novas recompilações;
- Alternativas: *QtScript*, *Kross* ...

O que vem por aí ?

- *ItemView: the next generation*

O que vem por aí (Qt 4.5) ?

- *QXmlToQQbjectCreator*;
- Melhorias no Qt Designer;
- Melhoria de até 70% nas operações de *painting*;
- *FileDialog* nativo no Gnome;
- *Widgets* com *Alpha;*
- Uso do Cocoa no OSX.

Introdução

- Módulos do KDE4:
  - *Qt*
  - *Phonon*
  - *Decibel*
  - *Telepathy*
  - *Plasma*
  - *KIO*

Phonon

- Phonon é o nome de particulas que descrevem um modelo quantizado de vibração em um sólido;

- No KDE 4 é a biblioteca que possibilita a vibração em um sólido.

Plasma

- Plasma é a parte liquida do sangue, e permite com que ele flua pelo corpo;
- Também é o nome que se dá ao quarto estado da matéria;
- Ou ao *Desktop Shell* do KDE.

Requisitos de Software

- CMake 2.6;
- Qt 4.4.x;
- kdelibs-dev 4.1.x;
- Sua distribuição instalará as outras dependências automaticamente.

Hello World

- Bibliotecas utilizadas:
  - *kdecore* e *kdeui*
- Arquivos-cabeçalho:
  - *Kapplication*
  - *KcmdLineArgs*
  - *KaboutData*
  - *KMessageBox*

Hello World

- *KApplication*:
  - Classe necessária em todos os programas;
  - Provê controle de eventos, atalhos, menus, ajuda e outras coisas mais para o aplicativo.

Hello World

- *KAboutData*:
  - Classe usada para guardar informações sobre o programa;
  - Todas as informações guardadas dentro da instância da classe ficará à mostra na opção do menu 'sobre'.

Hello World

- *KCmdLineArgs*:
  - Classe para manipulação de linhas de comando;
  - Provê acesso aos argumentos do programa, levando em consideração as opções especificas do Qt e KDE, e as do próprio aplicativo.

Hello World

- *KMessageBox*:
  - Classe que provê acesso à caixas de diálogo de informações. Vários métodos estáticos existem para a invocação de caixas de diálogo mais comuns.

Hello World

- Iniciando o código:

```
#include <KApplication>
#include <KAboutData>
#include <KCmdLineArgs>
#include <KMessageBox>
```


Hello World

- *KAboutData*:

```
KAboutData sobre("NomeDoPrograma",
                 0,
                  ki18n("Nome do programa"),
                 "0.1",
                  ki18n("Descrição do programa"),
                  KaboutData::License_GPL,
                  Ki18n(" © 2008 "),
                  Ki18n(" mais informações "),
                 "http://www.suapagina.org",
                 "seuemail@mail.com");
```

Hello World

- Corpo do *main*:

```
KcmdLineArgs::init(argc, argv, &sobre);
KApplication app;

KGuiItem botaoSim(i18n("Oie :)"), QString(),
        i18n("isso é um tooltip"),
        i18n("isso é um texto de ajuda"));

KMessageBox::questionYesNo(0, i18n( "olá mundo" ),i18n( "Oi." ), botaoSim);
```


Hello World

```
#include <KApplication>
#include <KAboutData>
#include <KCmdLineArgs>
#include <KmessageBox>

int main(int argc, char *argv[]){
      KAboutData sobre("NomeDoPrograma",                0,
                       ki18n("Nome do programa"),  "0.1",
                       ki18n("Descrição"),              KaboutData::License_GPL,
                       Ki18n(" © 2008 "),                Ki18n(" mais informações "),
                       "http://www.suapagina.org",  "seuemail@mail.com");

      KCmdLineArgs::init( argc, argv, &aboutData );
      KApplication app;

      KGuiItem botaoSim( i18n( "Oie :)" ), Qstring(),  i18n( "isso é um tooltip" ),
        i18n( "isso é um texto de ajuda" ) );
      KMessageBox::questionYesNo( 0, i18n( "olá mundo" ),i18n( "Oi." ), botaoSim );
}
```

Hello World

- Compilando:
  - *Autotools* ?
  - Arquivos de projeto ?
  - *./configure* ?
  - *CMake to the rescue*.

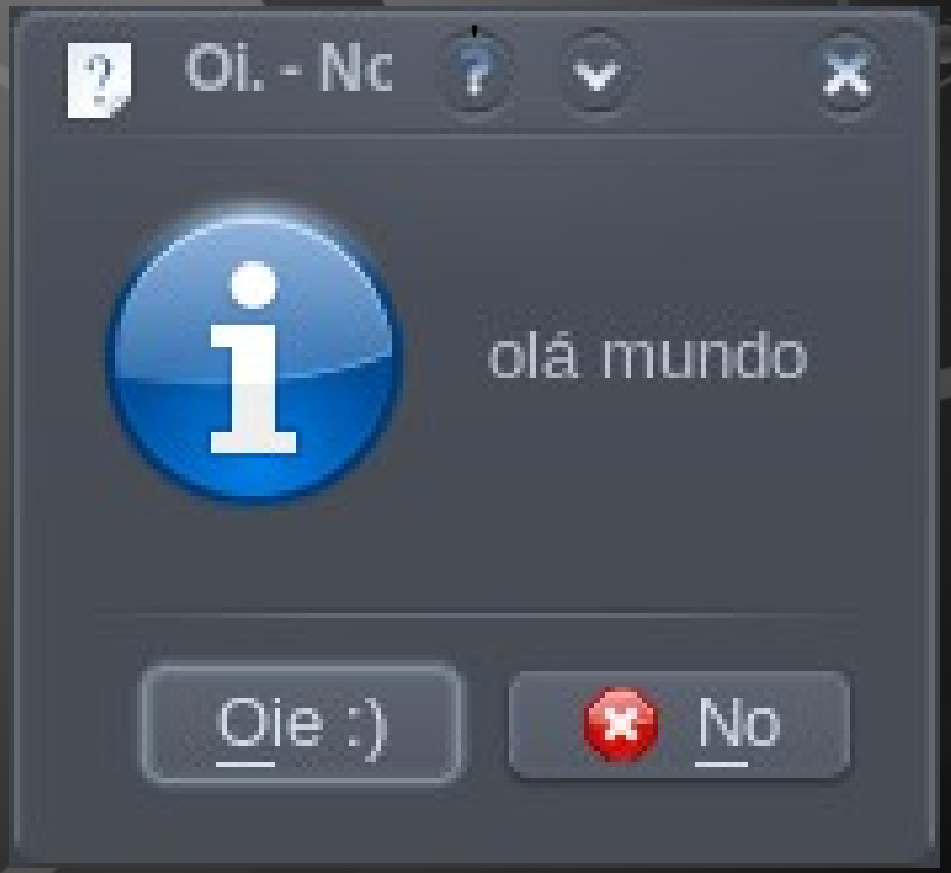
Hello World

- *CMakeLists.txt*:

```
project (tutorial1)

find_package(KDE4 REQUIRED)
include_directories(${KDE4_INCLUDES})

set(tutorial1_SRCS main.cpp)

kde4_add_executable(tutorial1 ${tutorial1_SRCS})
target_link_libraries(tutorial1 ${KDE4_KDEUI_LIBS})
```

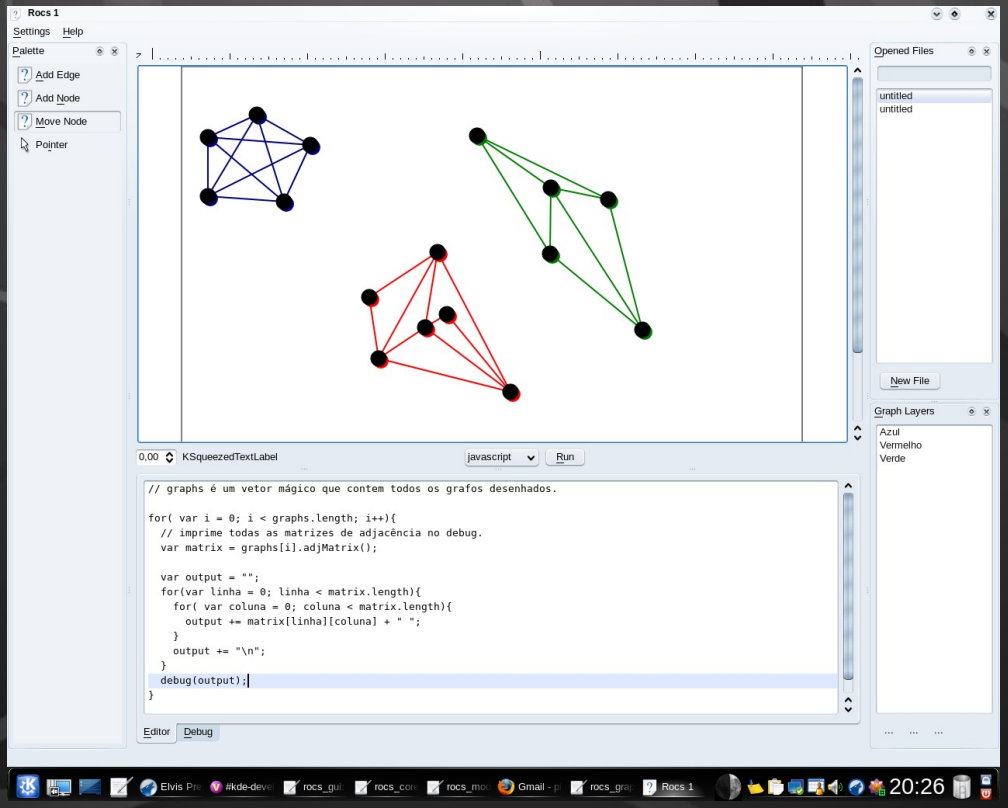
Hello World

- Compilando:
  - *cmake .*
  - *make*
- Executando:
  - *./tutorial1*



Introdução ao KDEUI

- Biblioteca com elementos padrões para *interface* de usuário.



Introdução ao KDEUI

- *KXmlGuiWindow*:
  - Ao invés de criar um *KMainWindow* ou um *QMainWindow* no Qt, Crie um *KXmlGuiWindow*;
  - Esqueça de metade de seus problemas com *layouts* de menus e *toolbars*.

Introdução ao KDEUI

- *KTextEdit*:
  - Classe de manipulação de entrada de texto do KDE. Estende a *QTextEdit* e implementa coisas padrões do KDE como *spell-checking*.

Introdução ao KDEUI

- Utilizando o *KXmlGuiWindow*:
  - Bibliotecas utilizadas:
    - *kdecore* e *kdeui*
  - Arquivos-cabeçalho:
    - *KApplication* - *KCmdLineArgs* - *KTextEdit*
    - *KAboutData* - *KXmlGuiWindow*
  - Arquivos do Projeto:
    - *mainwindow.h*
    - *mainwindow.cpp*
    - *main.cpp*

Introdução ao KDEUI

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <KXmlGuiWindow>
#include <KTextEdit>

class MainWindow : public KXmlGuiWindow{
public:
    MainWindow(QWidget *parent=0);
private:
    KTextEdit* textArea;
};

#endif
```

Introdução ao KDEUI

```
#include "mainwindow.h"

mainwindow::mainwindow() {
    textArea = new KTextArea();
    setCentralWidget(textArea);
    setupGUI(this);
}
```

mainwindow.cpp

Introdução ao KDEUI

```
#include <Kapplication>
#include <KcmdLineArgs>
#include <KaboutData>
#include "mainwindow.h"

int main(int argc, char *argv){
      KAboutData aboutData( "Aplicativo2", 0,
                                 ki18n("Aplicativo 2"), "1.0",
                                 ki18n("uma area de texto simples"),
                                 KAboutData::License_GPL,
                                 ki18n("Copyright (c) 2008") );

      KCmdLineArgs::init( argc, argv, &aboutData );
      KApplication app;

      MainWindow* window = new MainWindow();
      window->show();
      return app.exec();
}
```

main.cpp

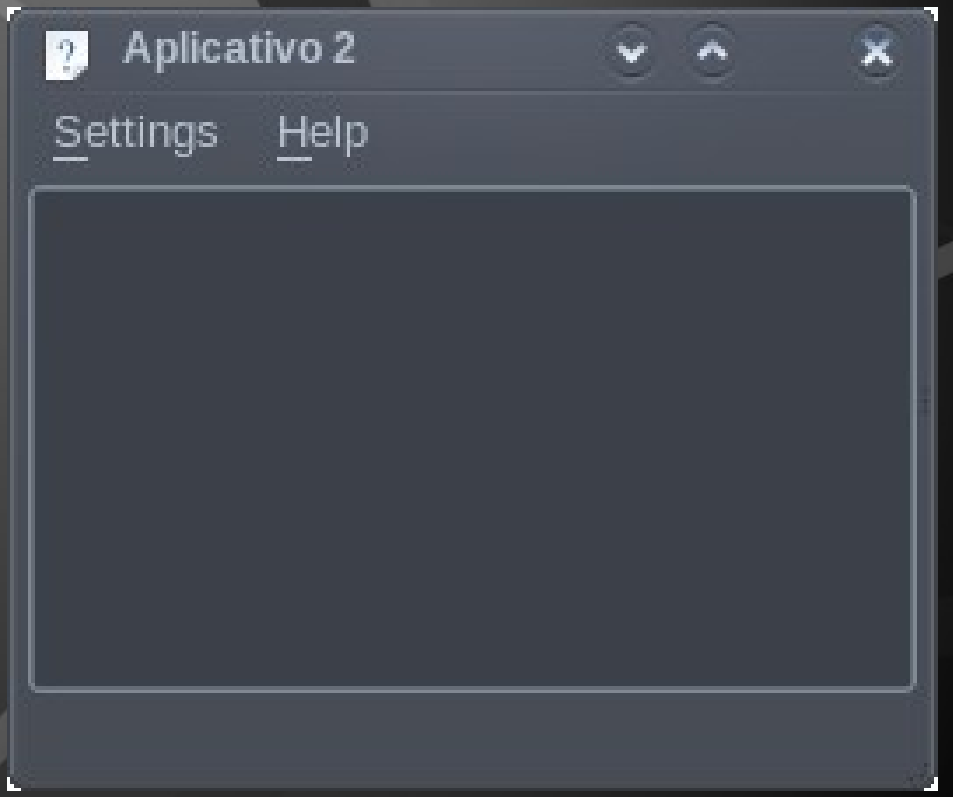
Introdução ao KDEUI

```
project (tutorial2)

find_package(KDE4 REQUIRED)
include_directories(${KDE4_INCLUDES})

set(tutorial2_SRCS
main.cpp
mainwindow.cpp)

kde4_add_executable(tutorial2 ${tutorial2_SRCS})
target_link_libraries(tutorial2 ${KDE4_KDEUI_LIBS})
```

CMakeLists.txt

Introdução ao KDEUI

- Compilando:
  - *cmake .*
  - *make*
  - *./tutorial2*



Introdução ao KDEUI

- Erro ?

```
[tomaz@localhost Aplicativo 2]$ ./tutorial2
QFSFileEngine::open: No file name specified
Aplicativo2(8292)/kdeui (kdelibs):
No such XML file  "Aplicativo2ui.rc"
```

Introdução ao KDEUI

- Arquivos rc:
  - Definição de uma *toolbar* e um *menubar* utilizando notação XML;
  - nomeui.rc, onde nome é o nome do programa definido no *KAboutData*.

Introdução ao KDEUI

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE kpartgui SYSTEM "kpartgui.dtd">
<gui name="tutorial2" version="1">
 <ToolBar name="mainToolBar" >
  <text>Main Toolbar</text>
  </ToolBar>
 <MenuBar>
  <Menu name="file" >
  </Menu>
 </MenuBar>
</gui>
```

- Criando um arquivo rc:
  - Facilidade?
  - Legibilidade?
  - Manutenção?

Introdução ao KDEUI

- Ainda sobre o rc:
  - ToolBar:
    - Define uma barra de ferramentas. Caso o nome seja mainToolBar, ela será mostrada junto com o programa principal logo abaixo do Menu. Pode ser desabilitado em tempo de execução.
  - MenuBar:
    - Define o Menu principal do programa. Não pode ser desabilitado em tempo de execução.

Introdução ao KDEUI

- Instalando o arquivo rc:
  - Arquivos rc não são compilados junto com o programa;
  - São carregados e interpretados em tempo de execução;
  - Precisam estar em uma pasta acessível;
  - Logo, é preciso instalar o arquivo.

Introdução ao KDEUI

```
project (tutorial2)

find_package(KDE4 REQUIRED)
include_directories(${KDE4_INCLUDES})
set(tutorial2_SRCS main.cpp mainwindow.cpp)
kde4_add_executable(tutorial2 ${tutorial2_SRCS})
target_link_libraries(tutorial2 ${KDE4_KDEUI_LIBS})

install(TARGETS tutorial2 DESTINATION ${BIN_INSTALL_DIR})

install(FILES tutorial2ui.rc
DESTINATION ${DATA_INSTALL_DIR}/tutorial2)
```

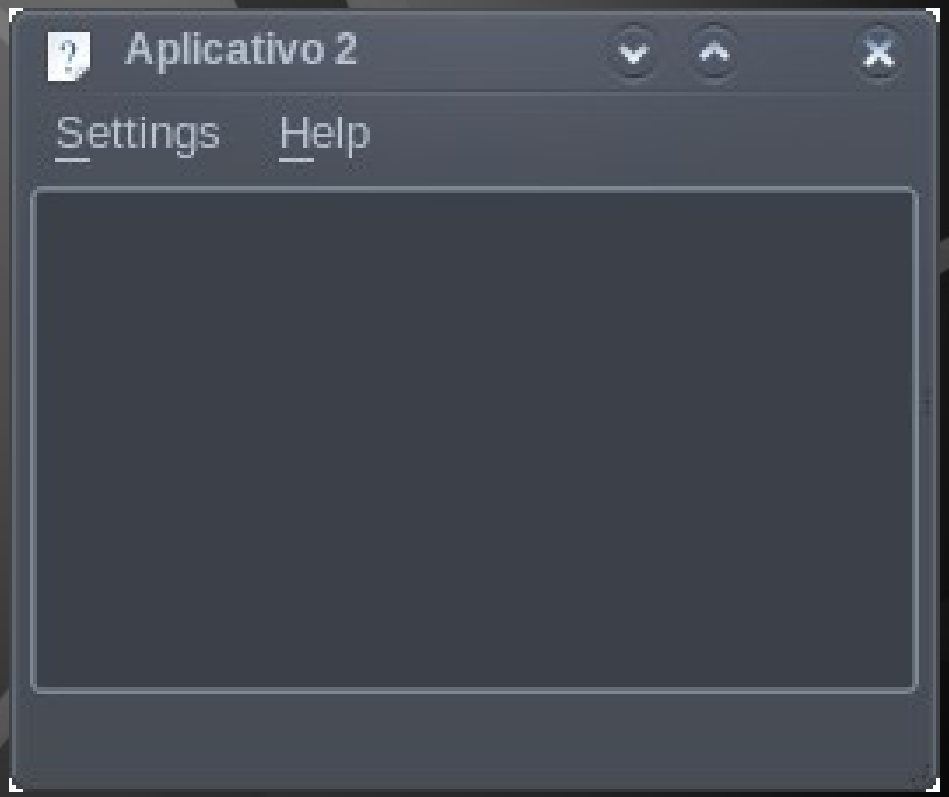
CMakeLists.txt

Introdução ao KDEUI

- Instalando:
  - A depender de como seu comando *cmake* foi executado, essa situação varia.

Introdução ao KDEUI

- Compilando e instalando:
  - *cmake .*
  - *make*
  - *sudo make install*
  - *tutorial2*



Introdução ao KDEUI

- Mas ... igual ?
  - Quase.
  - Diferenças Conceituais:
    - Menu dinâmico;
    - *ToolBox* dinâmico.

Introdução ao KDEUI

- *KActionCollection*:
  - Classe que agrupa coleções de *KActions* e *QActions*, geralmente as que aparecem visíveis ao usuario nas *toolbars*.

Introdução ao KDEUI

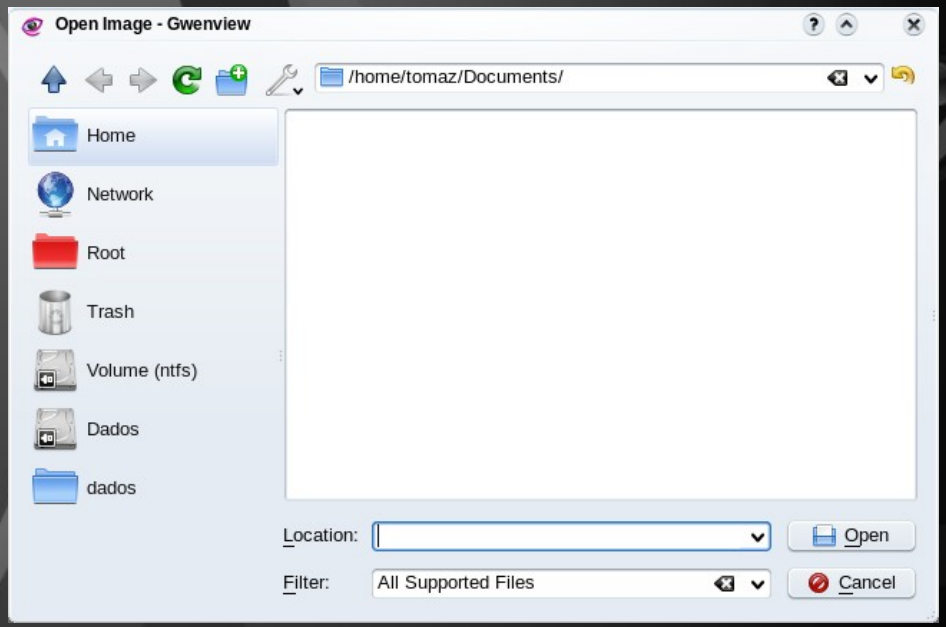
- *KIO/NetAccess*:
  - Classe que permite a navegação por redes como se estivessem no seu diretório físico de forma transparente.

Introdução ao KDEUI

- *KSaveFile*:
  - Classe de ajuda ao salvar *streams* em arquivos;
  - Salva apenas após todas as operações terem sido completadas.

Introdução ao KDEUI

- *KFileDialog*:
  - Uma caixa de dialogo pronta para ser usada, similar ao *QFileDialog*, mas com funcionalidade expandida para funcionar com o KIO.



Introdução ao KDEUI

- Adicionando funcionalidades:
  - Bibliotecas utilizadas:
    - *kdecore*, *kdeui* e *KIO*
  - Arquivos do Projeto
    - *mainwindow.h*
    - *mainwindow.cpp*
    - *main.cpp*

Introdução ao KDEUI

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <KXmlGuiWindow>
#include <KTextEdit>

class MainWindow : public
KXmlGuiWindow{
Q_OBJECT

public:
  MainWindow(QWidget
*parent=0);

private:
  KTextEdit* textArea;
  void setupActions();
  QString fileName;

private slots:
  void newFile();
  void openFile();
  void saveFile();
  void saveFileAs();
  void saveFileAs(const QString

&outputFileName);
};

#endif
```

mainwindow.h

Introdução ao KDEUI

```
#include "mainwindow.h"
#include <KApplication>

#include <KAction>
#include <KLocale>
#include <KActionCollection>
#include <KStandardAction>
#include <KFileDialog>
#include <KMessageBox>
#include <KIO/NetAccess>
#include <KSaveFile>
#include <QTextStream>
```

```
MainWindow::MainWindow(QWidget *parent)
: KXmlGuiWindow(parent), fileName(QString())
{
  textArea = new KTextEdit;
  setCentralWidget(textArea);
  setupActions();
  setupGUI();
}
```

mainwindow.cpp

Introdução ao KDEUI

- *KActions*:
  - Semelhante aos *QActions*, com suporte a Imagens e atalhos de forma simples.
- *KStandardActions*:
  - *KActions* pré prontas para seu uso, em todos os sabores e formas.

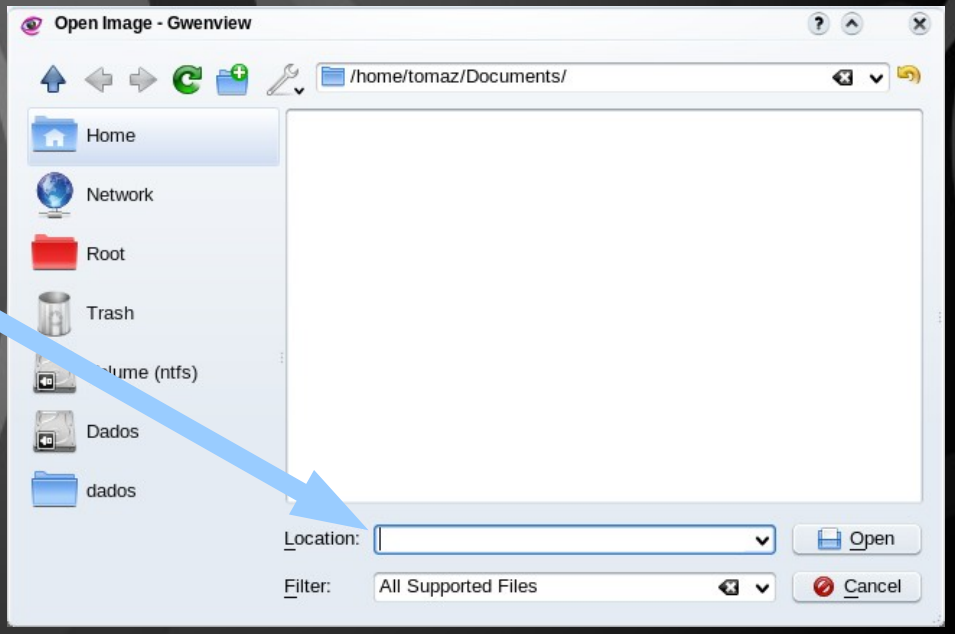
Introdução ao KDEUI

```
void MainWindow::setupActions(){
  KAction* clearAction = new KAction(this);
  clearAction->setText(i18n("Clear"));
  clearAction->setIcon(KIcon("document-new"));
  clearAction->setShortcut(Qt::CTRL + Qt::Key_W);
  actionCollection()->addAction("clear", clearAction);
  connect(clearAction, SIGNAL(triggered(bool)),
  textArea, SLOT(clear()));

  KStandardAction::quit(kapp, SLOT(quit()),     actionCollection());
  KStandardAction::open(this, SLOT(openFile()), actionCollection());
  KStandardAction::save(this, SLOT(saveFile()), actionCollection());
  KStandardAction::saveAs(this, SLOT(saveFileAs()), actionCollection());
  KStandardAction::openNew(this, SLOT(newFile()), actionCollection());
}
```

mainwindow.cpp

Introdução ao KDEUI

```
void MainWindow::saveFileAs(){
      saveFileAs(KFileDialog
            ::getSaveFileName());
}

void MainWindow::saveFile(){
      if (fileName.isEmpty()){
            SaveFileAs();
      }
      else {
            saveFileAs(fileName);
      }
}
```



Introdução ao KDEUI

```
void MainWindow::newFile(){
	fileName.clear();
	textArea->clear();
}

void MainWindow::saveFileAs(const QString &outputFileName){
	KSaveFile file(outputFileName);
	file.open();
	QByteArray outputByteArray;
	outputByteArray.append(textArea->toPlainText().toUtf8());
	file.write(outputByteArray);
	file.finalize();
	file.close();
	fileName = outputFileName;
}
```

mainwindow.cpp

Introdução ao KDEUI

```
void MainWindow::openFile(){
    QString fileNameFromDialog = KFileDialog::getOpenFileName();
    QString tmpFile;
    if(!KIO::NetAccess::download(fileNameFromDialog, tmpFile, this)){
        QFile file(tmpFile);
        file.open(QIODevice::ReadOnly);
        textArea->setPlainText(QTextStream(&file).readAll());
        fileName = fileNameFromDialog;
        KIO::NetAccess::removeTempFile(tmpFile);
    }
     else {
        KMessageBox::error(this, KIO::NetAccess::lastErrorString());
    }
}
```

mainwindow.cpp

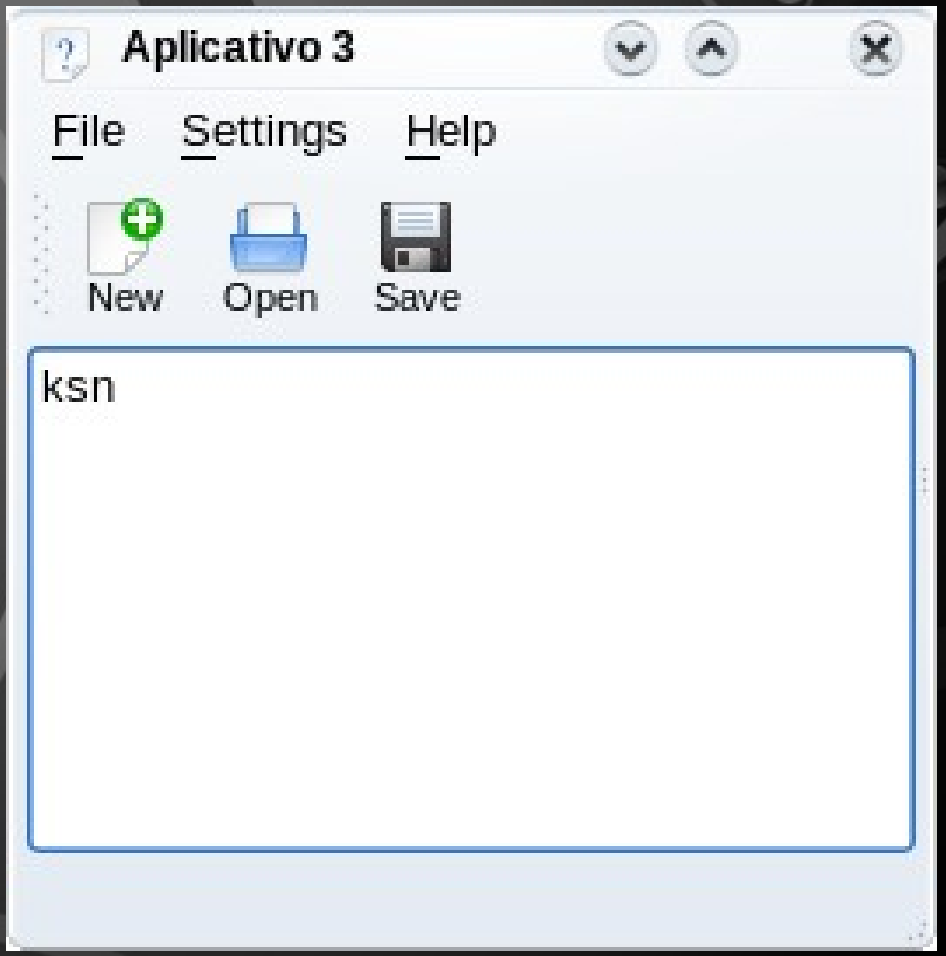
Introdução ao KDEUI

```
# Já que estamos utilizando a KIO, precisamos linkar com a KIO.

project (tutorial3)
find_package(KDE4 REQUIRED)
include_directories(${KDE4_INCLUDES})
set(tutorial3_SRCS main.cpp MainWindow.cpp)
kde4_add_executable(tutorial3 ${tutorial3_SRCS})
target_link_libraries(tutorial3 ${KDE4_KDEUI_LIBS}  ${KDE4_KIO_LIBS})

install(TARGETS tutorial3 DESTINATION ${BIN_INSTALL_DIR})
install(FILES Aplicativo2ui.rc DESTINATION ${DATA_INSTALL_DIR}/tutorial3)
```

CMakeLists.txt

Introdução ao KDEUI

- Compilando:
  - *cmake .*
  - *make*
  - *sudo make install*
  - *tutorial3*



FIM

- Usem o Qt e o KDE !
- Participem de projetos do KDE4 !
- Dúvidas, sugestões, colaborações:

sandro.andrade@gmail.com

tomaz.canabrava@gmail.com