



The Qt and KDE Frameworks: An Overview

Kevin Ottens



- 1 Introduction
- 2 Overview
- 3 Develop with Qt
- 4 Develop with KDE



- 1 Introduction
- 2 Overview
- 3 Develop with Qt
- 4 Develop with KDE



Prerequisites and Goals

Knowledge is a brick wall built line after line



C++

- Use of object oriented concepts
- Templates (at least a bit, but nothing hardcore)

*nix

- To know what is a shell
- Recall the overall structure of a typical Unix based desktop

Goals

- Give an overview of what can be done with Qt and KDE
- Give clues on how to go further
- Eventually give you the will to contribute 😊



History (1/4)

Since our past experiences determine what we are



Trolltech

- 1994: Trolltech creation in Oslo, Norway
- 1996: First Qt selling! (ESA)
- 1999: Qt2, offices in Australia
- 2000: Qt/Embedded, offices in the US
- 2000: Qt/X11 available under the GPL!
- 2001: Sharp uses Qtopia in its products
- 2001: Qt3!
- 2002: Teambuilder is released
- 2003: Qt/Mac available under the GPL!
- 2004: Qtopia Phone Edition is released
- 2005: Qt4!! offices in China



History (2/4)

Since our past experiences determine what we are



KDE: Warmup

- 14 October 1996: Matthias Ettrich announce on Usenet the "Kool Desktop Environment"
- Willing to use Qt which already had a lot of potential
- November 1996: kdelibs-0.0.1.tar.gz
- Just before Christmas: kwm, kpanel et kfm...
- February 1997: creation of the KDE-FreeQt Foundation

KDE: First steps in the media

- May 1997: Linux-Kongress presentation
- August 1997: First paper in a german journal
- 28 August 1997: First developers meeting



History (3/4)

Since our past experiences determine what we are



KDE: First versions

- 12 July 1998: KDE 1.0 (after a few betas)
- Availability of OpenParts (CORBA based), and of KMail
- Other versions (1.1, 1.1.1, 1.1.2)
- Second developers meeting
 - Move away from CORBA, creation of KParts
 - Matthias et Preston Brown get drunk and think they can write an ORB in one night...
 - ... the result is DCOP!
- 23 October 2000: KDE 2 (Konqueror, KParts, KIO, KOffice, DCOP)
- Konqueror is the first web browser to fully support the CSS2 specification



History (4/4)

Since our past experiences determine what we are



KDE: Fame and success

- 2001: talks about KDE in all the FOSS conferences
- 2 awards at the LinuxWorldExpo, 3 awards from the Linux Journal
- 25 February 2002: Third developers meeting
- 3 April 2002: KDE 3.0
- 22 August 2003: Kastle (Czech Republic)
- 3 February 2004: KDE 3.2
- 21 August 2004: aKademy (Germany)
- 26 August 2005: aKademy (Spain)
- 29 November 2005: KDE 3.5
- 14 October 2006: KDE has ten years

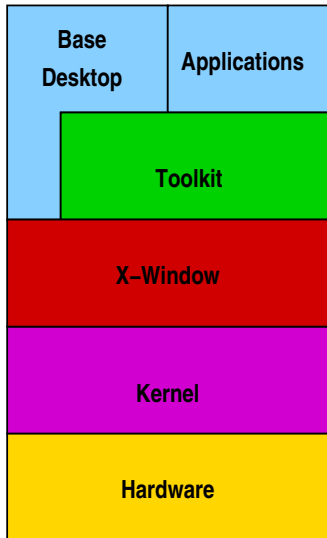


- 1 Introduction
- 2 Overview
- 3 Develop with Qt
- 4 Develop with KDE



Unix Desktop Software Stack

Like an onion, to find the heart you have to remove the layers



In our case...

- Base Desktop == KWin, Plasma, etc.
- Applications == Konqueror, KMail, Konversation, etc.
- Toolkit == Qt, kdelibs
- X-Window == XFree86 ou X.org
- Kernel == Linux, *BSD...
- Hardware == anything that can run a X server

KDE applications work under Windows or Mac OS X



Qt, an overview

Far away everything looks simpler



From the authors

- "Qt is a comprehensive C++ application development framework [...]"
- "Code Less. Create More." (Trolltech's slogan)

What Qt offers

- Slightly "modified" C++
- Tools and a *multi-platform* API to...
 - ... develop graphical user interfaces
 - ... make multi-threaded applications
 - ... implement network and inter-process communication
 - ... access databases, or process XML
 - ... interact with OpenGL
 - ... and much more!



KDE, an overview

Hm, it looks huge... even from far away



From the authors

- "KDE is a powerful Free Software graphical desktop environment for Linux and Unix workstations[...]"

What KDE offers

- A desktop environment
- Loads of applications
- A development framework
 - network transparency
 - component based architectures
 - interacting with the hardware
 - multimedia (audio, vidéo, streaming...)
 - natural language processing
 - and much more!



- 1 Introduction
- 2 Overview
- 3 Develop with Qt**
- 4 Develop with KDE



- 3** Develop with Qt
 - Key concepts and tools
 - Complex Widgets
 - IPC with D-Bus
 - Canvas, animations
 - Automated tests



QObject

The mother of all thing... or almost



Features

- Parent/child relationship, useful for
 - Memory management
 - Widgets layout
- Signals and slots
- Introspection
- Properties system

Drawback

- Difficulties for multiple inheritance



Signals and Slots (1/4)

Morse is overrated



Metaphore

- An object "emits" a signal to signify something potentially interesting for the outside
- One or more objects receive the signal thanks to a method having a compatible signature

Primitives

- `connect()` / `disconnect()`
- `emit`

Advantages

- Loose coupling
- Easy event based programming



Beacon

```
#include <QtCore/QObject>
#include <QtCore/QPoint>

class Beacon : public QObject
{
    Q_OBJECT

signals:
    void beamOfLight(QPoint pos, int degree);
};
```



Boat

```
#include <QtCore/QObject>
#include <QtCore/QPoint>

class Boat : public QObject
{
    Q_OBJECT

public slots:
    void lightSpotted(QPoint pos, int degree);
};
```



Signals and Slots (4/4)

Morse is overrated



Connecting

```
Beacon *lighthouse;
```

```
Boat *tanker;
```

```
[...]
```

```
connect(lighthouse,  
        SIGNAL(beamOfLight(QPoint, int)),  
        tanker,  
        SLOT(lightSpotted(QPoint, int)));
```



Implicit sharing

What is your is mine... almost



"Copy on write"

- Usable by value
- `operator=()` only duplicate a pointer
- Duplicate data on writing
- Reentrant operation



Implicit sharing

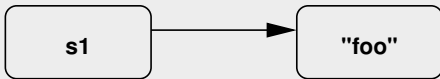
What is your is mine... almost



"Copy on write"

- Usable by value
- operator=() only duplicate a pointer
- Duplicate data on writing
- Reentrant operation

```
QString s1 = "foo";
```





Implicit sharing

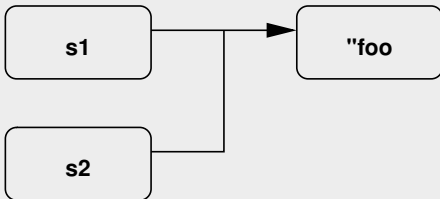
What is your is mine... almost



"Copy on write"

- Usable by value
- `operator=()` only duplicate a pointer
- Duplicate data on writing
- Reentrant operation

```
QString s2 = s1;
```





Implicit sharing

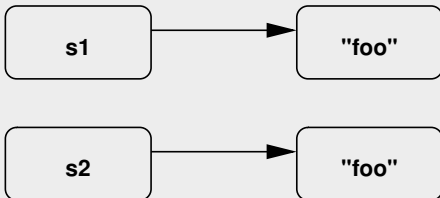
What is your is mine... almost



"Copy on write"

- Usable by value
- `operator=()` only duplicate a pointer
- Duplicate data on writing
- Reentrant operation

```
s1.append("/bar");
```





Implicit sharing

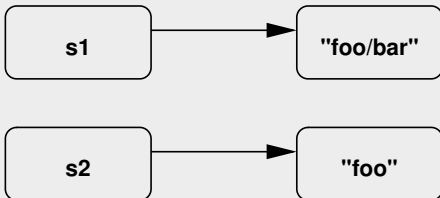
What is your is mine... almost



"Copy on write"

- Usable by value
- `operator=()` only duplicate a pointer
- Duplicate data on writing
- Reentrant operation

```
s1.append("/bar");
```





Implicit sharing

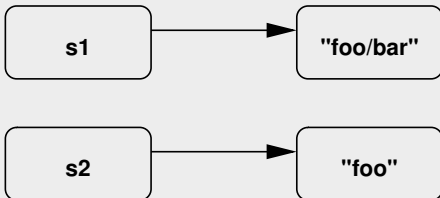
What is your is mine... almost



"Copy on write"

- Usable by value
- `operator=()` only duplicate a pointer
- Duplicate data on writing
- Reentrant operation

```
s1.append("/bar");
```





Implicit sharing

What is your is mine... almost



"Copy on write"

- Usable by value
- `operator=()` only duplicate a pointer
- Duplicate data on writing
- Reentrant operation

Covered classes

Images, polygons, strings, url, variants, collections...



Implicit sharing

What is your is mine... almost



"Copy on write"

- Usable by value
- `operator=()` only duplicate a pointer
- Duplicate data on writing
- Reentrant operation

(About collections)

- Everything necessary: `QList`, `QMap`, `QSet`, `QMultiMap`...
- Three available iteration styles:
 - STL iterators
 - Java iterators
 - `foreach()`



Graphical User Interface

Ooooh, beautiful!



Available bricks

- "Regular" widgets (QPushButton, QCheckBox, QLineEdit...)
- "Complex" widgets (QTextEdit, QTreeView...)
- OpenGL display
- QGraphicsView canvas

Short demonstrations?

- Borrowed and adapted from Qt examples
- texture: display an OpenGL object (243 lines of C++)
- widgets: showcase of a few Qt widgets with styling support (237 lines of C++)



Qt Designer (1/2)

Draw to win



Facts

- Writing GUI code by hand is boring
- Resulting code is generally heavy to maintain

designer + uic

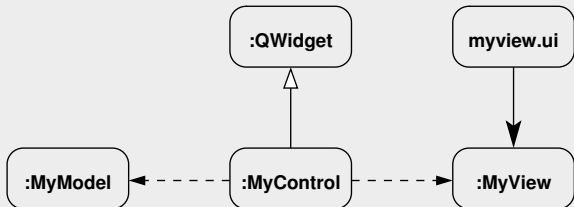
- Designer: GUI graphical editor
- Result: ".ui" files
- uic: generate C++ classes from ".ui" files

Way to work

- We keep **only** the ".ui" files
- Classes generated only during the build
- Push to apply MVC



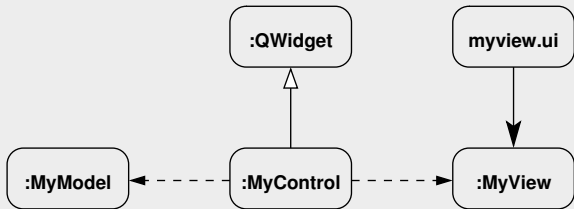
MVC with Designer



- MyView is generated by uic from myview.ui
- MyModel is a domain class to display or edit
- MyControl listen to MyModel and MyView widgets signals
- Inheriting from QWidget allow to use the control part like a build block in a more complex GUI



MVC with Designer



- MyView is generated by uic from myview.ui
- MyModel is a domain class to display or edit
- MyControl listen to MyModel and MyView widgets signals
- Inheriting from QWidget allow to use the control part like a build block in a more complex GUI



QMake (1/2)

Building is my passion... hmmm, in fact not



Facilitate portability

- Describing application building rules
- Generate Makefile or Visual Studio files
- Possibility to add specific rules for a given platform

Important variables

- TEMPLATE: project type (app, lib, subdirs)
- HEADERS, SOURCES and FORMS: files used for building
- TARGET: name of the built product
- CONFIG: building options (debug, multi-thread...)



A short example

```
TEMPLATE = app
TARGET   = seashore_simulator

CONFIG += qt warn_on release

HEADERS  = beacon.h boat.h
SOURCES  = main.cpp beacon.cpp boat.cpp
FORMS    = seashoreview.ui
```



3 Develop with Qt

- Key concepts and tools
- **Complex Widgets**
- IPC with D-Bus
- Canvas, animations
- Automated tests



Interview (1/3)

Model/View finally usable



About MVC

- **Architecture** Pattern
- Easy to understand... generally difficult to practice
- Useful for complex widgets or dialogs

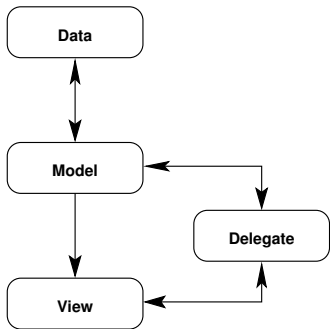
Interview Specifics

- MVC architecture in place for complex widgets
- "Generic" models interface (table/list/tree)
- Objects communication through signals and slots



Interview (2/3)

Model/View finally usable



Model/View/Delegate

- The model interfaces the data source with the other components
- The view asks to the model the data to be displayed
- The delegate does the rendering with the view, and during an editing it indicates the modifications to the model
- All the references to data are passed thanks to `QModelIndex`



Interview (3/3)

Model/View finally usable



Available classes

- `QAbstractItemModel`: Base interface for models
- `QAbstractListModel`: List oriented models
- `QAbstractTableModel`: Table oriented models
- `QListView`: List view
- `QTableView`: Table view
- `QTreeView`: Tree view
- `QAbstractProxyModel`: Filtering models interface

Short demonstration



And also...

In short!



Scribe: rich text management

- QTextDocument: read-only text content
 - Hierarchical structure
 - QTextBlock, QTextFrame, QTextTable, QTextList
- QTextCursor allows to manipulate the text using the cursor metaphor
- QTextEdit: text edit widget

Mainwindow: flexibility in the main window

- QMainWindow: Main window with a menu bar
- QToolBar: Tool bars on window borders
- QDockWidget: Movable and detachable panels tied to the window



3 Develop with Qt

- Key concepts and tools
- Complex Widgets
- **IPC with D-Bus**
- Canvas, animations
- Automated tests



Overview

D-Bus WTH?



D-Bus: Desktop Bus

- Grandson of DCOP: Desktop COmmunication Protocol
- Remote Procedure Call
- Keeps some concepts of Qt

Model

- Each application using D-Bus is a service
- The services communicate through a bus
 - Session bus, attached to a user session
 - System bus, unique for the system
- Each service exposes a hierarchy of objects
- Objects expose interfaces (methods+signals)



A few tools

Play with your desktop



qdbusviewer: graphical tool

- Useful for developers and power users
- List services, available objects and interfaces
- Allows to call methods

qdbus: command line tool

- Same features than qdbusviewer
- Ease shell scripts writing to control applications



Make a D-Bus call (1/2)

Wow! Complicated...



Method call using QDBusMessage

```
#include <QtDBus/QDBusConnection>
#include <QtDBus/QDBusMessage>
[...]
QDBusConnection bus
    = QDBusConnection::sessionBus();
QDBusMessage msg
    = QDBusMessage::createMethodCall(
        "org.kde.application",
        "/path/to/object",
        "org.kde.Interface",
        "method");
msg << 2 << "foo" << 0.5;
// =>
```



Make a D-Bus call (2/2)

Wow! Complicated...



Method call using QDBusMessage

```
// =>
QDBusMessage reply = bus.call(msg);
QList<QVariant> args = reply.arguments();
if (reply.type()==QDBusMessage::ReplyMessage
    && args.size()==1
    && args.at(0).canConvert(QVariant::StringList)) {

    QStringList result
        = args.at(0).toStringList();
    [...]
}
```



Method call using QDBusInterface

```
#include <QtDBus/QDBusConnection>
#include <QtDBus/QDBusInterface>
[...]
QDBusInterface interface("org.kde.application",
                        "/path/to/object",
                        "org.kde.Interface");
QDBusReply<QStringList>
    = interface.call("method", 2, "foo", 0.5);

if (reply.isValid()) {
    QStringList result = reply;
    [...]
}
```



Expose an object on the bus

Expose yourself to the world!



Use of `QDBusConnection::registerObject()`

```
class Example : public QObject
{
    Q_OBJECT
    [...]
public slots:
    Q_SCRIPTABLE QStringList method(int, QString);
    Q_SCRIPTABLE Q_NOREPLY void asyncMethod();
};
[...]
```

```
Example *ex = new Example;
QDBusConnection c = QDBusConnection::sessionBus(),
c.registerObject("/path/object", ex,
    QDBusConnection::ExportScriptableSlots);
```



Define a D-Bus interface (1/2)

Expose yourself to the world!



XML Specification

```
<node>
  <interface name="org.kde.Dummy">
    <method name="method">
      <arg name="arg1" type="i" direction="in"/>
      <arg name="arg2" type="s" direction="in"/>
      <arg name="ret" type="as" direction="out"/>
    </method>
    <method name="asyncMethod">
      <annotation
        name="org.freedesktop.DBus.Method.NoReply"
        value="true"/>
    </method>
  </interface>
  [...]
</node>
```



Define a D-Bus interface (2/2)

Expose yourself to the world!



Code generation

- `qdbusxml2cpp -a ...` : generate a QtDBus adapter
- `qdbusxml2cpp -p ...` : generate a QtDBus interface

Adapter use

```
#include "dummy_adaptor.h"
[...]  
Dummy::Dummy() : QObject()  
{  
    new DummyAdaptor(this);  
    QDBusConnection c  
        = QDBusConnection::sessionBus();  
    c.registerObject("/path/object", this);  
}
```



Define a D-Bus interface (2/2)

Expose yourself to the world!



Code generation

- `qdbusxml2cpp -a ...` : generate a QtDBus adapter
- `qdbusxml2cpp -p ...` : generate a QtDBus interface

Interface use

```
#include "dummy_interface.h"  
[...]  
  
org::kde::Dummy *iface = new org::kde::Dummy(  
    "org.kde.service", "/path/to/object",  
    QDBusConnection::sessionBus(), this);  
  
QStringList list = iface->method(2, "foo");
```




- 3** Develop with Qt
 - Key concepts and tools
 - Complex Widgets
 - IPC with D-Bus
 - **Canvas, animations**
 - Automated tests



Arthur (1/2)

Draw me a round table!



Architecture

- QPainter: take care of the drawing operations (drawLine(), drawRect(), etc.)
- QPaintDevice: object on which you can draw thanks to a QPainter
 - QWidget
 - QPixmap, QImage, QPicture
 - QPrinter
- QPaintEngine: interface used by QPainter to effectively paint
 - Provided by a QPaintDevice
 - Several backends available (raster, OpenGL, PostScript, CoreGraphics, X11 with XRender)



Arthur (2/2)

Draw me a round table!



Noticeable points

- Double buffering on widgets
- Advanced drawing operations
 - Brushes `QGradient` and sons (linear, radial, conical)
 - Alpha channel support in `QColor`
 - Anti-aliasing:
`painter.setRenderHint(QPainter::Antialiasing)`
- Tiny SVG 1.2 support since Qt 4.1
 - `QSvgWidget`: display able to render a SVG
 - `QSvgRendered`: allors to render an SVG on any `QPaintDevice`

Short demonstration: `pathstroke` and `svgviewer`



QGraphicsView

Model/View split for the canvas



Model

- QGraphicsScene
 - Allows to manage the objects in the scene
 - Distributes events to objects and manage their state
- QGraphicsItem, displayable objects
 - Texts, lines, polygons, pixmaps, SVGs, etc.
 - Collision detection
 - Drag'n'Drop, keyboard and mouse events, etc.

View

- Several views on the same scene
- Each view can have different render options, or different transformations

Demo "40 000 Chips"



Easy animations

- Control the animation evolution
 - Indicates the current frame
 - Indicates the animation progress
- Pausing is possible
- Reverse mode, loops

The cherry of the cake

- Automatic framerate control
- Several progress curves

Short demo (95 lines)



- 3** **Develop with Qt**
 - Key concepts and tools
 - Complex Widgets
 - IPC with D-Bus
 - Canvas, animations
 - **Automated tests**



Unit tests (Qt 4.1)

I love tests, oh yes!



QTestLib

- Unit tests framework adapted to Qt
- Data oriented tests
- Basic features for GUI testing

Available tools

- `QTEST_MAIN()`: Create the main for the test application
- `QVERIFY()`: Verify a condition
- `QCOMPARE()`: Verify equality of two values
- `QSignalSpy`: Introspection on signals



Typical test class

Tests! Tests! Tests!



```
class TestFoo : public QObject
{
    Q_OBJECT

private slots:
    void initTestCase();
    void testSomeStuff();
    void testWayMoreStuff();
    void testWayMoreStuff_data();
    void testFooSignal();
};
```




Typical test class

Tests! Tests! Tests!



```
QTEST_MAIN(TestFoo)

void TestFoo::testSomeStuff()
{
    FiboGen fg;

    QVERIFY(fg.isValid());
    QCOMPARE(fg.compute(0), 0);
    QCOMPARE(fg.compute(1), 1);
    QCOMPARE(fg.compute(2), 1);
    QCOMPARE(fg.compute(3), 2);
    QCOMPARE(fg.compute(4), 3);
    QCOMPARE(fg.compute(5), 5);
    [...]
}
```



Typical test class

Tests! Tests! Tests!



```
void TestFoo::testWayMoreStuff_data()
{
    QTest::addColumn<QString>("server");
    QTest::addColumn<int>("port");
    QTest::addColumn<QString>("path");
    QTest::addColumn<long>("value");

    QTest::newRow("Nominal case")
        << "service.foo.com" << 123
        << "/random/path" << 200;
    QTest::newRow("Broken path")
        << "service.foo.com" << 123
        << "/bad/path" << 100;
    [...]
}
```



Typical test class

Tests! Tests! Tests!



```
void TestFoo::testWayMoreStuff()
{
    QFETCH(QString, server);
    QFETCH(int, port);
    QFETCH(QString, path);
    QFETCH(long, value);

    Protocol p;
    p.connect(server, port);
    QString reply = p.query(path);
    QCOMPARE(reply.toLong(), value);
}
```



Use of QSignalSpy

Beware! I'm watching you...



```
void TestFoo::testFooSignal()
{
    QPoint pos(10, 20);
    Beacon *b = new Beacon(pos);
    QSignalSpy spy(b, SIGNAL(beam(QPoint, int)));
    beacon.rotate(15); // emits (10,20), 15
    beacon.rotate(20); // emits (10,20), 35

    QCOMPARE(spy.count(), 2);
    QCOMPARE(spy.at(0).at(0).toPoint(), pos);
    QCOMPARE(spy.at(0).at(1).toInt(), 15);
    QCOMPARE(spy.at(1).at(0).toPoint(), pos);
    QCOMPARE(spy.at(1).at(1).toInt(), 35);
}
```



- 1 Introduction
- 2 Overview
- 3 Develop with Qt
- 4 Develop with KDE



4 Develop with KDE

- Build system: CMake
- kdecop & kdeui
- KParts
- KIO
- Phonon
- Solid
- Plasma



Features

Die autohell! Die!!



Summary

- Generate files for the native build system
 - GNU Make
 - KDevelop
 - XCode
 - Visual Studio
- Separation between source and build directories

Why use CMake?

- Build files easy to write
- Portability
- Faster builds (no libtool)



Example

Do your best with what I give you



CMakeLists.txt

```
project(myproject)
find_package(KDE4 REQUIRED)
include (KDE4Defaults)
include_directories(${KDE4_INCLUDES})

add_subdirectory(lib)

set(projectSources main.cpp someclass.cpp)
kde4_add_ui_files(projectSources view.ui)
kde4_add_kcfg_files(projectSources settings.kcfg)
kde4_add_executable(myproject projectSources)
target_link_libraries(myproject ${KDE4_KIO_LIBS}
                       mylib)
```




Utilisation

The toad becoming charming prince... or the other way around



Preparing directories

- We assume the sources are in `project-src`
- `mkdir project-build; cd project-build`

Launching cmake

- `cmake ../project-src`
- Giving options is possible
 - `cmake -DCMAKE_INSTALL_PREFIX=/opt/kde4 ../project-src`
 - `cmake -DKDE4_BUILD_TESTS=ON ../project-src`
 - ...

Get back to the good old habits

- `make && make install`



4 Develop with KDE

- Build system: CMake
- kdecop & kdeui
- KParts
- KIO
- Phonon
- Solid
- Plasma



Convenience classes and methods



No need to make our lives uselessly complicated...

Visual integration

- `KStandardItem`: functions creating standard actions (`print()`, `saveAs()`...)
- `KIcon`: icons loaded by name respecting the current theme, with cache

System

- `KApplication`: session management, standard menus, D-Bus registering, etc.
- `KProcess`: `system()` under steroids
- `KStandardDirs`: find the resources of the application on disk



i18n

- Eighteen letters between i and n. - Hmm... "internationalisation"!



KLocale allows to...

- translate (`i18n()`)
- know the local currency
- use the right decimal separator
- correctly format dates

KBabel

- Base tool for translators
- Indicates the progress of the translation
- Allows to query dictionaries
- Spell checking integrated



Manage application settings

And now I check this one, disable this...



Configuration system: KConfig

- Default backend format similar to ".ini" files (grouped key/value sets)
- Takes care of the user language
- System configuration handled

To go further

- KConfigXT
 - XML description of the application settings
 - Automatic code generation (singleton)
- Kiosk
 - Locking of settings in the system configuration
 - Users profile management



Standard dialogs

Heteroclite monologues



KConfigDialog

- Manage the whole dialog life cycle
 - Memory freeing
 - Buttons activation state (defaults, ok, apply, cancel)
- Works with KConfigXT and Designer

KFileDialog (kio/kfile)

- User KIO: network transparency
- File previews

Short demonstration KConfigXT, KConfigDialog,
KFileDialog



4 Develop with KDE

- Build system: CMake
- kdecop & kdeui
- KParts
- KIO
- Phonon
- Solid
- Plasma



Overview

Let's explode our system in parts!



Graphical components system

- kdelibs has the needed facilities for component based architectures
- KParts is a family of those components
- "A widget with the associated feature, available as actions" (David Faure TM)

Types of KParts available

- ReadOnlyPart: file display
- ReadWritePart: file editor
- BrowserExtension: browser logic integration

Component development is not covered here...



Loading a component

It has to be useful...



Loading "by hand"

- `KPluginLoader`: load a library dynamically while application is running
- `KPluginFactory`: create components available in a library
- `KServiceTypeTrader` and `KMimeTypeTrader`: query the system to know the libraries available for a given set of constraints

"Intelligent" loading

- `KMimeTypeTrader::createInstanceFromQuery()`
- `KServiceTypeTrader::createInstanceFromQuery()`



Example of usage

Woot! A web browser!



Demonstration D-Bus+KParts

- Code presented by George Staikos during the first OSDW (<http://www.osdw.org>)
- Slightly modified
- For 99 lines of code, we get
 - A working web browser...
 - D-Bus driven
- With 66 more lines (written for this talk), we get
 - A minimalist browser interface...
 - Able to control all the browsers of a sessions



4 Develop with KDE

- Build system: CMake
- kdecore & kdeui
- KParts
- **KIO**
- Phonon
- Solid
- Plasma



Implement protocols (slaves)

- Regular network protocols (pop3://, smtp://, etc.)
- Network filesystems (ftp://, webdav://, etc.)
- Virtual filesystems (system://, trash://, etc.)

Deal with protocols and files

- MIME types system (KMimeType)
- Obtain information about a protocol (KProtocolInfo)
- Use a protocol (send/receive data, create directories, etc.)



Synchronous Use

Go fetch it! I'm waiting here.



KIO::NetAccess

- `download()`: fetch the data of an URL in a temporary file
- `removeTempFile()`
- `upload()`: symmetric of `download()`
- and also `mkdir()`, `move()`, etc.

Example

```
QString tmpFile;  
if(KIO::NetAccess::download(u, tmpFile, window)){  
    loadFile(tmpFile);  
    KIO::NetAccess::removeTempFile(tmpFile);  
}
```



Asynchronous User

Hey! I've more to do than waiting...



KIO::Job

- Create a job instance with a KIO method (`KIO::stat()`, etc.)
- Connect the interesting signals of the job
 - `result()` being the minimum required to know that the job is finished
- No need to keep a pointer on the job or to deallocate it, it deletes itself automatically after `result()` is emitted

Short demonstration (88 lines)



Create your ioslave

Work! You lazy!



KIO::SlaveBase

- Inherit the class
- Overload the supported methods
 - Each method maps to a possible operation (`copy()`, `get()`, etc.)
 - They're all "void" methods, and you need to give the results back to the calling job
 - For this we use specific methods (`data()`, `error()`, `finished()`, etc.)
- Provide a ".protocol" file
 - Describes the implemented protocol
 - Information available through `KProtocolInfo`

Short demonstration (107 lines)



4 Develop with KDE

- Build system: CMake
- kdecopre & kdeui
- KParts
- KIO
- Phonon
- Solid
- Plasma



Phonon

An environment which pump up the volume!



Identity

- Leader: Matthias Kretz
- Goal: Strengthen the multimedia abilities of KDE

Technical details

- High-level API designed by collecting use cases
- Backends, allowing to support as many multimedia frameworks as needed
 - Xine, GStreamer, NetworkMultiMedia (NMM)
 - DirectX
 - QuickTime
- Unit tests, and validation tests for backends



Sources and their management

- `MediaSource`: multimedia source (audio, video) coming from a file, an URL, a disc, etc.
- `MediaObject`: control the stream from a source, queue management

Paths

- `MediaNode`: node of a pipeline
 - `MediaObject`: it's also a `MediaNode`
 - `AudioOutput`: audio output (soundcard, network, etc.)
 - `VideoWidget`: video display
- `Path`: connects two nodes, effects injection

Demo, "phonon-player" (53 lines)



Complementary tools

We all want our life to be easier



Widgets

- `VolumeSlider`: control the volume of an `AudioOutput`
- `SeekSlider`: control the progress of a `MediaObject`
- `EffectWidget`: effect configuration

Utilities

- `VolumeFaderEffect`: dynamic setting of the volume (fade to silence, cross-fading...)
- `VideoPlayer`: basic video player



4 Develop with KDE

- Build system: CMake
- kdecopre & kdeui
- KParts
- KIO
- Phonon
- **Solid**
- Plasma



Identity

- Leader: hmm... yeah ok... I'm guilty
- Goal: Improve interaction between hardware and desktop applications

Technical details

- Architecture with backends
- "Fake backend" for unit tests writing
- Several domains
 - Hardware discovery
 - Network management
 - Power management
- High level API: make developers life easier



Hardware discovery (1/2)

What's good in there?



Principles

- The system has Devices, each one having a unique identifier
- Devices are organized in a hierarchy
- Each Device has interfaces of different types
- The set of DeviceInterfaces from a Device describes what the device can do

Notifications

- `Solid::DeviceNotifier::deviceAdded(QString)`
- `Solid::DeviceNotifier::deviceRemoved(QString)`



Hardware discovery (2/2)

What's good in there?



Find devices

- `Solid::Device::allDevices()`
- `Solid::Device::devicesFromType()`
- `Solid::Device::devicesFromQuery()`

```
QList<Solid::Device> all  
    = Solid::Device::allDevices();
```



Hardware discovery (2/2)

What's good in there?



Find devices

- `Solid::Device::allDevices()`
- `Solid::Device::devicesFromType()`
- `Solid::Device::devicesFromQuery()`

```
QList<Solid::Device> processors
= Solid::Device::listFromType(
    Solid::DeviceInterface::Processor);
```




Hardware discovery (2/2)

What's good in there?



Find devices

- `Solid::Device::allDevices()`
- `Solid::Device::devicesFromType()`
- `Solid::Device::devicesFromQuery()`

```
QList<Solid::Device> usbDrives
    = Solid::Device::listFromQuery(
        "StorageDrive.bus == 'Usb'");
```



Hardware discovery (2/2)

What's good in there?



Find devices

- `Solid::Device::allDevices()`
- `Solid::Device::devicesFromType()`
- `Solid::Device::devicesFromQuery()`

Manipulate devices

- `Solid::Device::is<T>()`
- `Solid::Device::as<T>()`

```
Solid::Device dev = ...  
if (dev.is<Solid::Processor>()) {  
    ...  
}
```



Hardware discovery (2/2)

What's good in there?



Find devices

- `Solid::Device::allDevices()`
- `Solid::Device::devicesFromType()`
- `Solid::Device::devicesFromQuery()`

Manipulate devices

- `Solid::Device::is<T>()`
- `Solid::Device::as<T>()`

```
Solid::Device dev = ...
if (dev.is<Solid::Camera>()) {
    QVariant handle =
        dev.as<Solid::Camera>()->driverHandle();
    ...
}
```



Hardware discovery (2/2)

What's good in there?



Find devices

- `Solid::Device::allDevices()`
- `Solid::Device::devicesFromType()`
- `Solid::Device::devicesFromQuery()`

Manipulate devices

- `Solid::Device::is<T>()`
- `Solid::Device::as<T>()`

Demo, "storage-plug" (94 lines)



Solid::Networking

- `status()` / `statusChanged()`
- `shouldConnect()` / `shouldDisconnect()`

Solid::Powermanagement

- `appShouldConserveResources()`
- `requestSleep()`
- `begin/stopSuppressSleep()`



Solid::Networking

- `status()` / `statusChanged()`
- `shouldConnect()` / `shouldDisconnect()`

Solid::Powermanagement

- `appShouldConserveResources()`
- `requestSleep()`
- `begin/stopSuppressSleep()`



Solid::Networking

- `status()` / `statusChanged()`
- `shouldConnect()` / `shouldDisconnect()`

Solid::Powermanagement

- `appShouldConserveResources()`
- `requestSleep()`
- `begin/stopSuppressSleep()`



Solid::Networking

- `status()` / `statusChanged()`
- `shouldConnect()` / `shouldDisconnect()`

Solid::Powermanagement

- `appShouldConserveResources()`
- `requestSleep()`
- `begin/stopSuppressSleep()`



Solid::Networking

- `status()` / `statusChanged()`
- `shouldConnect()` / `shouldDisconnect()`

Solid::Powermanagement

- `appShouldConserveResources()`
- `requestSleep()`
- `begin/stopSuppressSleep()`



4 Develop with KDE

- Build system: CMake
- kdecopre & kdeui
- KParts
- KIO
- Phonon
- Solid
- Plasma



Plasma

An environment adapted to your needs



Identity

- Leader: Aaron Seigo
- Goal: Make the desktop elegant, and improve the human/computer interactions

Technical details

- *Kicker + KDesktop + SuperKaramba = Plasma*
- "Plasma Engine" proposing a set of services
- "Plasmoids" writable using different languages
 - C++ and ECMAScript supported out of the box
 - potentially Ruby, Python and Java on demand
- One "plasmoid", two forms



And now?



A few hints on contributing to KDE

We want you !

Why?

- Build a network
- Improve your skills
- Obtain the best out of the project

How?

- Write code
- Make donations
- Use your artistic talents
- Write documentation
- Translate!
- Evangelize
- Help other users
- Ask for it... use it



Websites (Qt)

For geeks attached to their keyboard



Trolltech

<http://www.trolltech.com>



Trolltech's Online Reference Documentation

<http://doc.trolltech.com>



QtCentre: The Ultimate Qt Community site

<http://www.qtcentre.org>



Trolltech Labs

<http://labs.trolltech.com>



Websites (KDE)

For geeks attached to their keyboard



KDE

<http://www.kde.org>



KDE Techbase

<http://techbase.kde.org>



KDE.News a.k.a "the Dot"

<http://dot.kde.org>



KDE-Apps: Applications for your KDE desktop

<http://www.kde-apps.org>



Planet KDE

<http://planetkde.org>



Questions?

Kevin Ottens
ervin@kde.org