basysKom

Von MFC nach Qt: Migrationswege für Softwarehersteller

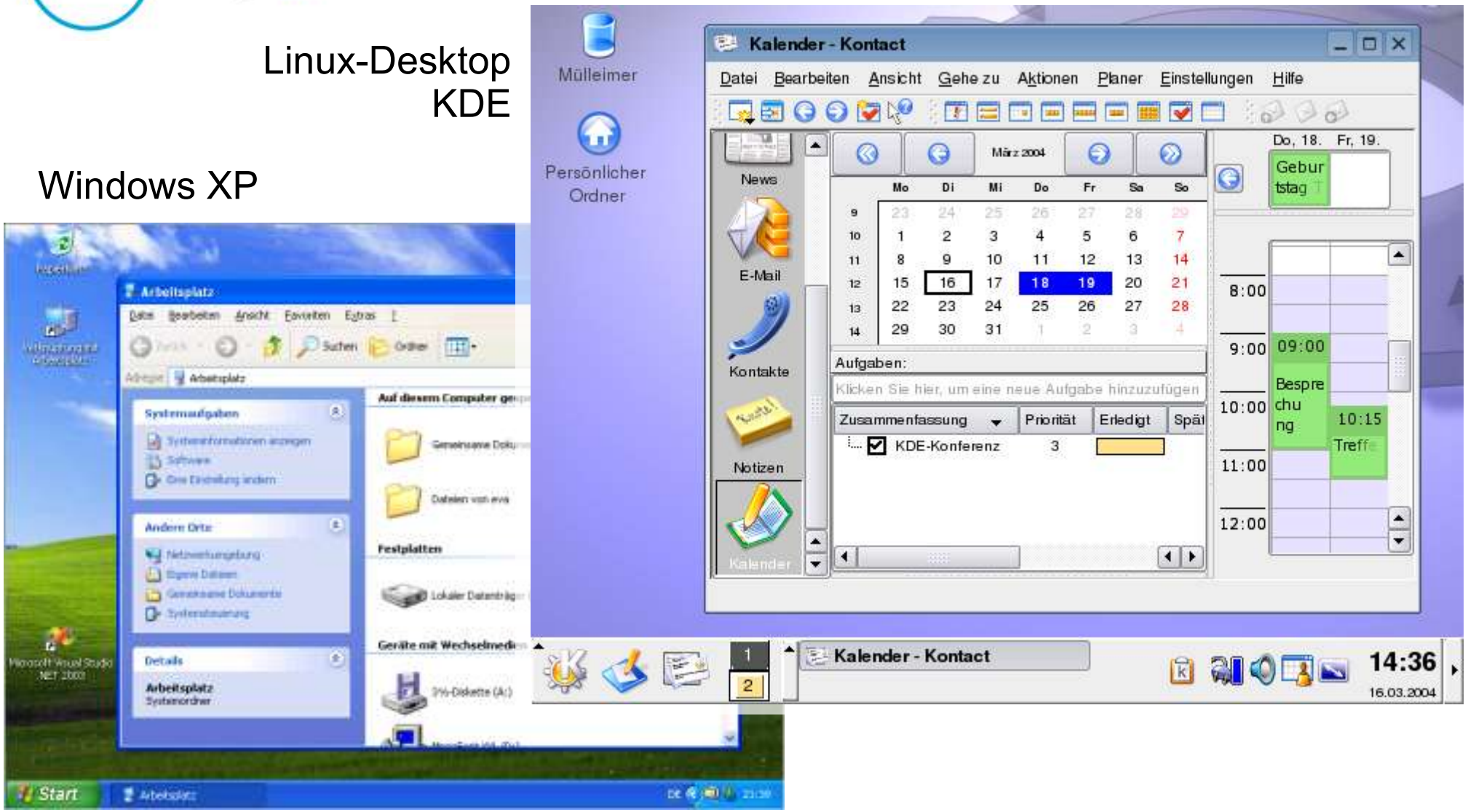
Eva Brucherseifer,
basysKom GbR

Ingenieurbüro
Brucherseifer & Ackermann GbR
IT System Kompetenz

Der Linux-Desktop

Linux-Desktop
KDE

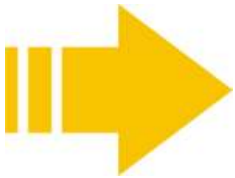
Windows XP



Herausforderung

- Visual Basic /
  Visual Basic for Applications
- C# und .NET
- **C++ mit MFC**

- **Fallbeispiel:**
  Adressverwaltung

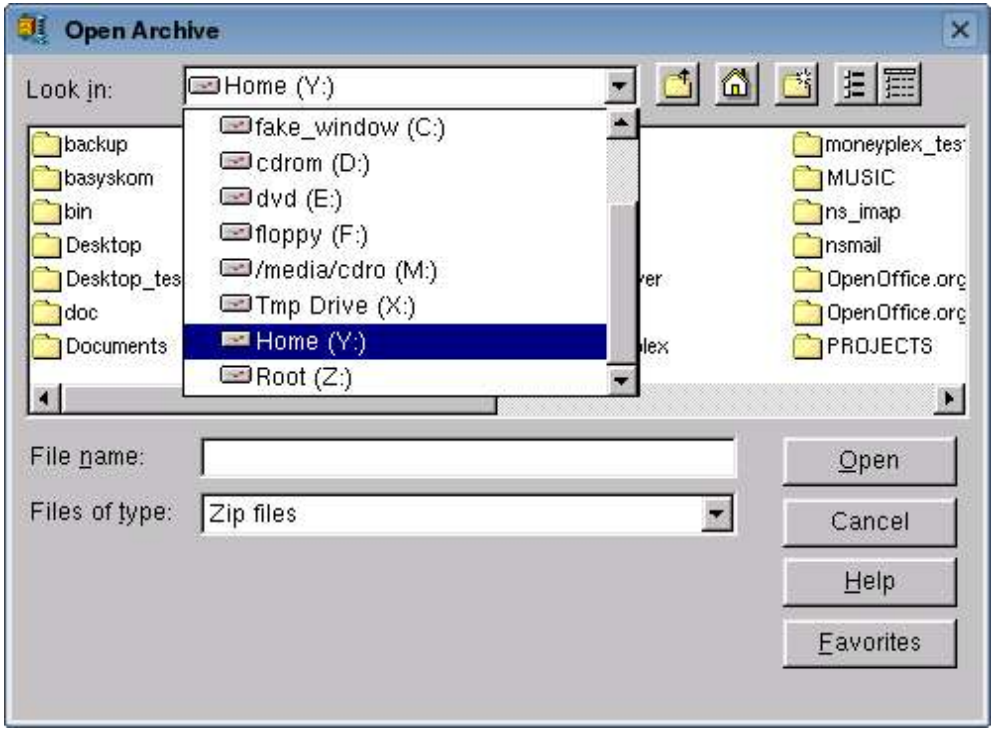


Zielplattformen

- Windows
  - → XP
- Linux und Unix
  - → KDE
- MAC OS X
  - → Aqua
- Linux/Embedded

Windows-Programme

- Terminal-Lösung
  - ➔ Terminal Server + Linux-Client
- Emulation mit Basissystem Linux
  - ➔ Vmware
  - ➔ Wine, CrossOver Office



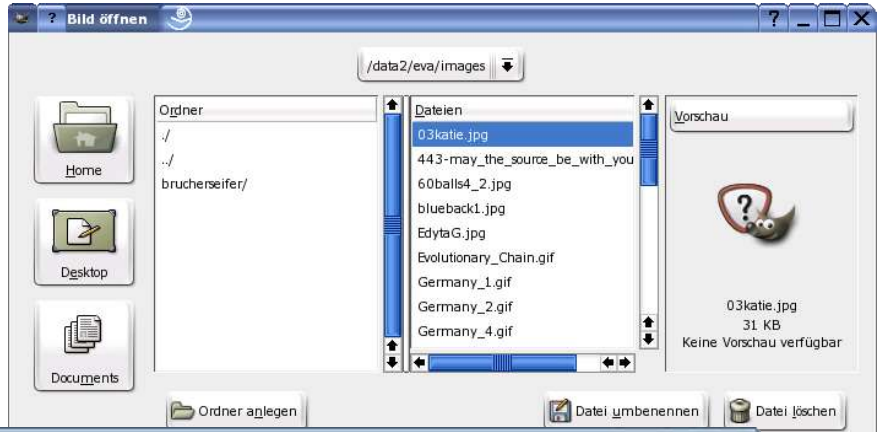
Wine-Emulation

➔ Portierung auf plattformunabhängigen Code

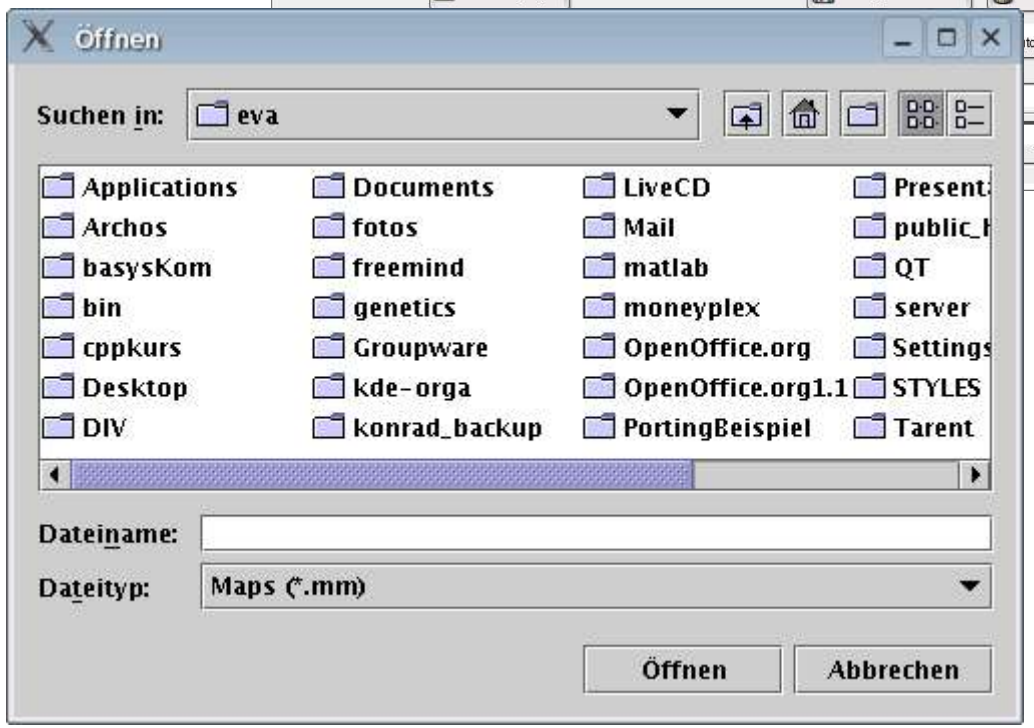
Portable Toolkits unter Linux

- Native Toolkits:
  - GNOME: Gtk (C)
  - **KDE: Qt (C++)**
- Java:
  - Swing, ATK, SWT
- Bindings:
  - Skriptsprachen, C, Java
  - wxWindows
  - Qt

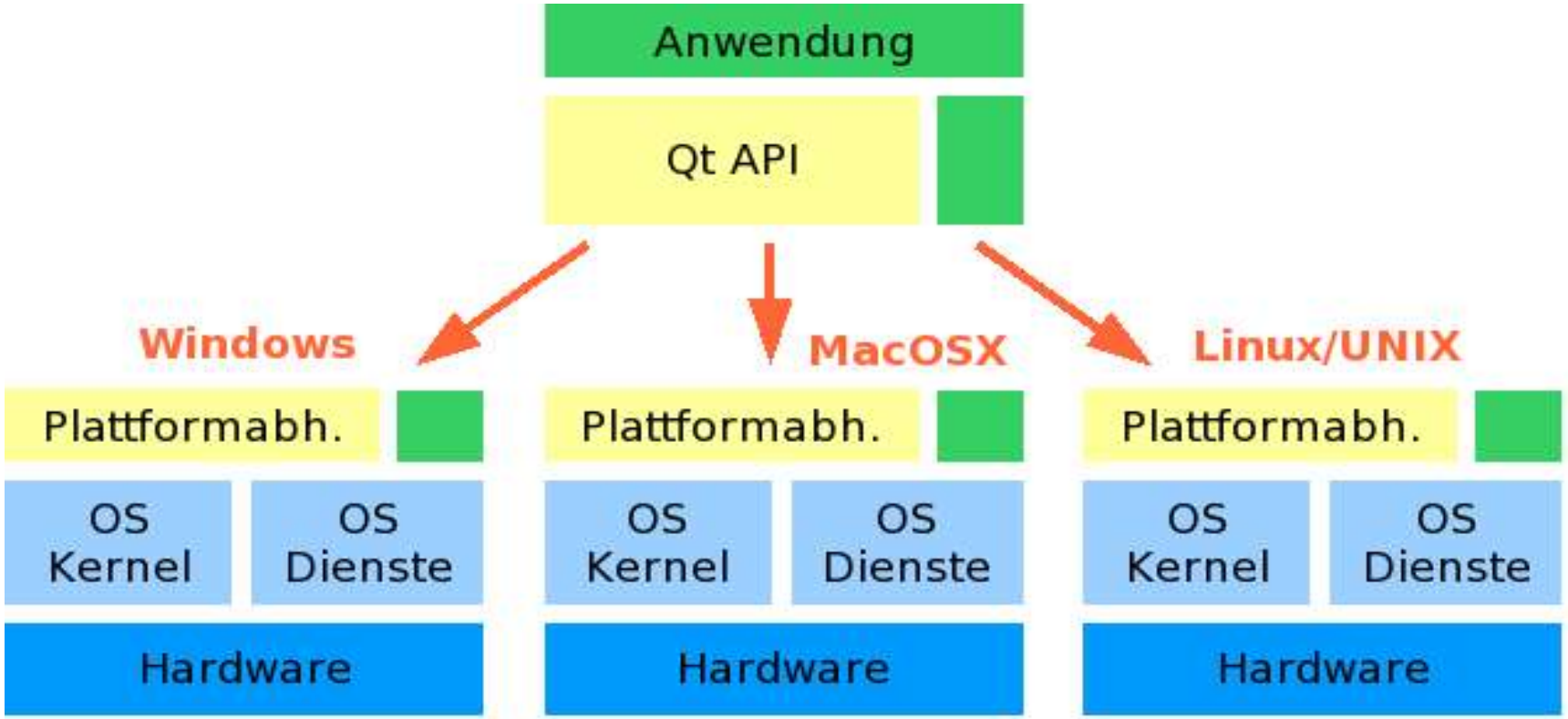
Gtk



Java



Plattformunabhängigkeit



Das Toolkit Qt

- Hersteller: Trolltech AS, Oslo
- Verfügbare Plattformen:
  - Windows (Win32/MFC)
  - MacOSX (Appearance Manager)
  - Linux/UNIX (nativ)
  - Linux embedded (nativ)
- Doppel-Lizenz:
  - GPL oder Qt Commercial License

Das Toolkit Qt

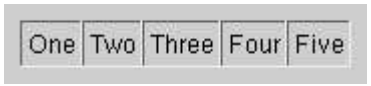
- Merkmale der Bibliothek:
  - C++, strikt objekt-orientiert
  - hohe Codequalität
  - gute Dokumentation
  - im Source-Code verfügbar
  - aktive Weiterentwicklung
  - Support und Training verfügbar durch Trolltech und Dritt-Anbieter

Qt- Funktionsumfang

- **Systemmethoden und Infrastruktur:**
  - Strings, Container, Netzwerk, Datenbank-Anbindung, XML, ...
- **GUI:**
  - Event-Handling, Widgets und Dialoge, Workspace-Frameworks, OpenGL, ...
- **Spezielle Infrastruktur:**
  - Übersetzungssystem, Plugin-Framework, Layout-Management, ActiveQt, MFC-Anbindung, ...
- **IDE:**
  - Qt Designer, Qt Linguist, Qt Assistant, qmake
  - Integration in KDevelop, Visual Studio

Qt- Funktionsumfang

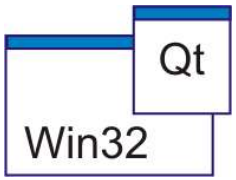
- **Layouts**
  - automatische Positionierung von Kindwidgets
  - Handhabung von Resize-Events
  - Automatische Updates, wenn sich der Inhalt ändert z.B. Fontgröße, Text, Inhalt von Kindwidgets



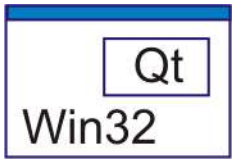
- **Signals und Slots**
  - Events und Methoden können dynamisch zur Laufzeit verbunden und gelöst werden

Schrittweise Migration

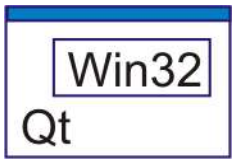
Qt Windows Migration Framework



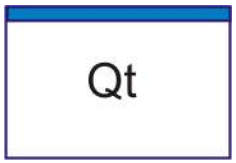
Qt-Dialoge können als externe dll's geladen werden



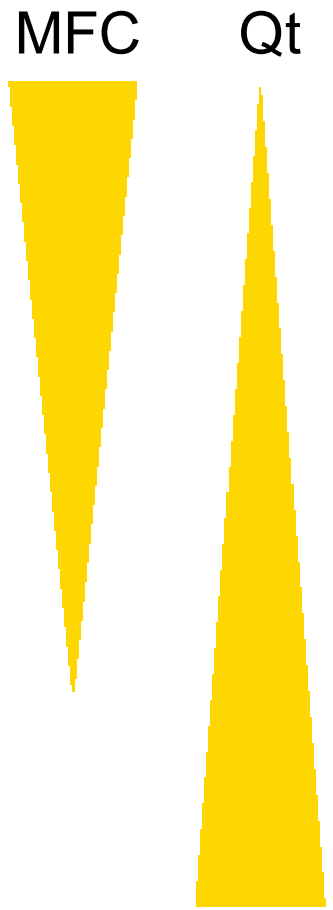
Qt-Widgets können in Win32-Programmen verwendet werden



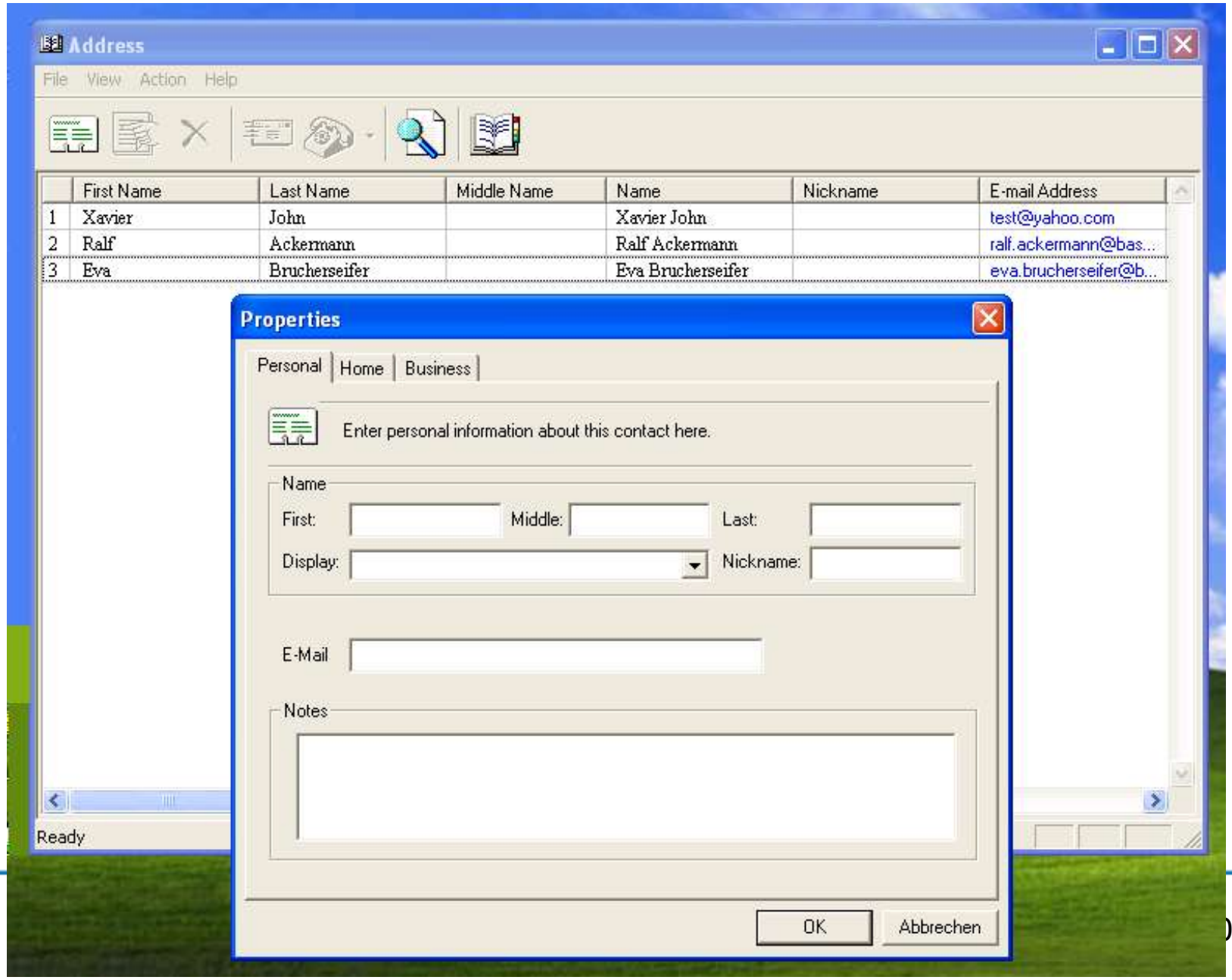
Win32-Widgets können in Qt-Programmen verwendet werden



reines Qt-Programm



Fallbeispiel: Address

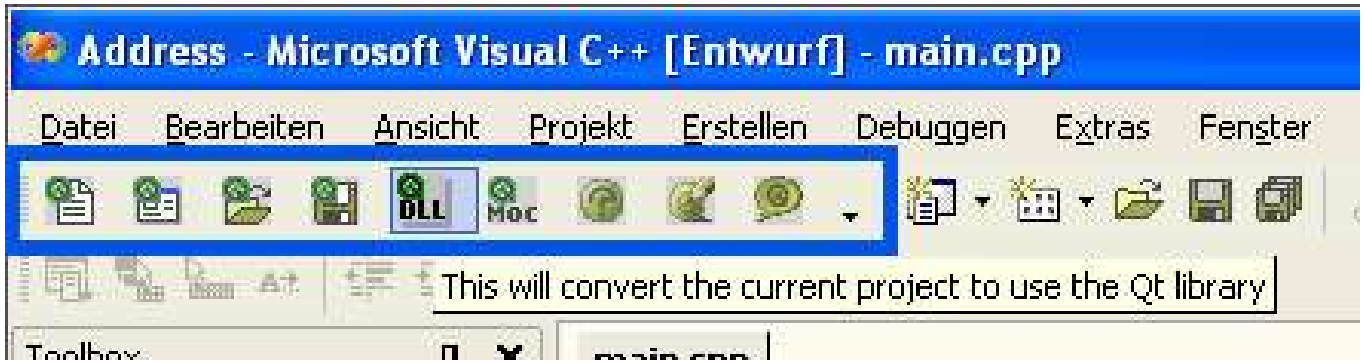


Fallbeispiel: Address

- Beispielanwendung:
  - „Address“
  - Quelle: codeguru.com
- Funktion:
  - Adressliste
  - Eingabe von Adressen in Editor
  - Abspeicherung in CSV-Datei
- Umfang:
  - 13.000 Zeilen Code
  - 35 Klassen
  - davon 20 Klassen für die Tabellenansicht der Adressen

Schritt 1: Vorbereitung

- Portierung:
  - unter Windows
  - Visual C++-Projekt
- Linken mit Qt-Bibliothek
  - Win/Qt 3.3.2
  - qtwinmigrate aus Qt Solutions



Schritt 1: Vorbereitungen

- Identifizierung der nicht-portablen Elemente
  - TAPI-Anbindung
  - Plazierung in Systemtray
- Identifizierung der Eigenentwicklungen, die durch Qt-Klassen ersetzt werden sollen

Schritt 1: Vorbereitungen

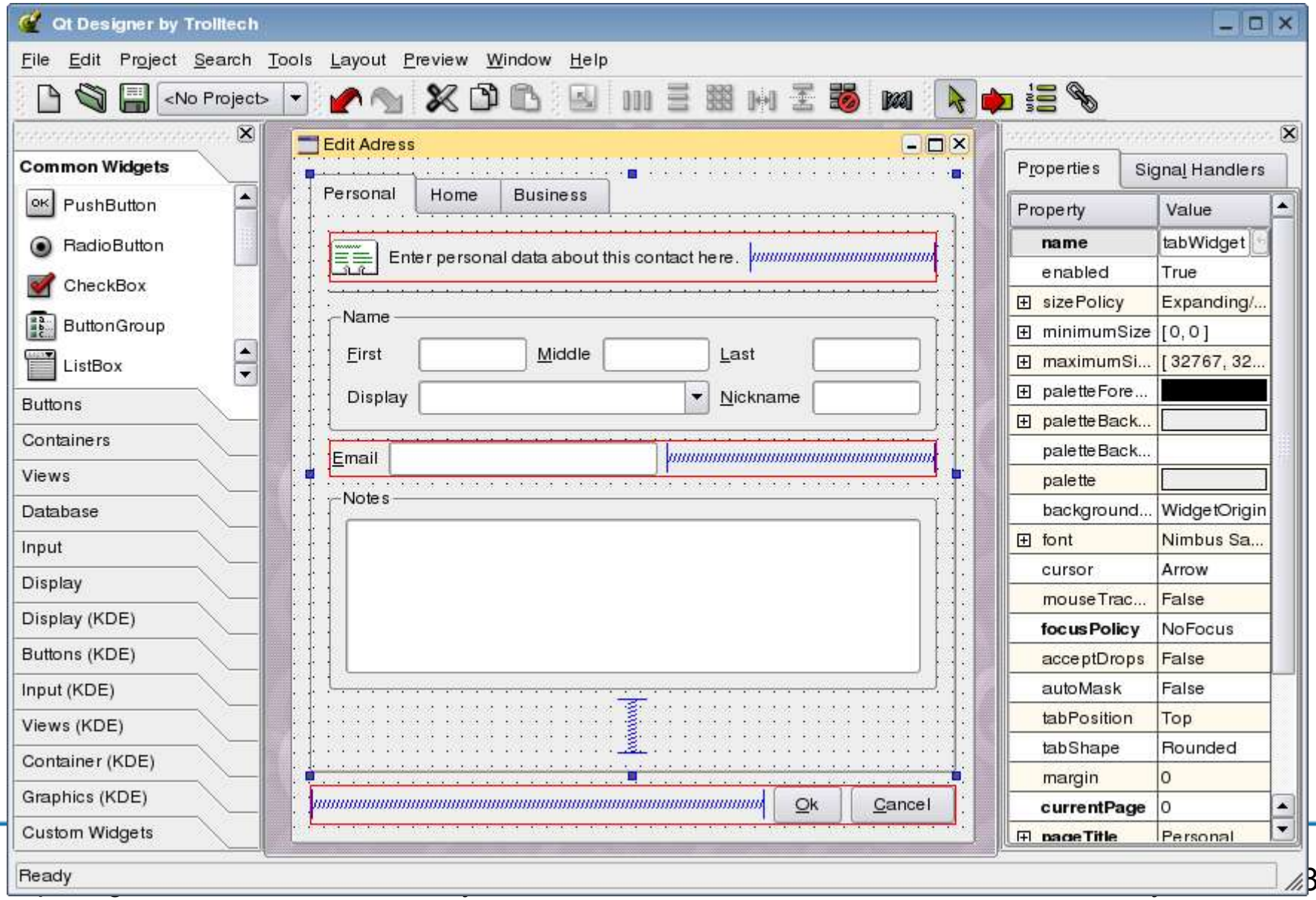
- Verbindung der Event-Loops von MFC und Qt

Schritt 2: Der Dialog

Portierung des Adress-Dialogs auf Qt

- Alternative 1:
  Import der Ressource-Datei im Designer
  - manuelle Reperatur der Dialoge
  - Einfügen zusätzlicher Funktionen:
    Layout Management, alternative Widgets
- Alternative 2:
  Neuimplementierung des Dialogs im Designer
  - Variablennamen der Widgets in Qt
    = Variablennamen der Daten in MFC

Schritt 2: Der Dialog



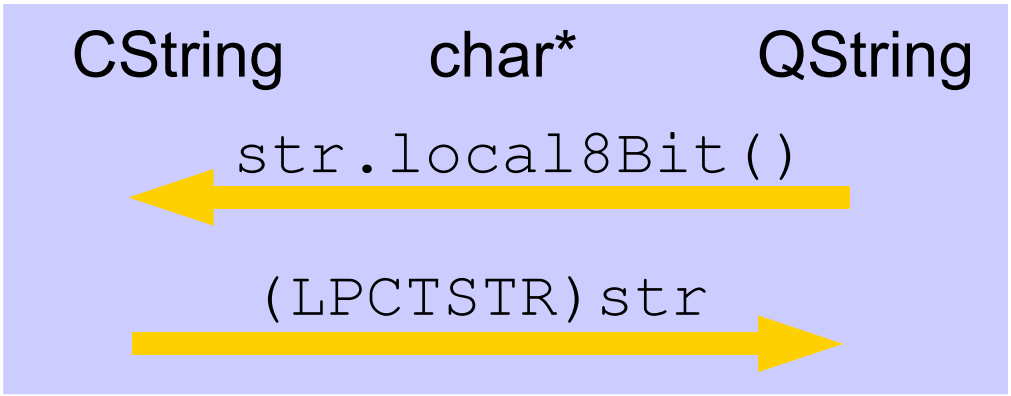
Schritt 2: Der Dialog

- Einbinden des neuen Dialogs

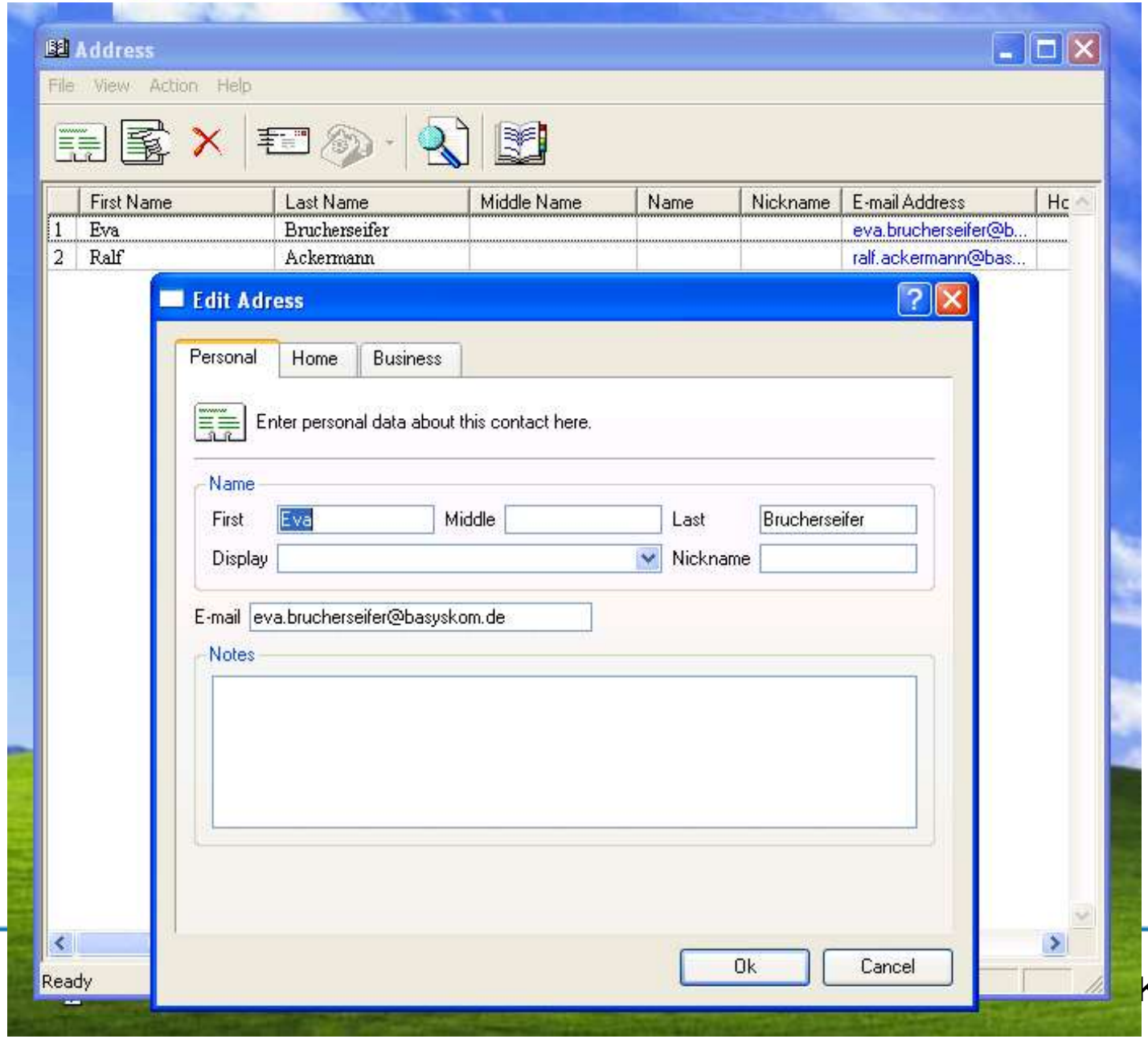
```
win.= 0, true );
```

Schritt 3: Übertragung der Daten

- Übertragung der Funktionen
  - Anzeige des vollständigen Namens
  - Aufruf der persönlichen Webseite aus dem Dialog heraus
- Verbinden der Funktionen mit den Widgets
  - im Designer
  - Signals und Slots
- Abgleich der Daten mit den Datenstrukturen



Schritt 3: Übertragung der Daten

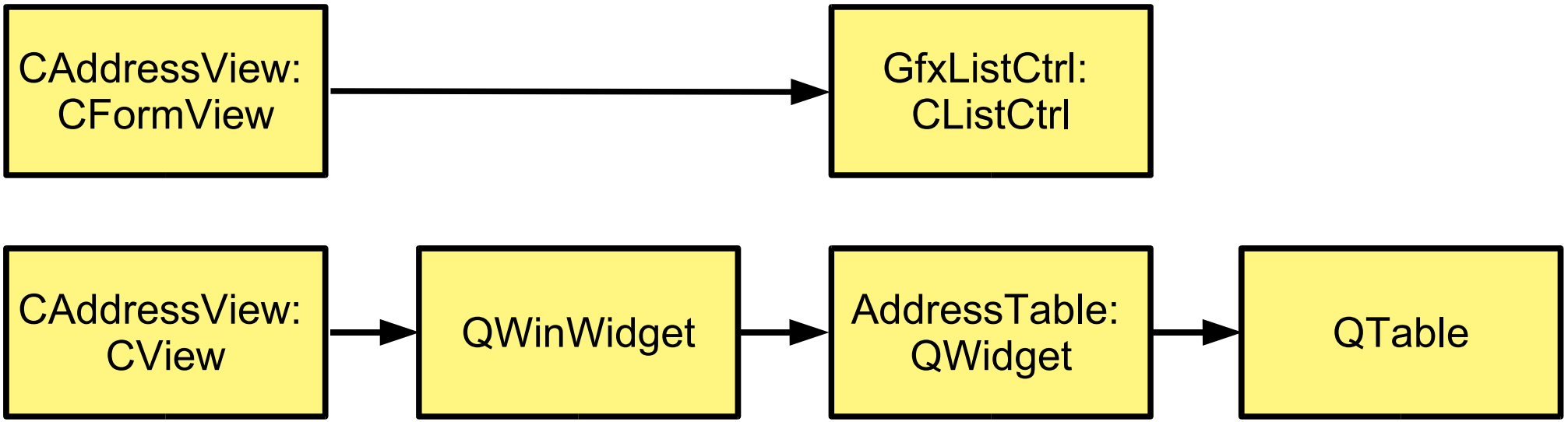


Schritt 4: Backend

- Aufbau der Datenklassen mit Qt-Klassen
  - Personendaten: CPerson `CString => QString`
  - Liste der Personen: `CArray => QPtrList`
- MFC-Dateiformat: binär
  - Funktion `void Serialize(CArchive& ar)`
  - undokumentiert
- Lösung für Qt-Programm:
  - altes Format nachbilden
  - oder Konverter in neues Format implementieren
    (gemischter MFC/Qt-Code,
    ausführbar unter Windows, evtl. über Wine)

Schritt 5: View-Widget

- Ersetzen des zentralen Widgets:
  allgemein: `CWnd` => `QWidget`
  hier: `CListCtrl` => `QTable`
  `CFormView` => `QView`



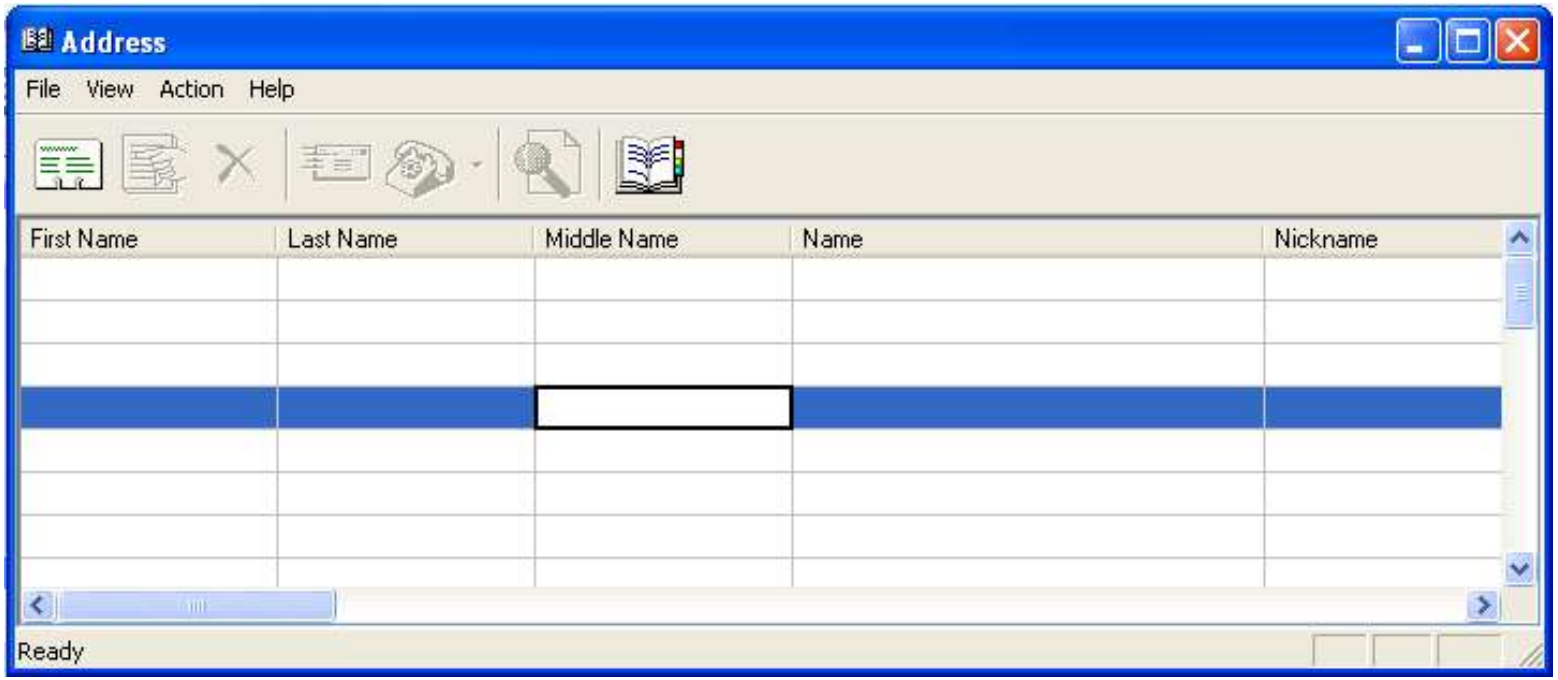
Schritt 5: View-Widget

```
void CAddressBookView::setWidget(QWidget *widget);
```

Schritt 5: View-Widget

- Anbinden des Layouts über ON_WM_SIZE()

```
void CAddressWidgetView::OnSize(UINT nType, int cx, int cy)
```



Schritt 6: QApplication

- Neuimplementierung des MainFrames
  cMainFrame => QMainWindow
- Neuimplementierung der Menüs
- Entfernen der alten View (CaddressView)
- Portierung der Programmlogik
- Entfernen des „Glue-Codes“

oder

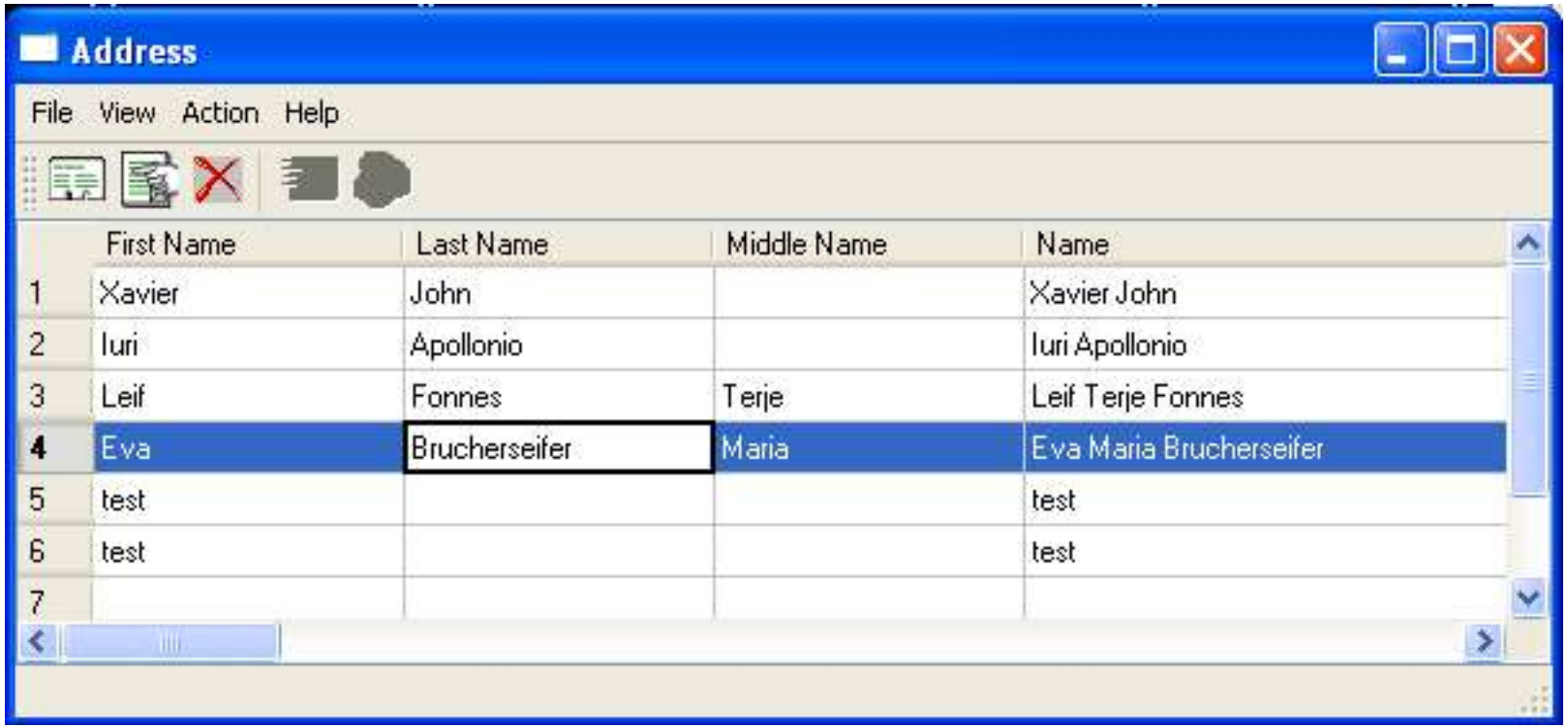
- Neuimplementierung des Kernprogramms
- Anbindung der portierten Teile

Schritt 6: QApplication

- Ersetzen der MFC-Applikation `CWinApp`

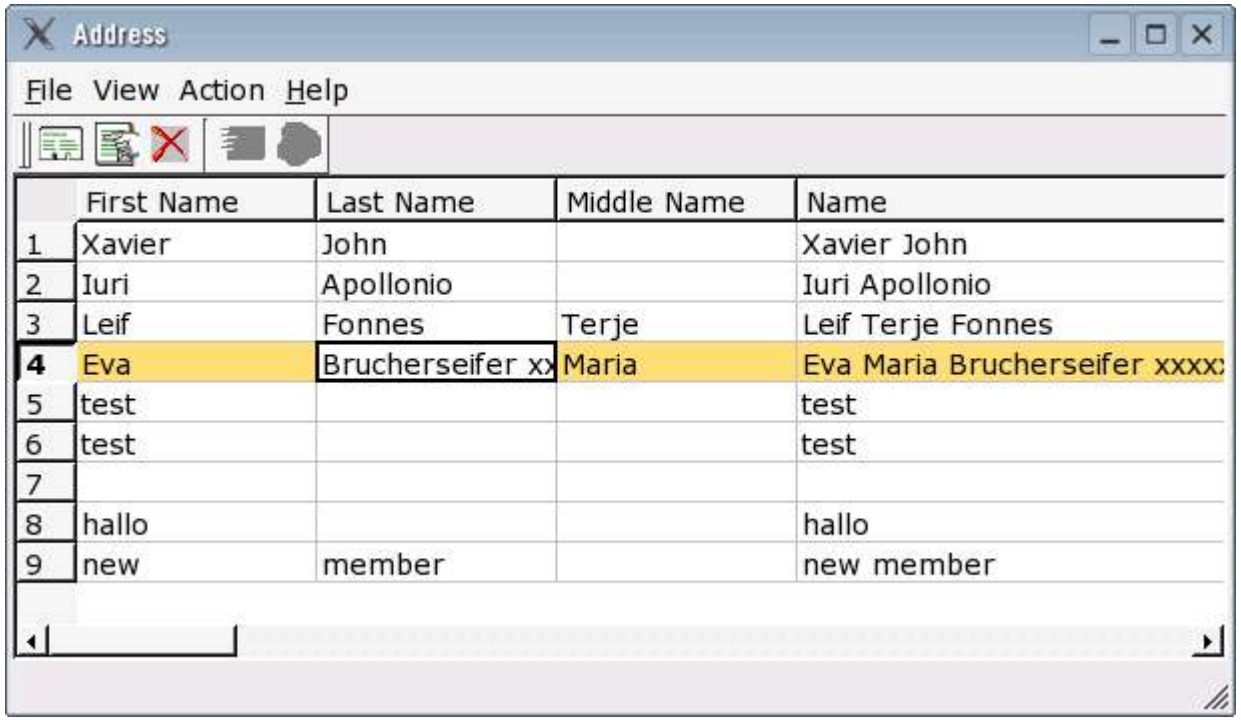
```
  app.setMainWidget(&frame);
  frame.show();
  return app.exec();
}
```

Ergebnis


Address
File View Action Help
First Name Last Name Middle Name Name
1 Xavier John Xavier John
2 Iuri Apollonio Iuri Apollonio
3 Leif Fonnes Terje Leif Terje Fonnes
4 Eva Brucherseifer Maria Eva Maria Brucherseifer
5 test test
6 test test
7

Nachbearbeitung

- Speicherung der Einstellungen mit QSettings
- ggf. Nachbearbeitung der Icons (andere Formate)
- Verwendung von Desktop-Funktionen



Bewertung

- MFC-Code und Qt-Code kann durch die Migrationsbibliothek gemischt werden
  - schrittweise Portierung
- Unterschiede in Funktion der Widgets und Klassen benötigen eine sorgfältige Implementierung
  - in den meisten Fällen: C => Q
- Automatisierung durch Skripte bedingt möglich
- ggf. Restrukturierung des Codes vor der Portierung sinnvoll

Beispiele verfügbarer Software

OpenOffice (frei)	Office	eigenes Toolkit, Java
Gimp (frei)	Grafik	Gtk
Scribus	DTP	Qt
Matlab (Mathworks)	Math./Simulation	Java+C
Eagle (CadSoft)	Platinenlayout	Qt
Parity (ParitySoftware)	ERP	Qt
Tax 2004 (Buhl)	Steuerprogramm	Emulation, Wine
Map&Route (Telekom)	Telefonbuch	Gtk
Opera (Opera)	Webbrowser	Qt
Moneyplex (Matrica)	Homebanking	Qt
Brockhaus/Duden	Lexikon	Qt

Nutzung der Zielplattform KDE

- standardisierte Dateiformate
- Transparenter Zugriff aufs Netzwerk
- einheitliche Dialoge
- zentrale Desktop-Einstellungen (Fonts, Sprache, etc)
- **Skripting**
- **Komponenten**

Skripting

- DCOP = Desktop Communication Protokol
- Verwendung
  - aus Skripten
  - aus Programmen
- Skripte/Kommandozeile:

```
dcop <id> <interface> <function> [parameter]
```

Beispiele mit KWord:

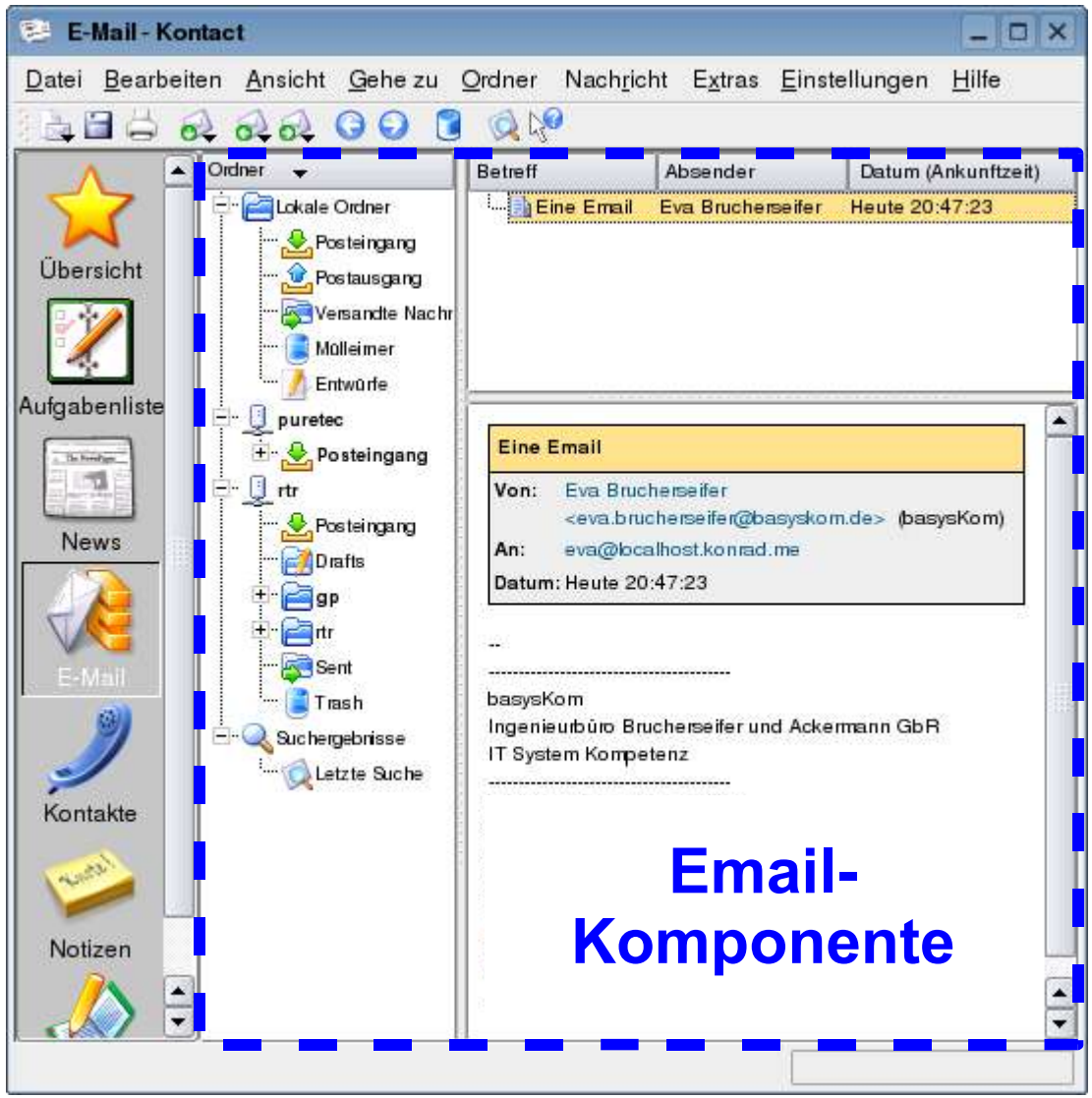
[illegible]

Skripting

- libdcop
  - C++
  - Verwendung von DCOP-Schnittstellen
  - Anbieten von Schnittstellen in eigenen Programmen
  - Teil von kdelibs
  - Abhängigkeiten: X, Qt
  - stabile API
- Bindings:
  - Python
  - Perl
  - Java
  - C

Komponenten

- Dialoge und Widgets
  - File/Print-Dialoge
- Komponenten
  - Web
  - Editor
  - Email
  - Adressen
  - Termine
  - Office



Einbindung von Komponenten

- libkparts
  - zur Einbettung der Widgets und für rudimentäre Funktionen
  - Kommunikation über DCOP
  - die meisten Komponenten greifen auf Funktionen von KApplication zurück
    => Umwandlung von QApplication in KApplication nötig
  - zur automatischen Integration der Komponentenfunktionen in das Anwendungsmenü: Verwendung von KMainWindow nötig
  - KTrader: Klasse zum Abfragen der KDE-Datenbank nach passenden Komponenten
- Alternative: XEmbed
  - Out-of-process-Lösung
  - Embedded einen kompletten Fensterinhalt in ein QT-Programm

Zusammenfassung

- plattformunabhängiger Code für
  - weiterhin bestehenden Windows-Markt
  - wachsenden Linux-Markt
- plattformabhängiger Code für Anbindung an die Desktop-Plattform
- für native Programme und einfache Portierung: Qt Toolkit

Kontakt:eva.brucherseifer@basyskom.de