Plattformunabhängige Softwareentwicklung

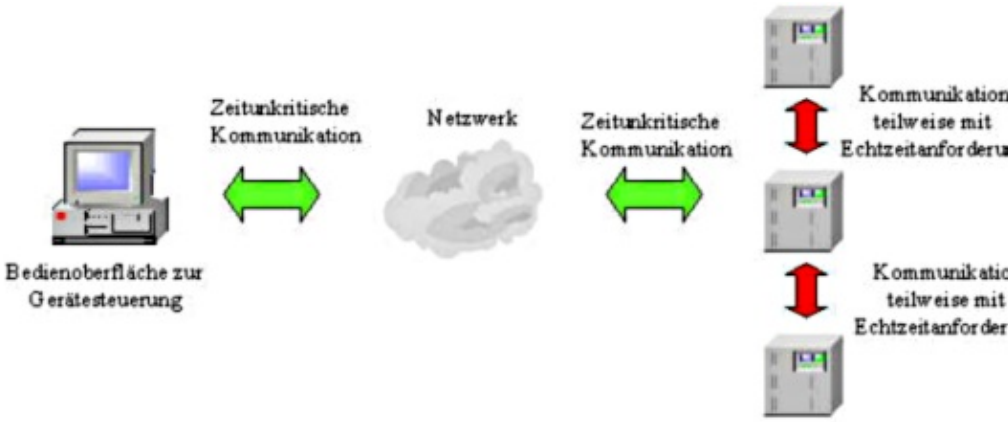
von Michael Tschater
<tschater/at/web.de>

Über den Autor:

Michael beschäftigt sich hauptsächlich mit hardwarenaher Software−Entwicklung (Firmware). Bei seinem aktuellen Projekt muss zusätzlich eine Strategieentscheidung über eine Entwicklungsumgebung getroffen werden, mit der zukünftig ein Front−End zum Ansteuern seiner Firmware programmiert werden soll.

Zusammenfassung:

Nahezu alle Geräte in der Industrie lassen sich heutzutage über ein Netzwerk steuern. I
Bedienoberfläche läuft dabei auf Standard−Hardware und arbeitet als reiner Client, der
zeitunkritisch Daten sendet (z.B. Initialisierungsparameter) und empfängt (z.B.
Meßergebnisse). Im folgenden Schaubild ist die hier angesprochene Kommunikation g
gekennzeichnet.



Bei Softwareprojekten stellt sich vielfach die Frage, welche Betriebssysteme unterstütz
werden sollen. Während der Leser dieses Artikels wohl in Richtung Linux tendieren
dürfte, werden aber auch andere Betriebssysteme (hauptsächlich Windows) gefordert.
Prinzipiell spielt das verwendete Betriebssystem für den genannten Anwendungsfall ke
große Rolle, es muß lediglich gewährleistet sein, daß der Benutzer intuitiv in gewohnte
Weise zu den gewünschten Ergebnissen kommt.

Der folgende Artikel soll aufzeigen, dass es keine Entscheidung für eine einzelne
Plattform geben muss, sondern dass es ohne weiteres möglich ist Software zu schreiben
die sich auf mehreren Betriebssystemen übersetzen läßt. Als Zielplattform beschränkt s
der Artikel auf Standard PCs mit Linux und Windows. Es sollte auch möglich sein die
erstellten Anwendungen auf einem Mac unter MacOSX zu übersetzen, dies kann aber a

Mangel an Hardware nicht untersucht werden.

Einleitung

Bei betriebssystemunabhängigen Bibliotheken unterscheidet man zwei Ansätze zum Darstellen von Steuerelementen in Dialogen:

1. Native Bibliotheken: Für die Darstellung von Elementen werden die entsprechenden Routinen des Betriebssystems genutzt. Dadurch ist gewährleistet, dass alle Steuerelemente aussehen wie in den Standard−Applikation dieses Betriebssystems. Eine native Bibliothek stellt Steuerelemente unter Linux anders dar als unter Windows 2000 oder Windows XP.
2. Die zweite Möglichkeit ist ein entsprechendes Look&Feel selbst zu programmieren, d.h. sämtliche Steuerelemente werden von der Bibliothek gezeichnet und sehen unter allen Betriebssystemen gleich aus.

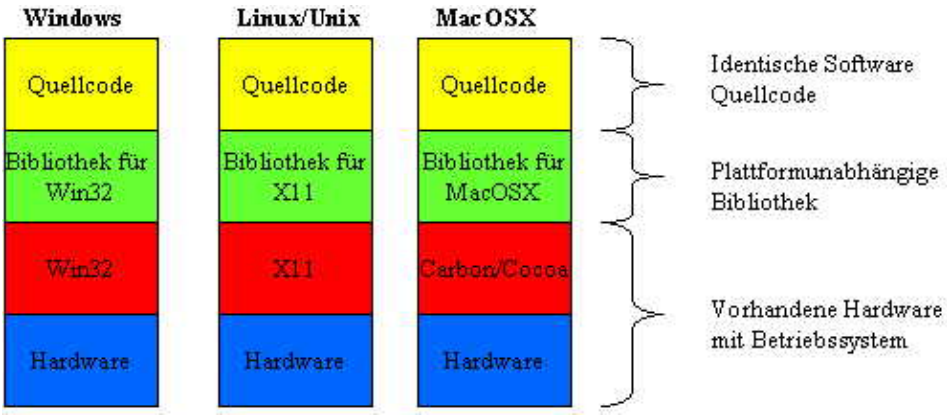
Neben den technischen Merkmalen der Bibliotheken spielen in der Praxis noch weitere Faktoren eine Rolle, die ebenfalls verglichen werden sollen:

- Entwicklungsungebung: Eine integrierte Entwicklungsumgebung (z.B. mit GUI Builder, Makefile Generator) vereinfacht die Softwareentwicklung.
- Dokumentation und Support: Bei auftretenden Problemen ist schnelle Hilfe nötig.
- Kosten: Während die meisten Bibliotheken für private Anwendungen frei verfügbar sind, fallen beim gewerblichen Einsatz teilweise Kosten an. Bei Grundsatzentscheidungen zu Softwareprojekten gilt es solche Kosten vor den Entscheidungsträgern zu rechtfertigen.
- Tatsächlicher Aufwand beim Portieren zwischen den Systemen.

Im konkreten Fall wird auf einen weiteren Punkt Wert gelegt, der jedoch nicht für alle Projekte zutreffen wird:

- Die erzeugte Software soll native Steuerelemente verwenden um sich nahtlos in die jeweilige Systemarchitektur einzugliedern. Der Benutzer soll keine Unterschiede zwischen dieser Software und der existierenden Software auf dem System erkennen können.

Stellt man die Bibliotheken in einem Schichtenmodell dar, ergibt sich folgendes Bild:



Programmiersprachen

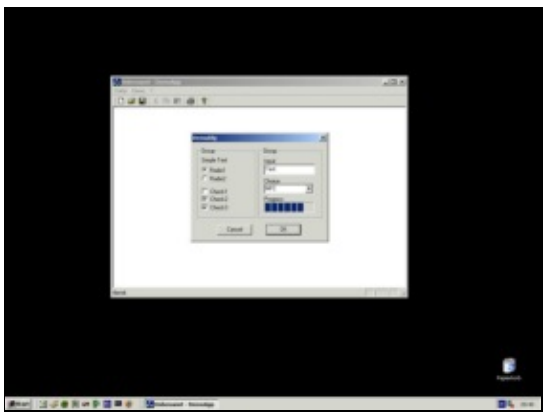
Das erste Auswahlkriterium ist die Programmiersprache. Hier bieten sich bereits mehrere Möglichkeiten auf die im Anchluss konkret eingegangen wird:

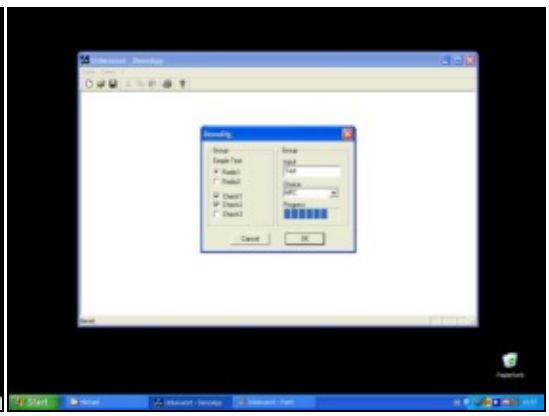
1. C/C++ Bibliotheken.
2. Java
3. Kylix
4. Smalltalk
5. Mozilla

Die Alternativen zu C und C++ werden etwas ausführlicher angesprochen, da sie auch unter Software–Entwicklern weniger bekannt sind.

Eine Beispielanwendung

Um die einzelnen Softwarepakete praktisch vergleichen zu können wird mit allen Bibliotheken eine Beispielanwendung erzeugt. Die implementierte Anwendung verfügt über keinerlei Funktionalität, zeigt aber die wichtigsten Steuerelemente. Für die Windows Seite wird zudem eine reine Windows Software (Visual C++ 6.0, MFC Klassenbibliothek) erzeugt, an der sich die anderen Pakete bezüglich des Look&Feels messen müssen. Als Linux Distributionen kommen RedHat Fedora Core 2 und Debian 3.0 zum Einsatz.





Windows 2000 und Windows XP Screenshot (Quellcode für Visual C++ hier).

C/C++ Bibliotheken

Trolltech Qt

Qt ist eine Klassenbibliothek der norwegischen Firma Trolltech für die plattformübergreifende Programmierung unter C++. Der Linux Windows Manager KDE basiert auf dem Qt Paket. Ursprünglich wurde Qt unter einer Lizenz vertrieben, die für viele Linuxanwender nicht akzeptabel war. Aus diesem Grund wurde die GTK+ Bibliothek entwickelt, die Grundlage für den Gnome Windows Manager ist. Inzwischen ist sowohl die Linux− als auch die MacOS−Version unter der GPL inklusive Quelltexten verfügbar, Qt für Windows wird nach wie vor kommerziell vertrieben. Zur Evaluaierung kann eine zeitlich befristete Testversion von der Webseite heruntergeladen werden, dabei wird unterschieden zwischen Evaluierungen für anschließende kommerzielle Nutzung und Evaluierung für akademische Zwecke. Im folgenden wird auf die kommerzielle Evaluierungsversion eingegangen, die eine Registrierung erfordert.

Neben den Versionen für Windows, Linux (Unix) und Mac ist eine embedded Version verfügbar, die auf embedded Linux Varianten läuft und eine schlankere Fensterverwaltung bietet.

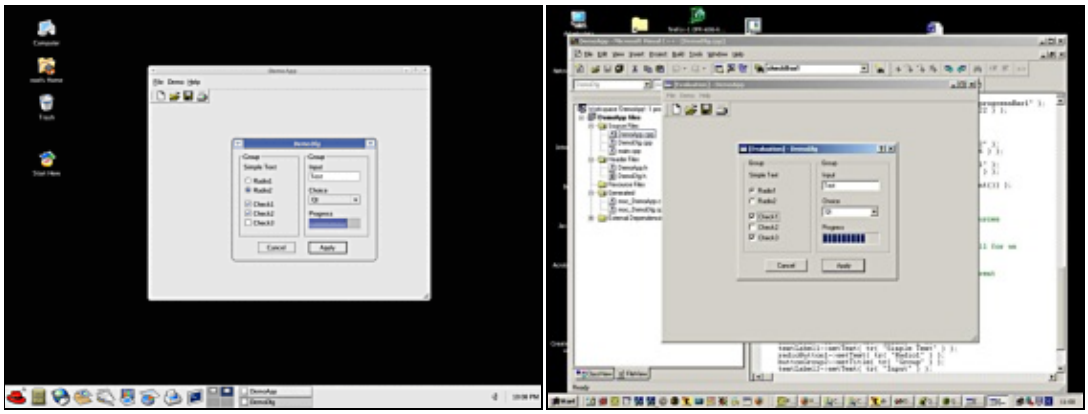
Unter Linux verläuft die Installation wie erwartet völlig problemlos. Im Programmpaket enthalten ist der GUI Generator Qt Designer. Die enthaltene umfangreiche Dokumentation gefällt durch eine detaillierte Beschreibung von Beispielprojekten, einem Kapitel zum Schnelleinstieg, sowie einer Klassenübersicht. Der Qt Designer ist leider keine komplette Entwicklungsumgebung, für die Code−Generierung muss eine Shell bemüht werde. Als Ouput des GUI Generators erhält man eine XML Beschreibung der erstellten grafischen Oberfläche. Aus dieser Beschreibung wird über das Qt−Tool qmake ein gültige Makefile erzeugt. Das Makefile erzeugt aus der Oberflächen−Beschreibung zunächst C++ Quellcode−Dateien (Qt−Tool: uic) und ruft anschließend den Meta Object Compiler (Qt−Tool: moc) auf, der Qt spezifische Spracherweiterungen in C++ Quellcode umsetzt (z.B. Signal−Slot−Mechanismus). Erst danach kann ein Standard make aufgerufen werden.

Möchte man die Source−Files von Hand generieren ist folgende Sequenz nötig (Ausgangsbasis ist MyDialog.ui):

- uic MyDialog.ui > MyDialog.h
- uic −impl MyDialog.h MyDialog.ui > MyDialog.cpp
- moc −o moc_MyDialog.cpp MyDialog.h

Bei der Installation unter Windows benötigt das Toolkit einen Benutzer− und Firmennamen sowie eine Seriennummer. Diese Daten werden von Trolltech per Email nach der Registrierung mitgeteilt. Wider Erwarten läuft die Installation auf den Testsystemen nicht reibungslos ab. Eine Windows 2000 Installation mit Visual C++ 6.0 Entwicklungsumgebung wird zwar korrekt erkannt, dem Wizard gelingt es aber nicht, die Beispielprojekte zu übersetzen und die Installation muss abgebrochen werden. Auf einem zweiten System mit Windows 2000 wird eine Visual C++ 5.0 Umgebung erst gar nicht erkannt. In beiden Fällen muss also Hand angelegt werden um zu einem lauffähigen System zu gelangen. Wie bei der Linux Version wird auch unter Windows die Entwicklungsumgebung Qt Designer mitgeliefert. Die Dokumentation ist offensichtlich die gleiche wie unter Linux. Die Windows spezifischen Informationen gehen etwas unter. So muss der Visual

C++ Programmierer intensiv suchen, um die wichtigen Zeilen "For Visual Studio users, qmake can also generate '.dsp' files, for example: qmake –t vcapp –o hello.dsp hello.pro" zu finden.



Linux und Windows 2000 Screenshot (Quellcode für QtDesigner hier).

Übersicht über Qt

Name:	Trolltech Qt
Version:	3.3.2
Betriebssysteme:	Linux, Win32, MacOS, Solaris, IRIX, AIX, HP–UX
Programmiersprache:	C++
Lizenz:	GPL oder proprietäre Lizenz (kommerziell)
Vorteile:	• Basisbibliothek für den KDE Windows Manager unter Linux • Installationspakete in allen Linux Standard–Distributionen vorhanden (Installation sehr einfach) • generische Controls unter Windows • Mächtige Entwicklungsumgebung(en) • bewährt • Migrationsunterstützung für Win32 MFC Anwendungen (Qt/MFC Migration Framework) ermöglicht das schrittweise Umsetzen von MFC Quellcode.
Nachteile:	• evtl. Lizenzkosten (teuer) • Evaluierungssoftware läßt sich unter Windows nicht fehlerfrei installieren
Entwicklungsumgebung:	z.B. QtDesigner, KDevelop
WWW:	http://www.trolltech.com
Dokumentation:	Manuals, Tutorials, Mailing Listen z.B. http://doc.trolltech.com/3.3/index.html
Referenzprojekte:	• KDE Desktop (Default z.B. bei SuSE) • Opera Browser • Photoshop Album

Verbreitung:	sehr große Verbreitung

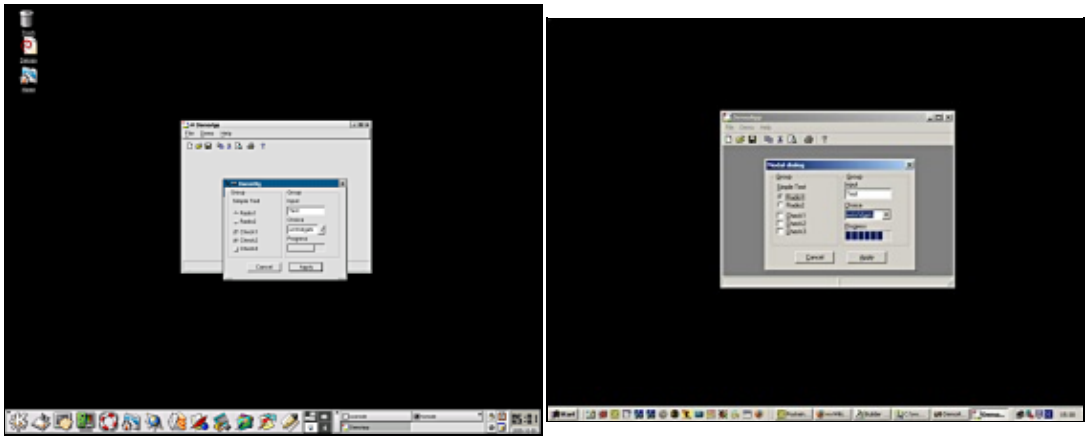
wxWidgets

Seit mittlerweilen 12 Jahren ist das wxWidgets Toolkit verfügbar, jedoch erst vor wenigen Monaten bekam das Paket seinen heutigen Namen. Der bis dahin verwendete Name wxWindows wurde nach "Gesprächen" mit Microsoft aufgegeben. wxWidgets verfügt über eine riesige Sammlung an Klassen für alle Bereiche. Neben der mächtigen Dokumentation sind sehr viele Beispielanwendungen verfügbar. Die Liste der Referenzanwendungen macht deutlich, dass es hier um ein ausgereiftes Softwarepaket handelt.

Die Programmierung erfolgt in C++ und ist ähnlich der Visual C++ Programmierung unter Windows. Ein Umstieg dürfte hier die geringsten Probleme verursachen.

Nach der Installation der Version wxWindows2.4.2 unter RedHat Fedora Core 2 treten beim Übersetzen der Beispielprogramme Fehler auf. Die Ursache der Fehler liegt in GTK+ Aufrufen, die in der von RedHat gepatchten Version von GTK+ privat und somit nicht zugelassen sind. Hier zeigt sich, das die Dynamik der Open Source Gemeinde durchaus auch Probleme verursachen kann. Die Probleme sollten aber nicht überbewertet werden. Mit der Installation einer Standard GTK+ Bibliothek dürfte alles glatt laufen. Für die weitere Untersuchung im Rahmen dieses Artikel wird stattdessen auf die Debian Distribution Version 3.0 gewechselt. Hier funktioniert alles auf Anhieb.

Ebenfalls problemlos verläuft die Installation unter Windows. Das Generieren der Bibliotheken und Beispielprogamme funktioniert mustergültig.



Linux und Windows 2000 Screenshot (Quellcode hier).

Übersicht über wxWidgets

Name:	wxWidgets
Version:	2.4.2
Betriebssysteme:	Linux, Win32, embedded Devices

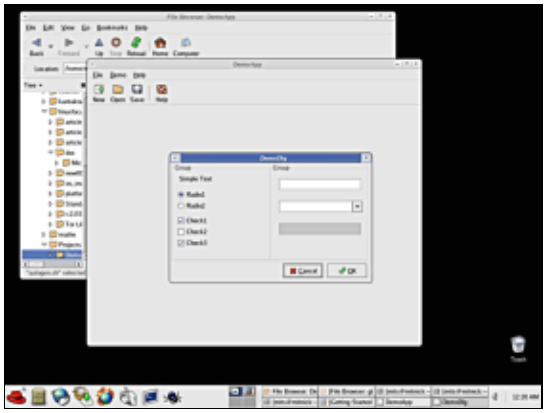
Programmiersprache:	C++
Lizenz:	LGPL
Vorteile:	• einfache Handhabung (viele Beispiele). • sehr gute Dokumentation. •
Nachteile:	• Probleme bei der Kombination Fedora Core 2 – wxWindows2.4.2
Entwicklungsumgebung:	
WWW:	http://www.wxwidgets.org
Dokumentation:	Manuals, Tutorials, Mailing Listen, Wiki z.B. http://wiki.wxwidgets.org
Referenzprojekte:	AOL Communicator
Verbreitung:	geringe Verbreitung

GTK+ (mit gtkmm)

Die Abkürzung GTK steht für "The GIMP Toolkit". Die beiden bekanntesten Projekte sind der Gnome Windows Manager, der in allen Standard Linux Distributionen enthalten ist, und die professionelle Grafikanwendung GIMP. Gnome ist neben KDE (siehe Qt) die zweite große Desktop−Umgebung unter Linux. Sie wird von vielen Distributionen als Standard−Umgebung installiert. Mit Einführung der Version 2 von GTK+ ist das Look&Feel wesentlich verbessert worden.

Eine Besonderheit von GTK+ ist, dass es komplett in C implementiert wurde. Konsequenterweise erzeugt der GUI−Builder glade2 ebenfalls C−Code. Durch Verwendung von gtkmm (früher GTK−−) läßt sich aber ebenfalls in C++ programmieren.

Ganz im Gegensatz zum professionellen Auftreten von GTK+ für Linux scheint es mit 'GTK+ for Win32' nicht weit her zu sein. Klickt man auf der GTK+−Hauptseite auf diesen Link erscheint sofort die Warnung: "**The program(s) might crash unexpectedly or behave otherwise strangely.** (But of course, so do many commercial programs on Windows.) The stability seems to depend a lot on the machine, display drivers, other software installed, and whatnot." (Stand 2004−09−06). Der wagemutige Software−Entwickler klickt natürlich dennoch auf die Download Seite und bekommt eine lange Liste an einzelnen Softwarekomponenten zum Download angeboten. Ein zusammenhängendes Paket sucht man vergebens. Stattdessen liest man die Anweisung, eine Reihe von Softwarekomponenten zu installieren und gegebenfalls, wenn bestimmte Sachen fehlen, die Download−Seite eben noch einmal zu besuchen. Dies passt zur Aussage der 'GTK+ for Windows' − Webseite: "You are expected to be pretty experienced to be able to use GTK+ in your own programs. This isn't Visual Basic.". Nach einer Installation der Basiskomponeneten und einem erfolglosen Versuch eine Beispielanwendung zu starten, dürfte den meisten Entwicklern allmählich die Lust auf eine tiefere Einarbeitung vergangen sein. Die völlig unprofessionelle Aufmachung der 'GTK+ for Win32' Einzelkomponenten (von einem Paket kann keine Rede sein) disqualifiziert das Software−Paket und jeglicher (professionelle) Einsatz ist ausgeschlossen.



GTK+ Screenshot für Linux (Quellcode für glade2 hier)

Übersicht über GTK+

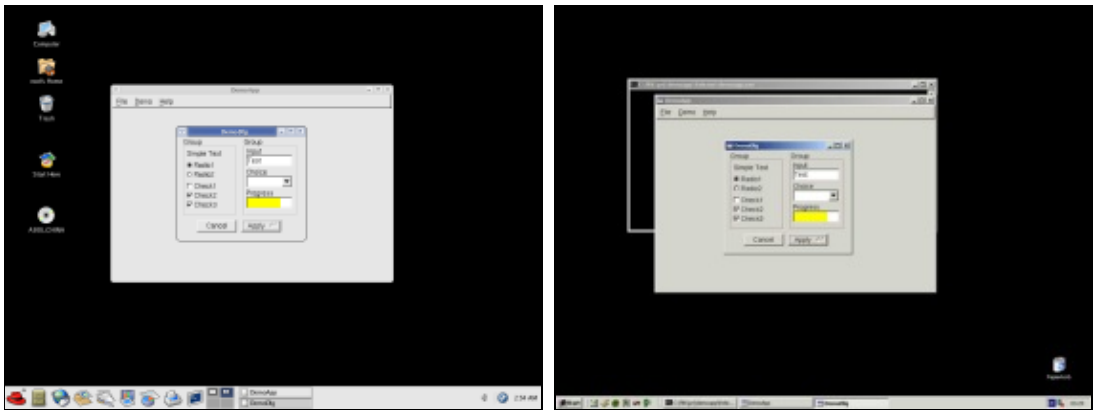
Name:	GTK+ – The GIMP Toolkit
Betriebssysteme:	Linux, Win32
Programmiersprache:	C (C++ mit gtkmm)
Lizenz:	LGPL
Vorteile:	• Basisbibliothek für den Gnome Windows Manager unter Linux • Installationspakete in allen Standard–Distributionen vorhanden (Installation sehr einfach) • generische Controls unter Windows • bewährt (unter Linux)
Nachteile:	• Win32–Implementierung ist unhandlich läuft nicht stabil (Stand 09–2004)
Entwicklungsumgebung:	z.B. glade2 (GUI Builder), Anjuta
WWW:	http://www.gtk.org
Dokumentation:	Manuals, Tutorials, Mailing Listen z.B. http://developer.gnome.org/doc/API/2.0/gtk/index.html
Referenzprojekte:	• Gnome Desktop • GIMP • Gnumeric
Verbreitung:	Linux: sehr große Verbreitung, Windows: sehr geringe Verbreitung

FLTK

Beim FLTK Toolkit (Fast, Light Tool Kit) handelt es sich um ein weitestgehend unbekanntes Paket, dass als Nachfolger von XForms implementiert wurde. Auf der Web−Seite bekommt man die kompletten Sourcen zum Download angeboten. Die Größe von 2.3MB (.tar.gz, Linux) bzw. 3MB (.zip, Win32) macht dem Namen alle Ehre. Unter Linux erfolgt die Installation problemlos: auspacken und "make" aufrufen, fertig. Danach stehen dem Benutzer die Bibliotheken, Beispielanwendungen, der GUI−Builder "fluid" und ein Programmierhandbuch zur Verfügung. Dass bei dieser Größe die Anzahl der zur Verfügung gestellten Klassen geringer ist als bei den Schwergewichten Qt und wxWindows dürfte klar sein. Die enthaltenen Klassen decken den GUI−Bereich ab, d.h. Fenster, Menüs, Controls, OpenGL und Darstellung von Bildern. Klassen für Netzwerk−Kommunikation und ähnliches sind nicht vorhanden.

Unter Windows läuft die Installation nicht ganz so rund. Bei Verwendung der Visual C++ Entwicklungsumgebung muss lediglich ein Hauptprojekt übersetzt werden, dabei kommt es aber zu Problemen mit den Grafikbibliotheken. Der Einfachheit halber können diese in der Konfigurationsdatei config.h auskommentiert werden.
Eine weitere Besonderheit der Windows−Variante ist, dass sich beim Starten eines Projekts stets auch ein DOS−Fenster öffnet, wenn die FLTK−Bibliothek in der DEBUG Variante erstellt wurde. Dies ist kein Bug, sondern ein Feature: Auf diese Art wird gewährleistet, dass Anwendungen die von der Konsole gestartet werden auf stderr und stdout schreiben können.

Insgesamt gesehen macht das FLTK Toolkit einen sehr durchdachten Eindruck. Die Dokumentation hebt insbesondere die geringe Größe der Executables (80kB für ein "hello world") und schlanke schnelle 2D und 3D Grafik (OpenGL/Mesa) hervor. Desweiteren wird auf eine gute Portierbarkeit hingewiesen.



Linux und Windows 2000 Screenshot (Quellcode hier).

Übersicht über FLTK

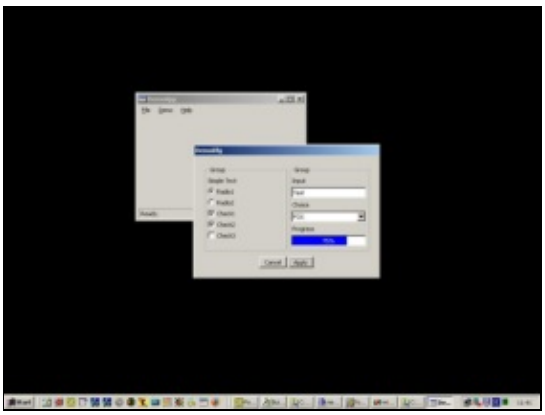
Name:	Fast Light Tool Kit
Version:	1.1.5rc3
Betriebssysteme:	Linux, Win32, MacOS
Programmiersprache:	C++

Lizenz:	LGPL
Vorteile:	• sehr schlanke Bibliothek • Quellcode inklusive Dokumentation und Entwicklungsumgebung "fluid" • gute OpenGL Unterstützung (wurde nicht untersucht) • generische Controls unter Windows
Nachteile:	• Installation unter Win32 (Visual C++) nicht fehlerfrei • fluid−Entwicklungsumgebung läuft nicht stabil unter Windows
Entwicklungsumgebung:	z.B. fluid (GUI Builder), Eclipse
WWW:	http://www.fltk.org, Download: http://freshmeat.net/projects/fltk/
Dokumentation:	Manuals, Tutorials, Mailing Listen z.B. http://www.fltk.org
Referenzprojekte:	• http://vtkfltk.sourceforge.net/
Verbreitung:	geringe Verbreitung; auch unter Software−Entwicklern weitestgehend unbekannt

FOX Toolkit

Das Fox Toolkit bezeichnet sich selbst als eines der schnellsten verfügbaren Toolkits. Es verfügt über eine Vielzahl von Steuerelementen und eine OpenGL Anbindung.

Die obligatorischen Test−Installationen unter Windows und Linux verlaufen problemlos. Neben Beispielprojekten ist eine ausführliche Dokumentation verfügbar. Leider fehlt bei der getesteten Version die Klassenübersicht, die aber online eingesehen werden kann. Bei der weiteren Untersuchung leistet sich das Toolkit keine Schwächen.



Windows 2000 Screenshot (Quellcode hier)

Übersicht über FOX

Name:	FOX Toolkit
Version:	1.2.9
Betriebssysteme:	Linux, Win32
Programmiersprache:	C++
Lizenz:	LGPL
Vorteile:	• gute Dokumentation
Nachteile:	•
Entwicklungsumgebung:	
WWW:	http://www.fox−toolkit.org
Dokumentation:	Manuals, Tutorials, Mailing Listen
Referenzprojekte:	• X File Explorer (Xfe)
Verbreitung:	geringe Verbreitung

Weitere Möglichkeiten

Neben den angesprochenen Bibliotheken sollen der Vollständigkeit halber noch weitere Projekte genannt werden, auf die aber nicht eingegangen wird:

- GNUstep [http://www.gnustep.org/]: unter Windows nur bedingt einsetzbar
- Visual Component Framework [http://vcf.sourceforge.net/]: keine komplette Linux−Version verfügbar

JAVA

Im Jahre 1995 stellte die Firma Sun eine neue Programmiersprache vor. Neben normalen Desktop−Computern war Java für Industriegeräte (Kaffemaschinen, Toaster, ...) vorgesehen. Der große Durchbruch gelang aber zunächst über Internet−Anwendungen (Applets) in Verbindung mit Web−Browsern. In der Zwischenzeit wird Java aber auch für Standalone Anwendungen verwendet, für die es sich durch verschiedenen Eigenschaften sehr gut eignet.

Im folgenden sollen die wichtigsten Eigenschaften von Java aufgezählt und kurz erklärt werden.

Plattformunabhängigkeit

Java ist plattformunabhängig. Java Anwendungen bestehen aus einem Bytecode, der von einer virtuellen Maschine interpretiert wird. Dadurch sind die Anwendungen auf jeder Hardware lauffähige für die eine passende virtuelle Maschine existiert. Das Interpretieren durch die virtuelle Maschine bedeutet eine geringere Abarbeitungsgeschwindigkeit gegenüber compilierter Software. Um diesen Nachteil auszugleichen sind mittlerweile Verbesserungen wie beispielsweise Just−In−Time−Compilierung (JIT) entwickelt worden, die zur Laufzeit Programmanweisungen der virtuellen Maschine in Anweisungen für die physaklische Maschine übersetzt. Als Ergebnis erhält man hier ein angepasstes Programm im Speicher, dass ohne Interpretation schnell ausgeführt werden kann. Die Hotspot−Technologie führt zusätzlich eine Analyse des Laufzeitverhaltens zur weiteren Optimierung durch.

Objektorientierung

Java ist objektorientiert. Bei der Objektorientierung ließen sich die Entwickler der Sprache von Smalltalk inspirieren. Vermutlich aus Performance−Gründen sind aber dennoch primitive Datentypen vorhanden, die nicht als Objekte verwaltet werden.

Sprachsyntax

Die Sprachsyntax ist ähnlich wie bei C und C++, jedoch wurden fehlerträchtige Inkonsistenzen nicht übernommen. Eine Grundsatz beim Entwicklen der Sprache war, dass Java die besten Konzepte der existierenden Programmiersprachen vereinen soll.

Einige Beispiele sind:

- Kein Präprozessor: Ein Präprozessor und Header−Dateien sind nicht mehr nötig, da alle Informationen direkt aus den Klassendateien gelesen werden.
- Zeiger: Java kennt keine Zeiger, sondern verwendet statt dessen Referenzen. Eine Referenz repräsentiert ein Objekt.
- Garbage−Collector: Um Probleme mit dem Anlegen und Löschen von Objekten zu umgehen wird die Objektverwaltung von der Java−Laufzeitumgebung übernommen. Beim Verlassen des Wirkungsbereichs werden Objekte automatisch gelöscht. Nicht freigegebene Objekte bzw. Speicherbereiche sowie falsche Destruktoren werden durch diese Technik verhindert.
- Ausnahmebehandlung: Im Gegensatz zur C++ Ausnahmebehandlung werden Java Exceptions wesentlich intensiver genutzt und sind oftmals obligatorisch.

Klassenbibliothek

Java verfügt über eine umfangreiche Klassenbibliothek: JFC (Java Foundation Class) zum Erstellen von Oberflächen (Durchgesetzt hat sich hier der Codename Swing).

Sicherheit

Von einem Verifier wird Java−Code zunächst auf strukturelle Korrektheit und Typsicherheit überprüft. Ein Security−Manager überwacht die Zugriffe auf die Peripherie. Jegliche Sicherheitsprobleme werden über Exceptions zur Laufzeit gemeldet.

Eignung für Projekte

Die genannten Vorteile haben Nebeneffekte, die Java nicht für alle Projekte sinnvoll erscheinen lassen. Diese Eigenschaften sind jedoch keine Fehler oder Schwächen, sondern sind bewußt nicht implementiert worden und gehören zur Sprachphilosophie.

Dazu gehören z.B.:

- plattformspezifische Peripherie−Zugriffe
- direkte Hardware−Zugriffe
- Eingriffe in das Betriebssystem

Java Development Kit (JDK)

Das Java Development Kit kann von der Sun Internetseite heruntergeladen werden und umfasst eine Basisausstattung an Applikationen, Java−Klassen und die Online−Dokumentation. Bei den Anwendungen handelt es sich um einen Compiler, einen Debugger, einen Appletviewer, sowie verschiedenen Hilfsprogrammen, die zum Erstellen und Testen von Java Anwendungen und Java Applets notwendig sind. Die Ausstattung bietet jedoch nur das Nötigste, der Compiler etwa muss von der Kommandozeile aus bedient werden. Desweiteren ist im Paket das Java Runtime Environment (JRE, enthält die virtuelle Maschine) enthalten, die zum Ausführen des Bytecodes notwendig ist. Die Dokumentation umfasst schließlich die Beschreibung der kompletten API.

JHelloWorld

Unter Benutzung des Standard JDK soll die obligatorische Hello−World Anwendung implementiert werden:

1. Schritt: Erzeugen des Quellcodes.

```
sh>vi Helloworld.java
```

```
public class HelloWorld {

  public static void main (String[] args) {


    System.out.println("Hello World!");

  }
```

```
}
```

Dateiname und Klassenname müssen übereinstimmen.

2. Schritt: Übersetzen.

```
sh>javac Helloworld.java
```

3. Schritt: Anwendung unter Verwendung der virtuellen Maschine starten.

```
sh>java Helloworld
```

JavaScript und Java

Fälschlicherweise werden zwischen JavaScript und Java oft Gemeinsamkeiten vermutet. Dies ist grundsätzlich falsch. JavaScript wurde als Skript−Sprache zur Einbettung in HTML ursprünglich von der Firma Netscape entwicklet. Es ist keine eigenständige Programmiersprache und ist vom verwendeten Browser abhängig. Der Name JavaScript ist also mehr als Marketing−Gag zu verstehen.

Normierungsversuche

Bisher sind sämtliche Normierungsversuche der Sprache Java gescheitert. Grund hierfür dürfte sein, dass Sun die alleinige Kontrolle über die Weiterentwicklung des Java−Standards nicht aus der Hand geben möchte.

Dekompilierung

Ein Wunder Punkt von Java ist die Tatsache, dass Anwendungen dekompiliert werden können. Trotz aller Sicherheitsmechanismen kann, zumindest im Augenblick, der Bytecode wieder in Source Code umgewandelt werden. Da der Java Compiler Bytecode für einen virtuellen Prozessor erstellt, müssen im Gegensatz zum klassischen Assembler−File weitere wichtige Informationen enthalten sein, die die Dekompilierung deutlich vereinfachen. Dies ist vor allem dann kritisch, wenn spezielles Wissen im Source Code steckt.

Wundersprache oder kurzfristiger Hype?

Nachdem das Java Konzept anfänglich für die ultimative Lösung für plattformunabhängige Programmierung gehalten wurde, hat sich die Euphorie wieder gelegt. Neben Versionskonflikten zwischen verschiedenen virtuellen Maschinen wird immer wieder auf die Geschwindigkeit verwiesen. Viele Firmen sind nach einer Versuchsphase wieder zur Standard−C++ Programmierung zurückgekehrt. Dies wird z.B. von den wxWidgets Entwicklern bestätigt, die von wieder steigender Anzahl von Downloads berichten.

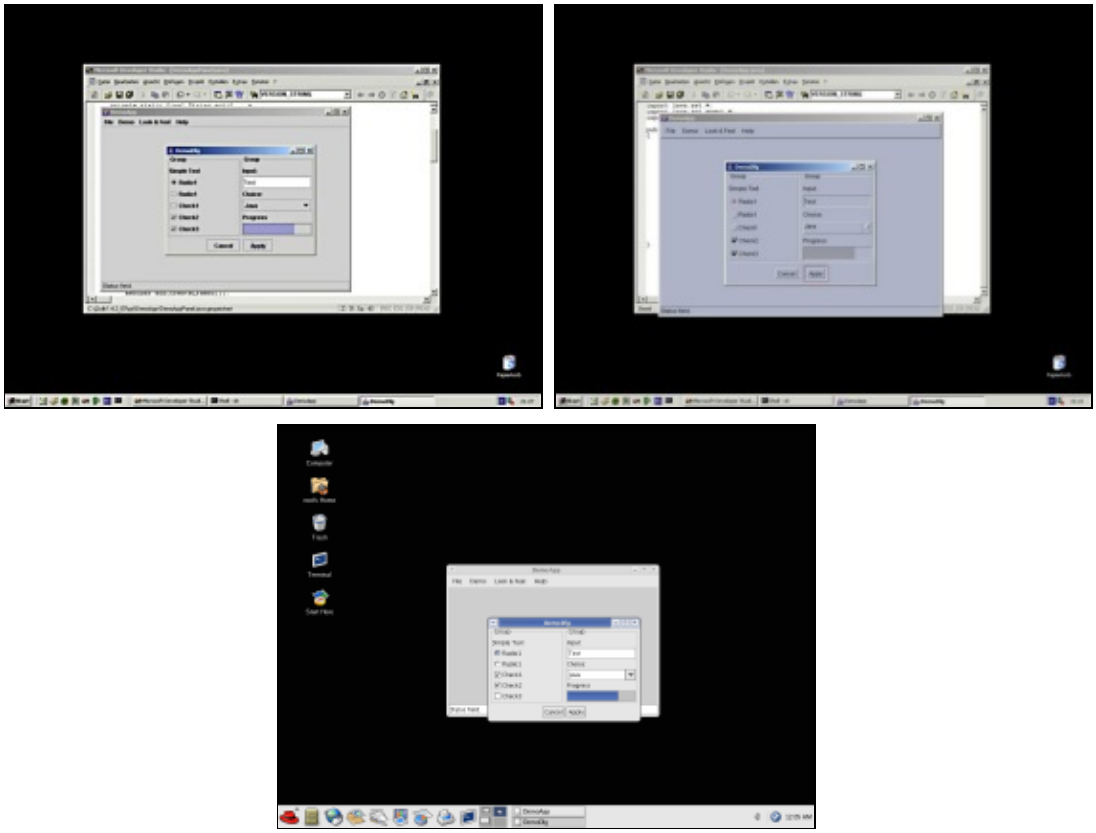
Eine interessante Webseite in diesem Zusammenhang ist:
http://www.internalmemos.com/memos/memodetails.php?memo_id=1321. Hier sprechen sich (angeblich)

Sun Mitarbeiter gegen Java aus.

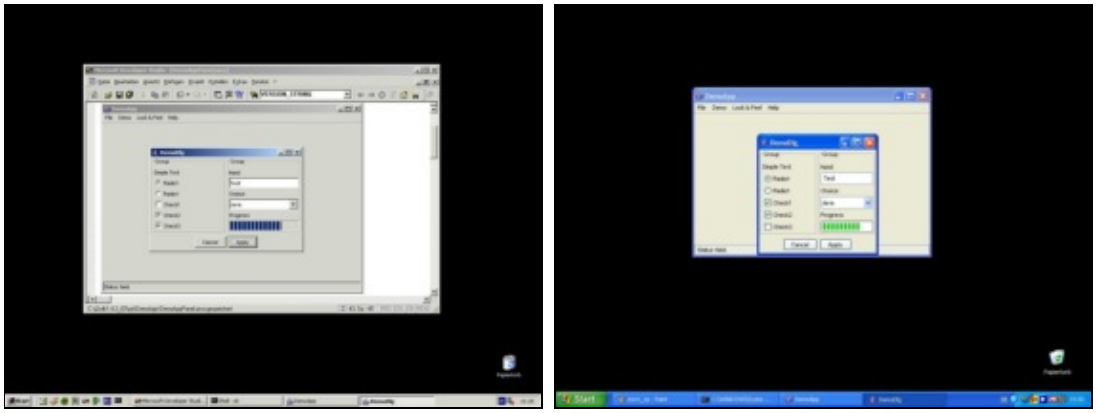
Grafische Oberflächen mit Java

Standardmäßig bietet Java 2 Möglichkeiten zur Programmierung von grafischen Oberflächen:

1. Zum einen ist eine umfangreiche Klassenbibliothek enthalten (JFC, Swing), die sämtliche Arten von Steuerelementen zur Verfügung stellt. Im Gegensatz zu den bisher vorgestellten Toolkits werden wird hier keine Betriebssystemfunktionalitäten verwendet, sondern sämtliche sogenannten Widgets werden durch Java Befehle gezeichnet. Dadurch ist es möglich das Look&Feel der Anwendung zur Laufzeit umzuschalten. Die im Anschluß folgenden Screenshots zeigen die identische Anwendung, die von dieser Möglichkeit Gebrauch macht.
2. Zum anderen gibt es die sehr rudimentären AWT Funktionen. Bei AWT fehlen komplexere Oberflächenelemente wie z.B. Bäume, weswegen es für die meisten Anwendungen unbrauchbar ist.

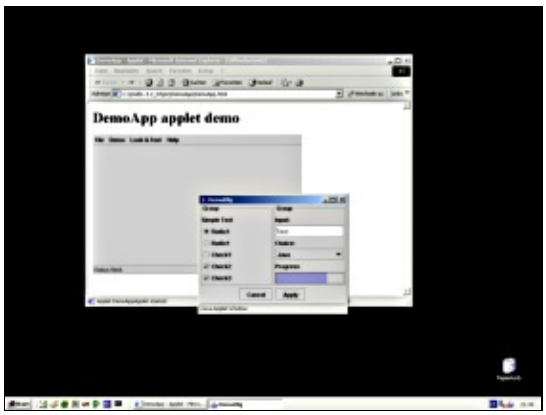


Java Screenshots im Metal–, Motif– und GTK+ Look & Feel (Quellcode hier)



Java Screenshot mit Windows Look & Feel unter Windows 2000 und Windows XP (identischer Sourcecode)

Da alle gängigen Browser Java unterstützen, können Anwendungen auch als sogenannte Applets im Browser laufen. Diese Technologie kann z.B. im Embedded Bereich genutzt werden. Der Java−Bytecode wird auf Anforderung über einen Webserver an die Clients (Browser) geschickt, die die Anwendung auf dem Host ausführen. Daraus resultiert dass der Webserver lediglich über genügend Speicherplatz zum Ablegen der Applets verfügen muss, die Rechenleistung spielt keine Rolle, da der Code in der virtuellen Maschine des Aufrufers läuft.
Der folgende Screenshot zeigt die identische Anwendung als Java−Applet in eine Webseite eingebunden.



Java Screenshot mit der Beispielanwendung als Applet (Beispielcode hier)

SWT und Eclipse

Obwohl Java über ähnliche Eigenschaften zur Darstellung von grafischen Steuerlementen verfügt wie andere Toolkits, gab es unter Software−Entwicklern in der Vergangenheit immer wieder unzufriedene Stimmen. Als größte Probleme wurde der mangelnde Funktionsumfang und geringe Geschwindigkeit von Swing genannt. Als Alternative wurde von IBM das Standard Widget Toolkit (SWT) entwickelt, das unter Java die Verwendung von nativen Elementen erlaubt. Ein Referenzprojekt ist die ebenfalls von IBM entwickelte IDE Eclipse, die eine offene Plattform für Entwicklungswerkzeuge darstellt. Sowohl Toolkit als auch Entwicklungsumgebung sind freie Software.

Abkürzung in Zusammenhang mit JAVA

JDK (Java Development Kit)	Das komplette Java Paket zum Erstellen von Java Anwendung bestehend aus Anwendungen, Java Klassen und Dikumentation.
JRE (Java Runtime Environment)	enthält die virtuelle Maschine und ist zum Benutzen von Java Anwendungen notwendig.
J2ME (Java 2 Micro Edition)	Java für Geräte mit geringen Ressourcen.
J2SE (Java 2 Standard Edition)	Java für den Desktop (Linux, Windows, ...)
J2EE (Java 2 Enterprise Edition)	Java zum Erstellen mehrschichtiger Client−/Server−Anwendungen sowie von Java Servlets und Java Server Pages.
JFC (Java Foundation Class)	Klassensammlung zum Erstellen von Oberflächen (−> Swing)

Übersicht über JAVA

Name:	JAVA 2 PLATFORM STANDARD EDITION DEVELOPMENT KIT 5.0
Version:	5.0
Betriebssysteme:	• Linux, Windows, Solaris (SUN) • Linux, Windows, AIX, Solaris (evtl. MacOS, OS/2, FreeBSD, Amiga, BeOS) (Jikes −> IBM)
Programmiersprache:	JAVA
Lizenz:	proprietäre Lizenz (SUN)
Vorteile:	• robuste Sprache (viele Fehlerquellen sind durch das Sprachkonzept ausgeschlossen) • große Anzahl von leistungsfähigen Tools wie z.B. Eclipse
Nachteile:	• proprietäre Sprache, alleinige Kontrolle durch SUN • virtuelle Maschine der Zielplattform muss passen • langsame Ausführungsgeschwindigkeit • SWT aufwändiger zu programmieren als Swing
Entwicklungsumgebung:	z.B. Eclipse
WWW:	http://java.sun.com
Dokumentation:	Manuals, Tutorials Allgemein: http://java.sun.com/j2se/1.5.0/docs/, http://www−e.uni−magdeburg.de/mayer/java.html SWT: http://eclipse−wiki.info/SWT, http://www.java−tutor.com/java/swtlinks.html

Referenzprojekte:	•
Verbreitung:	sehr große Verbreitung

Kylix

Kylix ist eine Cross−Platform Entwicklungsumgebung für Linux und Windows. Mit Hilfe der Borland CLX Bibliothek (Component Library for Cross−platform) können Anwendungen unter Delphi und C++ erstellt werden, die dann auf beiden Plattformen laufen. Laut einem Bericht auf der Wikipedia−Homepage (Link hier) ist diese Bibliothek aber nichts anderes als ein Wrapper für die bereits vorgestellte Bibliothek Qt. Zusätzlich handelt es sich bei der Kylix IDE offenbar um eine auf WINE (Link hier) basierende nicht−native Linuxanwendung, deren erstellten Executables gegen libwine gelinkt werden müssen. Wenn man dies berücksichtigt dürfte der Einsatz von Kylix für C++−Entwickler wenig Sinn machen, da hier der Einsatz von Qt mit einer freien IDE wesentlich geradlinige ist.

Übersicht über Kylix

Name:	Kylix
Version:	3
Betriebssysteme:	Windows, Linux
Programmiersprache:	Delphi, C++
Lizenz:	proprietäre Software
Vorteile:	• Entwicklung unter Delphi und C++
Nachteile:	• Lizenzkosten
Entwicklungsumgebung:	Kylix
WWW:	http://www.borland.de/kylix
Dokumentation:	
Referenzprojekte:	
Verbreitung:	geringe Verbreitung

Smalltalk

Ein Klassiker unter den Programmiersprachen ist Smalltalk. In den Jahren 1969/70 von Xerox entwickelt, ist die Sprache bis heute ein Musterbeispiel für völlige Objetorientierung. In Smalltalk ist alles ein Objekt, es gibt keine einfachen Datentypen. Smalltalk arbeitet wie Java und .Net (siehe unten) mit einer virtuellen Maschine. Die Sprachsyntax orientiert sich an der natürlichen Sprache, unterscheidet sich aber deutlich von anderen existierenden Programmiersprachen. Smalltalk wurde bereits damals mit einer grafischen Oberfläche programmiert, weshalb man sagt, dass Smalltalk seiner Zeit 10−15 Jahre vorraus war. Es ist möglich zur Laufzeit das Kernsystem zu ändern und diese Modifikationen sind im Anschluß, ohne dass ein Neustart nötig ist, gültig. Smalltalk war relativ erfolgreich bis Java aufkam.

Das obligatorische 'Hello world !' unter Smalltalk:

```
Transcript show: 'Hello world !'; cr.
```

Smalltalk wird auch heute noch für Projekte eingesetzt. Die am weitesten verbreitete Variante ist Smalltalk−80 (Normierung von 1980). Eine leistungsfähige Entwicklungsumgebung ist beispielsweise Squeak.

Übersicht über Smalltalk

Name:	Smalltalk (z.B. Squeak)
Version:	3.6
Betriebssysteme:	Windows, Linux, Solaris, MacOSX, Darwin
Programmiersprache:	Smalltalk
Lizenz:	Open Source
Vorteile:	• völlig objektorientiert
Nachteile:	• Smalltalk wird von Java verdrängt und hat eine deutlich kleinere Nutzerzahl
Entwicklungsumgebung:	z.B. Squeak
WWW:	http://www.smalltalk.org
Dokumentation:	
Referenzprojekte:	
Verbreitung:	geringe Verbreitung

Mozilla

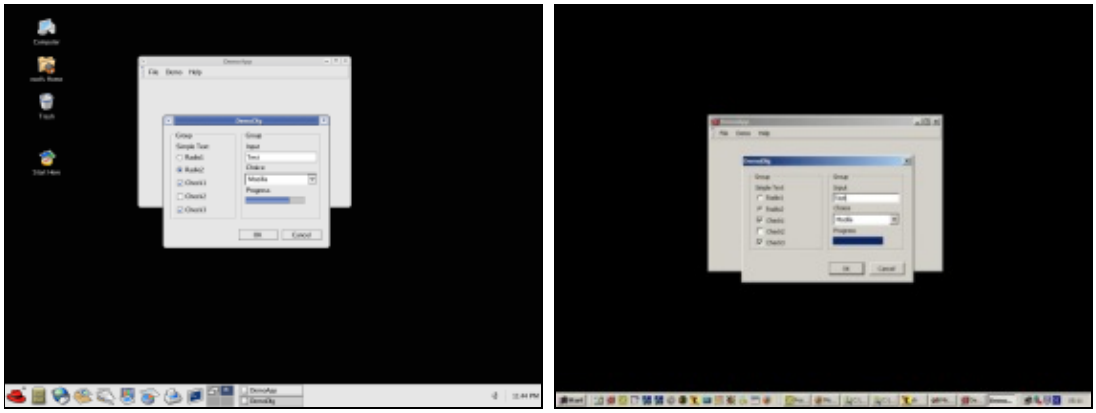
Mozilla? Ein Browser? Wie soll man mit einem Browser programmieren? Bei Mozilla handelt es sich nicht nur um einen Webbrowser, sondern zusätzlich um ein plattformübergreifendes Framework, das verschiedene Standards wie z.B. XUL (XML based user interface language) unterstützt. XUL wird verwendet um die Struktur und den Inhalt einer Applikation zu definieren. Sämtliche Dateien werden als Klartextdateien abgelegt. Mozilla unterscheidet nicht zwischen Programmen und Webseiten da diese konzeptionell sehr ähnlich sind. Folgende Zeile im Feld für dir URL eingegeben läßt den Browser sich selbst rendern:

```
chrome://navigator/content
```

Der folgende Code stellt im Mozilla Browser einen Button dar, der beim Anklicken ein Fenster mit dem Text "Hello World" öffnet.

```
<?xml version="1.0"?>

<!-- Beispiel XUL Datei -->

<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<box align="center">

  <button label="Push" onclick="alert('Hello World');" />

</box>

</window>
```

Softwareentwicklung mit Mozilla entscheidet sich grundlegend von der klassischen Programmierung. Viele innovative Konzepte haben Einzug gefunden, wie zum Beispiel die Trennung der Anwendung von der Darstellung. So können existierende Anwendungen durch geänderte 'Themes' ihr Aussehen ändern. Erfolgreiche Projekte wie der Mozilla Firefox Browser zeigen, dass es sich bei diesem Framework um eine ausgereifte und robuste Umgebung handelt.



Linux und Windows 2000 Screenshot (Quellcode hier).

Übersicht über Mozilla

Name:	Mozilla
Version:	1.6
Betriebssysteme:	Windows, Linux,
Programmiersprache:	XUL
Lizenz:	Mozilla Public License, Netscape Public License
Vorteile:	• innovative Konzepte • bestehende Webstandards werden unterstützt (JavaScript, Stylesheets, ...) • Anwendungen können im Browser oder als standalone Anwendung dargestellt werden.
Nachteile:	•
Entwicklungsumgebung:	
WWW:	http://www.mozilla.org
Dokumentation:	Manuals, Tutorials, Mailing Listen z.B. www.xulplanet.com
Referenzprojekte:	Mozilla Firefox Browser
Verbreitung:	große Verbreitung, aber selten bei Softwareprojekten eingesetzt

Microsofts Antwort

Natürlich hat auch Microsoft die Zeichen der Zeit erkannt und inzwischen einen eigenen Ansatz präsentiert. Unter dem Name .NET wurde eine Plattform entwickelt, die nicht zuletzt den Abwanderungsprozess von Software−Entwicklern zur Konkurrenzplattform Java eindämmen soll. Bei näherer Betrachtung finden sich auch tatsächlich viele Parallelen zwischen den beiden Konkurrenten, auch wenn diese durch unterschiedliche Namensgebung verschleiert werden. So heißt beispielsweise das Gegenstück zu Javas 'Bytecode' bei Microsofts C# 'Intermediate Language' (MSIL).

Was ist .NET ?

.NET ist eine proprietäre Microsoft Technologie auf der alle weiteren Microsoft Produkte basieren sollen. Die Unterstützung der bisher favorisierten MFC Bibliothek für Visual C++ wurde im Zuge der .NET Einführung eingestellt. .NET soll die Entwicklung von Netzwerk− und Internet−Anwendung vereinfachen und hat viele Ideen von Java übernommen. Es unterstützt objektorientierte Programmierung und wird mit einer einzigen Klassenbibliothek geliefert, die von verschiedenen Programmiersprachen (C#, VB.NET) genutzt werden kann. Das bedeutet dass aus dem Programmcode die 'Intermediate Language' erzeugt wird, die über das .NET−Framework auf die Zielhardware zugreift (vgl. Java Sourcecode −> Java Bytecode −> virtuelle

Maschine –> physikalische Hardware).

Zukünftige Windows Versionen werden mit dem .NET Framework ausgeliefert.

Was ist Visual Sudio .NET ?

Visual Studio .NET ist eine Programmierumgebung, die das Entwickeln von .NET–Software vereinfachen soll, es ist aber nicht zwingend erforderlich.

Unterschiede zwischen Visual Basic (VB) und VB.NET

Obwohl VB.NET aus Kompatibilitätsgründen viele alte VB Funktionen unterstützt und die Sprach–Syntax beibehalten wurde, ist VB.NET eine komplett neue Programmiersprache.

Welche Programmiersprache eignet sich am besten?

Da sowohl der VB.NET Quellcode als auch der C# Quellcode in die MSIL übersetzt werden, macht die verwendete Programmiersprache keinen Unterschied. So gibt es zum Beispiel auch keine Geschwindigkeitsunterschiede zwischen C# Code und VB.NET Code. Da der C# Compiler aber speziell für das .NET Framework entwickelt wurde, dürfte er das geeignetere Tool sein.

.NET und Linux

Trotz des plattformunabhängigen Ansatzes wird Microsoft wohl keine Linux .NET–Variante entwickeln, weshalb sich ein Entwicklerteam um Miguel de Icaza (Ximian: Evolution) dieser Aufgabe angenommen hat. Das Open Source Paket namens Mono ist mittlerweile in der Version 1.0 verfügbar.

Übersicht über .NET

Name:	Microsoft .NET–Framework
Version:	1.1
Betriebssysteme:	Windows, Linux
Programmiersprache:	C#, Windows: VB.NET
Lizenz:	proprietäre Lizenz
Vorteile:	• zukünftiger Bestandteil von Windows
Nachteile:	

	• proprietäre Software • keine Linux .NET Version verfügbar • komplett neue API
Entwicklungsumgebung:	Visual Studio .NET
WWW:	
Dokumentation:	
Referenzprojekte:	
Verbreitung:	bisher nur geringe Verbreitung

Zusammenfassung

Vor der abschließenden Beurteilung wird noch einmal auf die gestellte Aufgabe verwiesen: Es geht um die Entwicklung eines Front−Ends, das über Netzwerk−Kommunikation mit angeschlossener Hardware kommuniziert. Dabei soll der Software−Quellcode auf den Plattformen Linux und Win32 übersetzbar sein. Die Anwendung darf sich nicht von anderer Software auf dem System unterscheiden. Durch diese Aufgabenstellung wird das Bild auf die getesteten Pakete verfälscht und kann nicht als allgemeine Aussage gesehen werden.

Bestes Beispiel hierfür ist das FLTK Toolkit. Hier bekommt man in einem sehr kleinen Paket ein leistungsfähiges System. Die Stärken liegen in geringer Codegröße, guter Grafikanbindung und guter Portierbarkeit. Diese Eigenschaften machen das Toolkit für Projekte im Embedded− und Grafikbereich interessant. Für die Front−End Entwicklung spielt aber eher die Anzahl der Klassen, das Handling und das Aussehen der erzeugten Anwendungen eine Rolle. Somit ist FLTK für diese Aufgabe eher weniger geeignet.

Eine herbe Enttäuschung für Software−Entwickler dürfte das GTK+ Projekt unter Windows sein. Hier sollte die Linux Community deutlich mehr Einsatz zeigen. Mit Warnungen auf der Web−Seite gewinnt man jedenfalls kein Vertrauen. Dies ist um mehr schade, als dass das GTK+ Paket als reines Linux Paket sehr gelungen wirkt. Das Potential ist sehr groß, die Umsetzung auf die Windows Plattform läßt aber zu wünschen übrig.

Der Einsatz der Exoten Smalltalk und Mozilla bleibt Geschmacksache. Eine Firma, die mit selbstentwickelter Hardware Geld verdient, wird wenig Sinn für philosophische Ansätze haben. Auch wenn Smalltalk die bessere objektorientierte Programmiersprache ist und wenn Mozilla−XUL−Programmierung den sowieso vorhandenen Browser noch sinnvoller erscheinen läßt, so sind diese Pakete doch keine Mainstream Produkte zur Software−Entwicklung.

Kylix fällt bei dieser Betrachtung ebenso wie GTK+ für Win32 eher negativ auf. Von dem einstigen Glanz des Urprodukts Turbo Pascal ist nur wenig geblieben. Mit diesem Produkt präsentierte Borland in den 80er Jahren eine mächtige IDE die sowohl auf Home−Computern wie auch auf den frühen PCs lief. Es war bekannt für seinen günstigen Preis und seinen schnellen Code. In der Zwischenzeit hat sich viel geändert. Aus Borland wurde Inprise und dann wieder Borland. Aus Turbo Pascal wurde erst Object Pascal, dann Delphi und schließlich Kylix (natürlich mit Erweiterungen bzw. Veränderungen). Zumindest für neue Projekte dürfte ein Einsatz heute nur noch wenig Sinn machen.

Microsoft zeigt in diesem Umfeld, dass es die Zeichen der Zeit erkannt hat. Historisch betrachtet hat der Konzern zunächst versucht, den Java Standard mit Visual J++ zu durchsetzen. Neben den Java Standardbefehlen wurden Win32 API Zugriffe und selbst Zugriffe auf die Windows Registry zugelassen (Was der Sprachphilosphie völlig widerspricht). Außerdem wurden automatisch Win32 Executables erzeugt. Nach gerichtlichen Auseinandersetzungen mit Sun mußte daraufhin bei jedem Übersetzungsvorgang ein Warnhinweis angezeigt werden, dass die erzeugten Applikationen wahrscheinlich nicht auf anderen Betriebssystemen laufen werden. Das Ende der Geschichte war das Einstellen des Java Engagements von Microsoft. Stattdessen wurde eine völlig neue Strategie entwickelt. Mit .NET und #C wurde ein komplett neuer Standard entwickelt. Die Kombination Windows, .NET und C# sind sicherlich ein gut zusammenpassendes Paket, aber das muss man Microsoft auch für die ausgediente Kombination Windows mit Visual C++ und MFC Klassenbibliothek zugestehen. Der Nachteil dabei ist, dass man bedingungslos einem Anbieter ausgeliefert ist, der "seinen" Standard (hier: Windows) durchsetzen möchte. Mit an Sicherheit grenzender Wahrscheinlichkeit wird Microsoft in absehbarer Zeit keine Umsetzungen von .NET auf andere Betriebssysteme planen. Die freie Umsetzung Mono muss ihre Praxistauglichkeit erst noch beweisen. Hier kann zu gegenwärtigen Zeitpunkt noch kein Resumee gezogen werden, auch wenn schon erste Achtungserfolge erzielt wurden.

Ohne Einschränkung empfehlenswert sind somit die Pakete Qt, wxWindows und Java. Die Wahl fällt hier schwer, da alle 3 Pakete zum Erzeugen komplexer Front−End Software taugen. Je nach Gewichtung der Punkte Support, Kosten, Einsatzbereitschaft, Programmier−Philosophie, usw. dürften sich hier verschiedene Meinungen bilden. Die Unterschiede liegen eher im Detail, so verbietet die Java Philosophie eigentlich direkte Hardware Zugriffe. Dafür kann es wieder bei anderen Aspekten punkten. Rein technisch gesehen können diese 3 Konkurrenten die gestellte Aufgabe problemlos meistern.

Somit bleibt nur ein subjektives Ergebnis des Autors: Als Open Source Fan wird man bei der gestellten Aufgabe wohl am ehesten zu wxWindows tendieren. Neben einem stimmigen Konzept und guter Tool Unterstützung sind ebenfalls ausreichend Dokumentation vorhanden.
