Überblick über die GNOME-Entwicklungsplattform -- Material zum Vortrag "Einführung in die GNOME-Programmierung"

Matthias Warkus

Inhaltsverzeichnis

Umfang dieses Skripts

Es kann nicht Ziel dieses Skripts sein, eine detaillierte Anleitung oder auch nur eine vollständige Übersicht über die GNOME-Entwicklung zu geben; dazu ist das Feld mittlerweile zu umfangreich. Bereits einzelne Teilgebiete wie die GNOME-Komponentenarchitektur könnten je nicht bloß Material zu einem ganzen Vortrag, sondern zu einer ganzen Konferenz liefern. (Was Konferenzen zur GNOME-Entwicklung angeht, siehe auch GUADEC [http://www.guadec.org].)

Ich möchte daher nur versuchen, eine Art Streiflicht über das Thema GNOME-Programmierung zu werfen. Dazu werde ich skizzieren, wie die GNOME-Entwicklungsplattform aussieht und beschreiben, was eine GNOME-Anwendung eigentlich zu einer GNOME-Anwendung macht - sowohl im programmiererischen als auch im nicht-programmiererischen Bereich. Am Schluss soll ein Ausblick in die Zukunft von GNOME aus der Entwicklerperspektive stehen.

Dass der zugehörige mündliche Vortrag beim LinuxTag 2003 den Schwerpunkt mehr auf das Praktische legt als dieser doch eher trockene Überblick, liegt in der Natur der Sache. Wie ich beim Erarbeiten dieses Skripts bemerkt habe, wäre bereits ein einfaches Programmierbeispiel für eine ‘rechtsgültige’ GNOME-Anwendung nebst Schritt-für-Schritt-Erläuterung des Codes deutlich zu umfangreich, um hier neben allem anderem Platz zu finden. Dieser praktische Teil wird daher ‘live’, während des Vortrages stattfinden.

Sie können jedoch dem Paket gnome-2-beispiele.tar.gz [http://www.klinkenbuchse.de/gnome-2-beispiele.tar.gz], das ich auf meiner Homepage bereitgestellt habe, eine reichliche Menge von ausführlich kommentierten Programmierbeispielen für GNOME 2 entnehmen.

Architektur und Grundkonzepte der GNOME-Plattform

Die G-Welt

GNOME-Programmierung spielt sich in einer Welt ab, die ich auf Grund des inflationären Vorkommens des Anfangsbuchstabens G in Programm-, Bibliotheks-, Klassen-, Typ- und Funktionsnamen als ‘G-Welt’ bezeichnen möchte. Folgendes macht diese Welt aus:

- Zu Grunde liegende Programmiersprache ist C; darauf können allerdings beliebige Bindungen für andere Sprachen aufsetzen und tun es auch
- Quellbäume werden mit GNU Automake, Autoconf, Libtool etc. konfiguriert, wobei pkgconfig eingesetzt wird, um abzufragen, welche Pakete das System bereitstellt
- Die Lokalisierung erfolgt mit GNU Gettext, unterstützt durch Intltool
- Für einfache Datenstrukturen, Hilfsfunktionen und Abstraktionen wird GLib, die fundamentalste Bibliothek der GNOME-Plattform eingesetzt; bei allen Datentypen gilt die Konvention, dass die mit Großbuchstaben beginnenden Teile eines Typnamen den auf diesem Typen arbeitenden Funktionen in Kleinbuchstaben und durch Unterstriche getrennt vorausgehen; d.h. z.B., alle Funktionen, die auf dem Typen "GMemChunk" arbeiten, beginnen mit "g_mem_chunk_"
- Objektorientierung wird durch GObject bereitgestellt; fast alle Überprüfungen passieren zur Laufzeit, was Bindungen für interpretierte Sprachen deutlich einfacher macht
- GTK+ dient als Widget-Bibliothek zum Darstellen grafischer Benutzeroberflächen unter X11 oder inzwischen auch unter anderen Plattformen
- Mit dem Oberflächen-Editor Glade können Dialoge und Anwendungsfenster visuell entworfen und als XML-Beschreibungen gespeichert werden, die das Programm später zur Laufzeit lädt und darstellt

All dies kann ein Programm noch verwenden, ohne ein GNOME-Programm zu sein. "Reine" GTK+-Anwendungen wie The GIMP halten sich in der G-Welt, aber nicht im GNOME-Land auf.

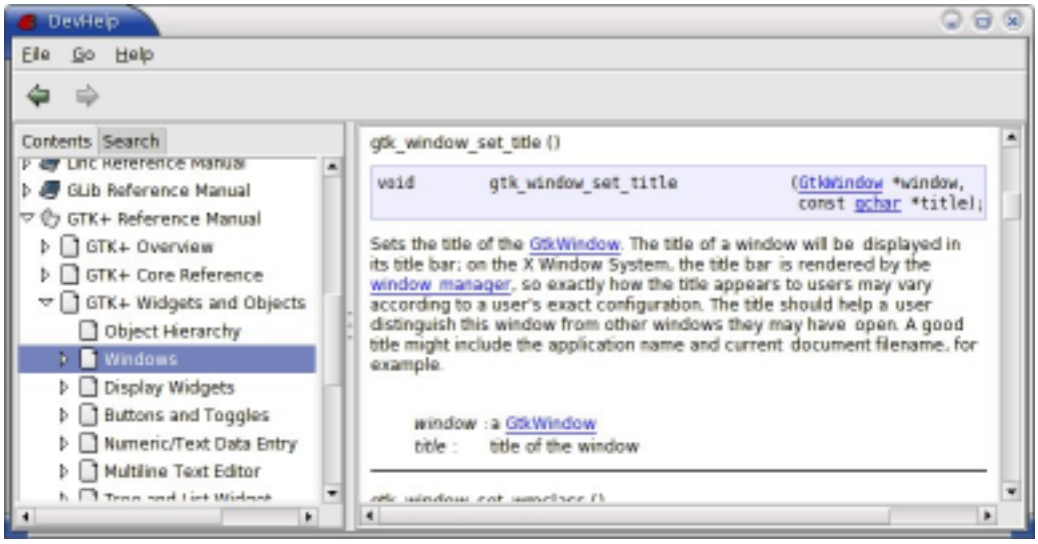
Die GNOME-Plattform als Bündel von Bibliotheken

GNOME stellt sich nicht als eine große, monolithische Bibliothek dar, die alle Klassen und Funktionen bereitstellt, die GNOME-Funktionalität ausmachen, sondern als ein ganzes Bündel einer (leider?) erstaunlich großen Zahl von Einzelbibliotheken, die jeweils einen bestimmten Zweck erfüllen. Nur einige davon exponieren jedoch für den Entwickler wichtige APIs; zu ihnen gehören:

- libgnome/libgnomeui -- GNOME-eigene Hilfsfunktionen und Erweiterungen von GTK+ um GNOME-spezifische Widgets. Diese Bibliotheken sollen irgendwann einmal unnötig werden.
- libgnomecanvas -- Der GNOME-Canvas, eine mächtige, an den Tk-Canvas angelehnte Zeichenfläche.
- GNOME-VFS -- GNOMEs virtuelles Dateisystem für synchrone und asynchrone, netzwerktransparente Dateizugriffe
- GConf -- GNOMEs Konfigurationssystem
- libxml2 -- Ein flexibler XML-Parser, der Königsweg zum einfachen Laden und Speichern strukturierter Informationen in der GNOME-Welt

- GNOME-Print -- GNOMEs Drucksystem: sorgt für Plattformunabhängigkeit und u.A. eine pixelgenaue Druckvorschau
- GtkHTML -- HTML-Renderer
- Bonobo -- GNOMEs Komponentenobjektmodell, in etwa vergleichbar mit COM u.Ä.

Der Dokumentationszustand der meisten dieser Bibliotheken hat sich in letzter Zeit um einiges verbessert, lässt aber immer noch viel zu wünschen übrig. An dieser Stelle sei erwähnt, dass es mit DevHelp [http://devhelp.codefactory.se] mittlerweile einen Hilfebrowser speziell für Entwickler gibt, der den Zugriff auf die Referenzdokumente der installierten Bibliotheken stark vereinfacht (siehe Abbildung).



Was macht eine Anwendung zur GNOME-Anwendung?

Nun ist es so, dass noch lange nicht jedes grafische Programm mit einer GTK+-Oberfläche, das vielleicht noch gegen die ein oder andere GNOME-Bibliothek linkt, schon eine GNOME-Anwendung ist. Das Wesen einer ‘wahren’ GNOME-Anwendung wird von zahlreichen Faktoren bestimmt.

Konfiguration mit GConf

Anwendungen haben es an sich, konfigurierbar zu sein. Dies bedeutet letztlich, dass irgendwo eine Reihe von Werten vorliegt, deren Beträge das Verhalten oder das Aussehen der Anwendung bestimmen.

Um dies zu realisieren, ginge man normalerweise etwa so vor:

1. Datenstrukturen für die Werte definieren
2. Die Anwendung auf diese Werte zurückgreifen lassen, wo immer sie sich nach ihnen richtet
3. Einen Konfigurationsdialog schreiben, der die Einstellungen in diese Werte einträgt
4. Implementieren, dass bei Programmende die Einstellungen in eine Datei niedergeschrieben bzw. bei Programmstart sie daraus gelesen werden

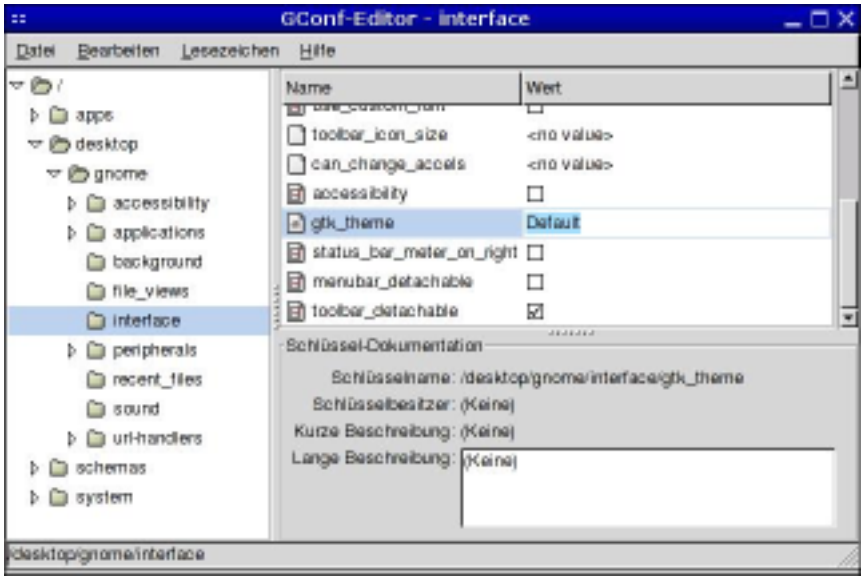
Hieraus ergeben sich sofort einige Probleme. Wie erfährt beispielsweise die Anwendung, wenn ein Konfigurationswert geändert wurde? Welches Dateiformat wird verwendet, und wo landet die Datei? Was passiert, wenn die Konfigurationsdatei geändert wird, während die Anwendung läuft? Und heißt es nicht das Rad neu erfinden, diesen Mechanismus für jede Anwendung neu zu schreiben?

GConf löst die Probleme auf ziemlich elegante Art und Weise. Es gibt eine zentrale Konfigurationsdatenbank, die eine als Verzeichnisbaum gegliederte Hierarchie von typbehafteten Schlüsseln enthält, die Werte aufnehmen können. Sie tun nunmehr folgendes:

1. Die Konfigurationsdaten in GConf-Werte herunterbrechen, die in GConf-Schlüsseln unterkommen können
2. Von Konfigurationswerten abhängige Anwendungsteile durch Signalhandler an GConf-Schlüssel koppeln
3. Den Konfigurationsdialog so schreiben, dass er die Einstellungen in die GConf-Schlüssel einträgt

Dies führt dazu, dass jede Änderung an der Konfigurationsdatenbank, ganz gleich, wann und von wo sie erfolgt, sofort an die Anwendung durchgereicht wird (dies wird übrigens auch von den GNOME-Ergonomierichtlinien verlangt -- siehe unten). Die Kopplung zwischen der Stelle, an der die Einstellungen vorgenommen werden, und jener, an der sie wirken, erfolgt nur über die Datenbank, ohne anwendungsinterne Logik. Auch über die Speicherung der Schlüssel müssen Sie sich keine Gedanken mehr machen. Seiteneffekt des Ganzen ist, dass es einfach ist, die Anwendung über externe Werkzeuge zu konfigurieren, was besonders Systemadministratoren freut.

Ein solches externes Werkzeug wird bei GNOME schon mitgeliefert, nämlich der GConf-Editor (siehe Abbildung).



Dateizugriff mit GNOME-VFS

GNOME-VFS ist eine Bibliothek, die GNOME-Anwendungen ein virtuelles Dateisystem bereitstellt; daher auch der Namensteil ‘VFS’ für ‘Virtual File System’.

Warum nun aber ein virtuelles Dateisystem benutzen und nicht einfach auf das native zugreifen? Neben dem allfälligen Rechtfertigungsgrund für Abstraktionsschichten, der Portabilität, geht es auch darum, den Begriff des Dateisystems zu erweitern -- mit GNOME-VFS können Sie nicht nur Inhalte auf einer lokalen Platte als Dateisystem ansprechen, sondern auf gleicher Weise auch den Inhalt einer Archivdatei oder den eines Web"=Servers auf der anderen Seite des Planeten.

Das Grundkonzept von GNOME-VFS steht auf zwei Säulen: Es werden URIen statt Dateinamen verwendet, um Dateien oder Verzeichnisse zu identifizieren, und die Funktionen sind transparent, das heißt, alle Zugriffe auf

Dateien verwenden dieselbe URI-basierte API, unabhängig davon, wo die Dateien sich befinden und in welcher Art auf sie zugegriffen wird. Wenn Sie also Ihre Zugriffe über GNOME-VFS abwickeln, können Sie überall dort, wo Sie sonst Dateinamen akzeptieren würden, URIen akzeptieren und ihre Benutzer damit zum Beispiel Dateien, die irgendwo im Netz liegen, genauso zugänglich machen wie lokale, ohne Code für Netzwerkzugriffe schreiben zu müssen. Die zur Unterstützung von Protokollen (in GNOME-VFS: "Schemen") notwendigen Routinen sind in dynamisch geladenen Modulen untergebracht, so dass beliebige neue Protokollhandler hinzugefügt werden können.

GNOME-VFS erweitert das URI-Konzept um ‘verschachtelte Schemen’. Das heißt, dass in einer Adresse ein weiteres Schema angehängt werden kann, mit dem das Ergebnis des vorigen Schemas geöffnet wird. So bedeutet beispielsweise ein URI wie "http://www.revival-revival.de/dublin78.tar.gz#gzip#tar/dublin78/wasn-brett.mp3" GNOME-VFS, ein Archivdatei aus dem Netz herunterzuladen, zu dekomprimieren und aus ihr eine bestimmte Datei zu entnehmen.

Über all dies hinaus liefert GNOME-VFS auch eine Menge praktischer Hilfsfunktionen mit, allen voran ein Subsystem für asynchrone Zugriffe. Dies ist für moderne grafische Anwendungen von immenser Bedeutung -- nur indem Dateizugriffe asynchron, das heißt, im Hintergrund, verlaufen, kann sichergestellt werden, dass Anwendungen auch während langen Zugriffen bedienbar bleiben.

GNOME-VFS übernimmt zu guter Letzt auch noch die Zuordnung zwischen Dateien und MIME-Typen einerseits sowie MIME-Typen und darauf arbeitenden Anwendungen andererseits. Das komplette Verknüpfungssystem ist nicht nur über den entsprechenden Konfigurationsdialog für den Benutzer, sondern über eine API auch für den Entwickler zugänglich.

Einbindung in die Infrastruktur: Menüeinträge und Dateitypen

Dies führt direkt zum nächsten Punkt, dem Einbinden von Anwendungen in die GNOME-Infrastruktur. Programme sollten im GNOME-Hauptmenü aufrufbar sein und außerdem bekannt geben, welche Dateitypen sie öffnen können.

Ersteres geschieht über einen .desktop-Eintrag im Verzeichnis share/applications des GNOME-Installationsbaumes. Das Format von .desktop-Dateien ist standardisiert (KDE verwendet dasselbe); der Eintrag nimmt Name, Beschreibung, Aufrufname, Kategorisierung etc. der Anwendung auf und landet automatisch im GNOME-Menü. Ein Beispiel sehen Sie hier (der Übersichtlichkeit halber wurden nur die deutschen und französischen Übersetzungen der natürlichsprachigen Einträge übernommen):

```
[Desktop Entry]
Encoding=UTF-8
Name=PDF file viewer
Name[de]=PDF-Dateibetrachter
Name[fr]=Visualiseur de fichier PDF
Comment=Tool for viewing PDF files
Comment[de]=Werkzeug zur Anzeige von PDF-Dateien
Comment[fr]=Visualiseur de fichier PDF (Portable Document Format)
Exec=gpdf
Icon=document-icons/gnome-application-pdf.png
Terminal=false
Type=Application
Categories=GNOME;Application;Graphics;VectorGraphics;Viewer;
StartupNotify=true
```

Das Verknüpfen von Dokumenttypen mit einer Anwendung ist geringfügig schwerer. Strenggenommen kann ein Dateityp gar nicht mit einer Anwendung verknüpft werden, sondern es kann nur ein neuer MIME-Typ für einen oder mehrere bestimmte Dateitypen deklariert und andererseits die Unterstützung einer Anwendung für bestimmte MIME-Typen bekannt gegeben werden.

Die Deklaration neuer MIME-Typen geschieht zum Beispiel, indem eine Datei mit dem Suffix .mime in das Verzeichnis share/mime-info des GNOME-Baumes kopiert wird. Eine solche Datei kann beliebig viele MIME-Typen deklarieren und Erkennungsmerkmale wie Dateinamensuffixe o.Ä. angeben. Hat eine Anwendung allerdings keinen eigenen, neuartigen Dateityp, reichen die zahlreichen von GNOME-VFS vorgegebenen Erkennungsheuristiken zur Bestimmung zahlloser mehr oder weniger gängiger MIME-Typen durchweg aus. Hier trotzdem zur Veranschaulichung die Datei gnumeric.mime, die alle von der GNOME-Tabellenkalkulation Gnumeric neu deklarierten MIME-Typen an Dateinamensuffixe bindet:

```
application/x-gnumeric
 ext: gnumeric

application/vnd.ms-excel
 ext: xls xlw

application/x-applix-spreadsheet
 ext: as

application/vnd.lotus-1-2-3
 ext: 123 wk4 wk3 wk1
  </programlisting>

  <para>Unterstützung für ein oder mehrere MIME-Typen wird bekanntgegeben
 durch Dateien mit dem Suffix .keys im selben Verzeichnis. Sehen Sie
 hierzu einen ebenfalls weitgehend selbsterklärenden Abschnitt aus↩
gnumeric.keys:</para>

  <programlisting>
<![CDATA[application/x-gnumeric:
      open=gnumeric %f
      view=gnumeric %f

    ↩
icon-filename=/gnome/garnome/share/pixmaps/gnumeric/gnome-application-x-gnumeric.png


application/vnd.ms-excel:
      open=gnumeric %f
      view=gnumeric %f

    ↩
icon-filename=/gnome/garnome/share/pixmaps/gnumeric/gnome-application-x-xls.png

application/vnd.lotus-1-2-3:
      open=gnumeric %f
      view=gnumeric %f

    ↩
icon-filename=/gnome/garnome/share/pixmaps/gnumeric/gnome-application-vnd.lotus-1-2-3.png
```

Ergonomische Gestaltung

Lange Zeit hatten sowohl die Kernkomponenten von GNOME als auch auf der GNOME-Plattform aufsetzende Drittanwendungen das Problem mangelnder Konsistenz in der Benutzerführung. Selbst in einfachen Dingen wie der Benennung und Anordnung von Menüeinträgen gab es jahrelang keine Konventionen.

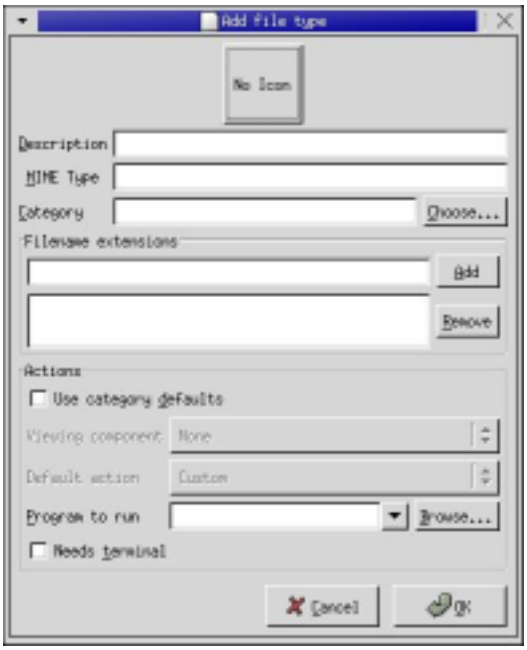
Selbst kleine Inkonsistenzen führen jedoch bereits zur Verwirrung der Benutzer, zu Verzögerungen in der Bedienung und damit zum Verlust von Produktivität. Der Anspruch von Ästhetik, Einheitlichkeit und Benutzerfreundlichkeit, den GNOME sich selbst stellt, kann nur eingelöst werden, wenn es einheitliche Richtlinien für die Gestaltung von GNOME-Anwendungen gibt.

Mit den seit August 2002 vorliegenden GNOME Human Interface Guidelines [http://developer.gnome.org/projects/gup/hig/] gibt es solche Richtlinien nun. Es würde den Rahmen dieses Skripts sprengen, im Einzelnen abzuhandeln, worin

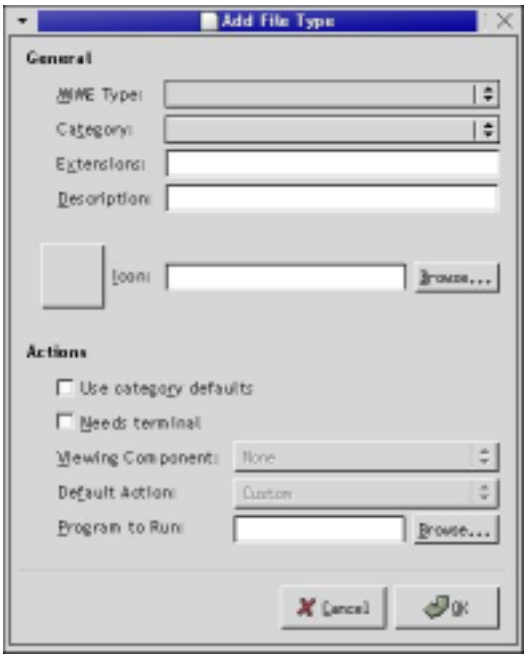
all diese Richtlinien bestehen. Von allgemeinen Regeln zur Ergonomie bis hin in die Details des visuellen Design von Anwendungen hinein geben die HIG GNOME-Entwicklern eine umfassende Hilfe zur ergonomischen, standardkonformen Gestaltung von Anwendungen an die Hand.

Erwähnenswert ist vielleicht, dass den HIG das Ideal der visuell ruhigen Anwendung zu Grunde liegt -- das heißt: Abstände statt Linien zur optischen Gruppierung; keine Überladung von Fenstern mit Bedienelementen; klare, ausgerichtete Blöcke von Elementen statt Chaos. Betrachten Sie zum besseren Verständnis die folgenden Abbildungen, die ein Dialogfenster einmal vor und einmal nach der richtlinienkonformen Umgestaltung zeigen.

Vorher:



Nachher:



Barrierefreiheit

Unter diesem Schlagwort versteht man in der Programmierung dasselbe wie in der Architektur: nämlich, so zu planen, dass von vornherein keine Zugangshindernisse für Behinderte entstehen -- so müssen auch nachträglich keine überbrückt werden.

Barrierefreiheit in der GNOME-Programmierung bedeutet zuallererst korrekte und aussagekräftige Beschriftung sowie Erreichbarkeit aller Bedienelemente einer Anwendung von der Tastatur aus. GTK+ und GNOME stellen dafür komfortable Hilfsfunktionen bereit, so dass es keinen großen Zusatzaufwand bedeutet, hierfür zu sorgen. Da GTK+' mitgelieferte Widgets ihre Daten ohnehin über das Accessibility Toolkit ATK weitergeben, muss nur bei selbstentwickelten Widgets diese 'von Hand' eingebaut werden.

Abgesehen von der für nichtbehinderte Benutzer bereits sehr angenehmen Tastaturbedienbarkeit stellt dies auch sicher, dass Zusatzanwendungen wie Bildschirmvorleser, Vergrößerungswerkzeuge, Bildschirmtastaturen usw. in die Anwendung eingreifen können. Ein weiterer Nebeneffekt ist somit, dass es später einmal sehr einfach sein wird, solchermaßen zugängliche Software von einem Makrorekorder o.Ä. fernzusteuern.

Außer im Code selber ist Barrierefreiheit auch noch in Teilen des visuellen Designs zu berücksichtigen; zum Beispiel sollten Symbole sich auch von Farbenblinden noch unterscheiden lassen können. Bei den im Repertoire von GTK+ bzw. GNOME enthaltenen Standardsymbolen ist dies weniger ein Problem, da spezielle Icon-Themen mit besonders großen und/oder kontrastreichen Symbolen existieren.

Lokalisierung mit Gettext

Zur Internationalisierung und Lokalisierung von Software mit GNU Gettext ist schon anderswo viel geschrieben worden. Grundsätzlich geht es darum, alle irgendwie Locale-spezifischen Zeichenkettenkonstanten in Aufrufe der Makros _() bzw. N_() zu klammern. Die Konstanten werden in einer .pot-Datei zusammengetragen, von Übersetzern in .po-Dateien für jede einzelne Locale übersetzt, und beim späteren Aufruf des Programmes wird dieses die übersetzten Zeichenketten verwenden.

Um zu vermeiden, dass ein Programm in anderen als der Locale, unter der es entwickelt wurde, unkomfortabel oder unbenutzbar wird, ist es wichtig, gewisse Regeln der Internationalisierung einzuhalten -- z.B. dürfen in Fensterlayouts keine festen Breitenvorgaben vorkommen, da sonst Beschriftungen, wenn sie in der Übersetzung länger werden als im Original, abgeschnitten werden. Auch dürfen Sätze nicht mit Zeichenkettenfunktionen aus Satzfragmenten zusammengebaut werden, da nichts und niemand gewährleistet, dass die Grammatik aller anderen Sprachen genauso funktioniert wie die der Entwicklungssprache.

Dokumentation

Der Quasistandard für strukturierte technische Dokumente, DocBook (mittlerweile in Form von DocBook XML), hat sich in GNOME schon seit Jahren durchgesetzt. Das Format ist detailliert in zahlreichen Quellen dokumentiert und wird von GNOME ohne Modifikationen verwendet, so dass ich es hier nicht weiter zu besprechen brauche. Wichtig ist es jedoch, seine GNOME-Anwendungen nicht nur nach dem GNOME Documentation Style Guide [http://developer.gnome.org/documents/style-guide/] mit einer DocBook-Dokumentation zu versehen, sondern diese dann auch bei der Installation anzumelden.

Hierzu gibt es ein spezielles, ebenfalls einigermaßen standardisiertes Metadateien-Format namens OMF, das unter GNOME von einer Software namens ScrollKeeper betreut wird. Eine sauber geschriebene OMF-Datei, bei der Installation angemeldet, gewährleistet, dass die mühsam geschriebene Dokumentation im GNOME-Hilfesystem überall dort zugänglich ist, wo sie hingehört. Betrachten Sie das folgende Beispiel für eine OMF-Datei; es sollte einigermaßen selbsterklärend sein:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE omf PUBLIC "-//OMF//DTD Scrollkeeper OMF Variant V1.0//DE"
"http://scrollkeeper.sourceforge.net/dtds/scrollkeeper-omf-1.0/scrollkeeper-omf.dtd">

<omf>
 <resource>
  <creator>hweitz@klinkenbuchse.de (Harald Weitz)</creator>
  <maintainer>dzimmer@klinkenbuchse.de (Diana Zimmer)</maintainer>
```

```
  <title>Handbuch für WunderText</title>
  <date>2001-03-27</date>

  <version identifier="2.0" date="2002-09-10" description="Aktualisiert für
   WunderText 2.0"/>

  <subject category="GNOME|Anwendungen|Büro"/>
  <description>Benutzerhandbuch für WunderText 2.0</description>
  <type>Benutzerhandbuch</type>

  <format mime="text/xml" dtd="-//OASIS//DTD DocBook XML V4.1.2//DE"/>
  <identifier url="file://opt/gnome/share/gnome/help/wundertext/C/wundertext.xml"/>
  <language code="C"/>
  <relation seriesid="01ddeea4-0a42-11d6-9cf9-ee43c422358d"/>
  <rights type="GNU FDL" license.version="1.1" holder="Harald Weitz"/>
 </resource>
</omf>
```

Ausblick

Sie werden sicher bemerkt haben, dass ich einen großen Bereich der GNOME-Programmierung in meinem Skript ausgeklammert habe, nämlich die des Komponentenobjektmodells Bonobo. Dieses CORBA-basierte System zum Exportieren und Einbetten von Komponenten und Controls spielt in GNOME 2 eine wesentlich größere Rolle als in GNOME 1 -- Bonobo ist mit der neuen Version unmittelbarer Teil der Kernplattform geworden.

Es würde trotzdem dieses Skript über Gebühr verlängern, hier Bonobo eingehend zu diskutieren, was angesichts der zahlreichen bereits vorliegenden Skripte und Online-Veröffentlichungen von auf diesem Gebiete Kompetenteren hoffentlich zu verschmerzen ist. Es bleibt mir zu wünschen, dass Bonobo Eingang in eine eventuelle zweite Ausgabe oder Fortsetzung von "GNOME 2.0 -- Das Entwicklerhandbuch" [http://www.galileocomputing.de/katalog/buecher/titel/gp/titelID-356] finden können wird.

Ansonsten sei noch darauf verwiesen, dass die nähere Zukunft (d.h. die GNOME-Version 2.4 und eventuelle weitere 2.x-Versionen) für GNOME-Entwickler keine unangenehmen Überraschungen bringen kann, da die binäre Abwärtskompatibilität zwischen GNOME-Versionen mit der selben Ziffer vor dem Punkt stets gewahrt wird.

Unter dem Arbeitstitel "libegg" entsteht jedoch schon seit einiger Zeit ein Sammelbecken von vorgeschlagenen API-Erweiterungen, die im Laufe der nächsten Monate und Jahre ihren Weg in die Plattformbibliotheken finden sollen. Hauptschwerpunkt ist es dabei, eine neue, einheitliche umfassende Lösung für das Programmieren von Menüs und (konfigurierbaren) Werkzeugleisten zu finden, wozu derzeit in GNOME mindestens drei verschiedene Wege existieren.

Als Nebenprodukt der weiteren Konsolidierung der GNOME-Plattform sollen, wie oben schon erwähnt, die Bibliotheken libgnome und libgnomeui in absehbarer Zeit verschwinden oder zumindest ihren Charakter als GTK+-Erweiterungsbibliotheken verlieren, da alle GNOME-spezifischen Widgets irgendwann entweder in GTK+ ‘einwandern’ oder durch äquivalente, neue GTK+-Widgets abgelöst werden sollen.

Nachwort: Zur Ermunterung

Es ist gut möglich, dass Sie dieses Skript und der zugehörige Vortrag mehr von der GNOME-Entwicklung abschrecken als zu ihr hinziehen. Wenn dies an bestimmten technischen Eigenheiten der Plattform liegt, kann ich Ihnen hierzu keinen Rat geben -- vieles ist da verständlicherweise Geschmackssache.

Eines möchte ich jedoch noch loswerden: Der Umfang der Arbeiten, die zu einer kompletten GNOME-Anwendung nötig sind, braucht sie nicht abzuschrecken. Die GNOME-Gemeinde ist riesig, und viele ihrer Mitglieder haben sich mittlerweile zu Spezialisten für Teilgebiete der Anwendungsentwicklung wie Ergonomie, Internationalisierung, Barrierefreiheit oder Dokumentation entwickelt. Wenn Ihre Anwendung gut und nützlich ist, werden Sie keinen der Schritte, der sie zu einer vollwertigen GNOME-Anwendung macht, alleine gehen müssen.