The GNOME desktop environment

Miguel de Icaza (miguel@gnu.org)
Instituto de Ciencias Nucleares, UNAM

Elliot Lee (sopwith@redhat.com)

Federico Mena (quartic@gimp.org)
Instituto de Ciencias Nucleares, UNAM

Tom Tromey (tromey@cygnus.com)

April 27, 1998

Abstract

We present an overview of the free GNU Network Object Model Environment (GNOME). GNOME is a suite of X11 GUI applications that provides joy to users and hackers alike. It has been designed for extensibility and automation by using CORBA and scripting languages throughout the code. GNOME is licensed under the terms of the GNU GPL and the GNU LGPL and has been developed on the Internet by a loosely-coupled team of programmers.

1 Motivation

Free operating systems[1] are excellent at providing server-class services, and so are often the ideal choice for a server machine. However, the lack of a consistent user interface and of consumer-targeted applications has prevented free operating systems from reaching the vast majority of users — the desktop users. As such, the benefits of free software have only been enjoyed by the technically savvy computer user community. Most users are still locked into proprietary solutions for their desktop environments.

By using GNOME, free operating systems will have a complete, user-friendly desktop which will provide users with powerful and easy-to-use graphical applications.

Many people have suggested that the cause for the lack of free user-oriented applications is that these do not provide enough excitement to hackers, as opposed to system-level programming.

Since most of the GNOME code had to be written by hackers, we kept them happy: the magic recipe here is to design GNOME around an adrenaline response by trying to use exciting models and ideas in the applications.

[1] We are using the term "free" in this document to denote "freedom", not price.

The GNOME desktop project is open to ideas and to new programmers. Many good ideas that have originated on the GNOME mailing list have been implemented in the system: GNOME is not static, and we are trying to have fun while we write it, so suggestions regarding making the system more friendly and more powerful for the seasoned Unix user are always welcome. We are not attached to our code: if someone comes up with a better implementation of anything inside GNOME, we apply natural selection and choose the best.

1.1 KDE and the Qt license

GNOME originally was a project to provide free systems with a component model similar in spirit to Microsoft's OLE2 and ActiveX and OpenDOC. This original incarnation of GNOME was not finished nor implemented.

Later, the K Desktop Environment project[12] appeared. KDE aims to provide a user friendly and consistent desktop suite of applications. Sadly, the KDE effort is based on the Qt user interface toolkit which is a non-free library. The main problem is that modification of the toolkit is not allowed. The ability to modify the interface toolkit is a requirement of free software[6] and of the Open Source Software Guidelines[17]. Although Qt is released in source form, permission to redistribute changes made to it is denied.

One option for our project was to write a free implementation of Qt, but this presented three major problems. First, Qt is a moving target. Second, implementing a replacement library to emulate an API is known to involve a lot of work into the "quirk" department: when a library is being written, the internal model used is more often than not different from the original model, and thus many little details and implementation dependencies on the original model will take a lot of work to be implemented properly in the emulation library.

Experience has shown this to be a major problem, and various examples exist: GNUstep[10] cloning the OpenStep API; Wine[19] cloning the Win16 and now the Win32 APIs; and LessTif[13] cloning the Motif API. All of these free software projects are examples of emulation code that has taken several years to develop and which is still not complete. These issues make the emulation approach undesirable.

The final problem was that choosing Qt/KDE would lock us into using C++, which, while not without its merits, has its share of critics too. We preferred to have the flexibility to write our applications in whatever languages we like, and did not want to alienate programmers who dislike C++. We have taken the route of making the GNOME framework accessible through as many programming languages as possible, including C++.

2 What makes up a GNOME application

A number of characteristics make up a GNOME application: a consistent interface; interoperability among the applications; being internationalized and localized; support for network transparency and compliance with existing standards.

Interoperability is achieved in a number of ways: drag and drop support is included in most GNOME applications and, the use of CORBA and scripting is encouraged to provide an extensibility path for GNOME users.

2.1 Look and feel

GNOME applications are supposed to follow the GNOME Style Guide[8]. The Style Guide is intended to ensure a uniform look and feel on the desktop. The Guide is currently rather incomplete, but it is being actively developed. Existing GUI standards make this task easier than it seems.

Currently there are no plans for "GNOME Branding," though this is occasionally mentioned.

2.2 Gtk+

Gtk+[11] is the LGPL toolkit designed by Peter Mattis and Spencer Kimball for use in the GNU Image Manipulation Program (GIMP)[9]. The toolkit has been maintained by a cast of hundreds since it was first released to the public, and it is now the foundation for the GNOME user interface.

Gtk+ is an object-oriented GUI toolkit written in C. Bindings for other languages have been written, allowing applications to use Gtk+ from a range of languages, including Perl, C++, Scheme, Objective-C, and TOM.

Gtk+'s primary objectives are correctness, performance, and usability. By correctness, we mean that the toolkit must be bug-free and must behave as expected. This is reflected in the Gtk+ source by the use of pre-conditions and numerous runtime sanity checks. These checks can be disabled at compilation time for improved performance. This has proven to be very helpful for the application programmer and this practice has been extended to the rest of GNOME.

By performance, we mean that Gtk+ must not consume a lot of resources — it must be small and fast. By usability, we mean that Gtk+ must implement sane, comprehensible and scalable GUI concepts.

Gtk+ rests on a layer of libraries which isolate it from Xlib, and so Gtk+ never calls Xlib directly. This layer is called Gdk, and it is made up of several modules which simplify much of the programming interface to the X Window System and abstract it such that porting Gtk+ applications to another graphics system is just a matter of porting the Gdk layer.

2.3 Gdk_Imlib

The X libraries do not provide an easy way to load and manipulate images from within application programs. Many programs need to load icons and pixmaps, and often need to manipulate these images in a number of ways. Also, many programs need to quantize images to the best color representation that the display can offer. For pseudo-color X visuals, images often have to be remapped to a common palette and dithered for better results.

Gdk_Imlib[7] is a library that complements Gtk+ in this respect, providing a uniform interface to common image manipulation tasks. Loading of images is accomplished through "native" format support via libraries like `libgif`, `libpng`, and `libjpeg`. Gdk_Imlib also has the ability to launch external programs to load images. In fact, if Gdk_imlib is not compiled with native support for a certain format, it resorts to calling external programs to load the requested image. The default external loaders are provided by ImageMagick and the NetPBM suite of utilities.

When loading an image, Gdk_Imlib automatically chooses the best X visual available, and it remaps and dithers images to the corresponding representation if necessary. It also provides convenient functions for image scaling, color correction, cropping, flipping, rotation, and saving via the native libraries or the external programs.

All images displayed in the GNOME desktop are rendered by Gdk_Imlib. This means that GNOME applications are well behaved and do not consume all of the available colors. Color-hungry applications can coexist peacefully with each other while still letting "badly behaved" applications like `xv` and Netscape run without colormap flashing.

2.4 Themes

Support for themes is included in GNOME in various stages: Gtk+ has been enhanced to provide pluggable look and feel modules. Look and feel modules enable GNOME applications to change their look: users can choose a specific look and feel module to make the application's widgets look the way they want. Currently two look and feel modules exist: The default Motif-like appearance and the bitmap-based module. The latter can be configured to use any sort of image achieving spectacular displays with minimal effort.

A theme is made up of a look and feel module and all of its configuration files plus a set of icons, sounds, and desktop images.

2.5 Virtual file system

The virtual file system (VFS) code implements a number of wrappers for the file system system calls. These wrappers create an extended file system space. Once a program starts using the VFS code, access to files across the network by using the FTP or HTTP protocols, and access to files inside various archiving formats will be transparent to the application.

The VFS code is designed to plug directly into existing code, and support for the virtual file system can be toggled at compile time. One existing problem with the VFS code is that wrappers only exist for the system calls, and no code has been written yet to support the `stdio` file system interfaces.

The virtual file system code was originally developed for the Midnight Commander[15] file manager. This file manager is now being used as part of the GNOME desktop environment.

2.6 GNOME application framework libraries

The core development libraries for the GNOME system are divided into the `libgnome` and `libgnomeui` libraries, and provide a convenient framework for application programmers.

`libgnome` provides non-interface related functionality like manipulation of configuration files, portable dynamic linking, non-blocking DNS lookups, global history for "recently used files," consistent command-line argument parsing, and others.

`libgnomeui` provides user interface components that are higher-level than those in Gtk+. These widgets often do work "behind the scenes" in order to keep the user interface consistent across the desktop, and to make the application programmer's life easier.

For example, the `GnomeApp` widget is a top-level window that can be used by applications as their "main" window. A `GnomeApp` window has detachable menu and tool bars, and it will automatically save the state and position of these when the window is closed. The `gnome-app-helper` module provides a convenient API for generating menus and tool bars without the drudgery of creating and organizing the widgets by hand.

A very important part of `libgnomeui` is the `gnome-stock` module. This is a collection of commonly-used icons and pixmaps that applications can use to enhance their visual appearance. Stock icons are provided for common operations such as "New file," "Open file," "Save file," "Quit program," "Cut/Copy/Paste," and "Help". `gnome-stock` also provides functions that create menu items and buttons with these icons. The result is that all applications can share the same set of icons, leading to a consistent look and feel throughout the desktop. The API has been designed to work with themes.

In addition to the stock icons, `libgnomeui` provides functions to create standard dialog boxes and "properties" or configuration dialogs as well. These insure consistency across applications; dialog boxes have a consistent look and feel, and all configuration dialogs operate in a similar way.

GNOME makes use of the standard X session management protocol, and includes a session manager implementation. GNOME applications must be able to save and restore their state when the session manager requests it. This is very convenient for users, as they can simply log out from their session and have it restored when they log back in. `libgnomeui` includes code to make session support easy to add to an application.

A port of Koen D'Hondt's XmHTML[20] widget is part of the GNOME libraries. XmHTML this is a powerful tool that enables a wide range of applications to use HTML to render their information. Currently this is being used in GNOME's mail, news and help browser programs.

2.7 Internationalization

To make GNOME succeed in the desktop, applications will have to support the user's local language. Most computer users are not full-time hackers, and would rather use a system that they can understand, with messages in their language, than a system that only displays English messages.

GNOME is using the GNU `gettext`[4] package to handle internationalization. Currently, translations to Spanish, German, French, Italian, Norwegian, Czech, Finnish, Russian, and Korean have been written and are being maintained.

2.8 CORBA

CORBA, the Common Object Request Broker Architecture, is a mechanism for communication and object sharing between languages, processes, and machines. OMG's[16] CORBA[2] is used as the communication channel between different GNOME components, allowing us to tie many different parts of the system together.

The lack of a reasonably efficient, free CORBA implementation with C mappings has delayed the adoption of CORBA in more parts of GNOME. The free C++ implementations of CORBA all either require too much memory to compile and generate huge stubs, or are difficult to port.

Work has thus begun on the ORBit project, the goal of which is to produce a small and efficient C ORB that works with Flick's optimizing stub compiler[5].

We are planning to use CORBA in two main situations: automation and the GNOME document model. The GNOME document model is currently under discussion and design and is going to provide a framework similar to OLE2[14] for free systems.

2.9 Extension languages

In addition to exporting functionality via CORBA, applications are encouraged to use a scripting language as their native extensibility mechanism: In this manner, programs need not require that a separate process be started to control the program with CORBA. Scheme[1] and Perl[18] are the preferred extension languages for GNOME. We are using the GNU Guile implementation of Scheme.

For example, the GNOME solitaire program has a C-based card game engine. Various solitaire games are written as Scheme programs which define the rules for each game.

2.10 Drag and Drop

No desktop could be complete without drag and drop support, so GNOME offers drag and drop capabilities in as many places as possible. Currently, Gtk+ uses the Xde drag and drop protocol. Programmers are hard at work implementing support for other protocols, including the industry-standard Motif/CDE drag and drop protocol.

2.11 Documentation

The GNOME project is using DocBook[3] as the source format for all documentation. DocBook is an SGML application specifically targeted at writing documentation for programs. We chose DocBook because it seems to be the format with the brightest future; it provides many useful features to the documentation writer, and it is being adopted by many free software projects.

Currently the API documentation that exists is written by hand. However, we are investigating the possibility of extracting this documentation from the source code.

3 The GNOME desktop

The GNOME desktop is launched by the `gnome-session` program. If it is the first time you use the GNOME desktop, a help browser, the panel and the file manager will be started for you. Otherwise, the programs that were active the last time that you were logged in will be launched.

The default GNOME file manager is a port of the ever-popular Midnight Commander file manager to the GNOME environment (you can guess who wrote this paragraph). It has evolved to become a full desktop manager and graphical file manager. All of the features you would expect from a file manager are there, and all of the power of Unix you would expect is provided as well in, well, an intuitive way.

The GNOME help browser is an integrated help browsing program based on HTML. It renders Unix manual pages, GNU Info pages, and HTML pages. GNOME documentation is provided in HTML format, which is automatically generated from DocBook source.

The last default component of GNOME is the Panel; this program provides the user with easy access to installed applications as well as a general-purpose container for various applets. The Panel is not only a boring menu facility, it can embed widgets from applications running in the desktop, and actually works very closely with these other applications through CORBA. This makes the Panel one of the most exciting and extensible navigation systems available.

3.1 Interacting with a window manager

If X11 has done anything, it has given us a diverse collection of window managers. Requiring that people use a GNOME-specific window manager would be asking for a low market share.

However, interactions between GNOME applications and the window manager can be fruitful, so a number of extensions to the existing window managers have been proposed, and GNOME makes use of these extensions to create a better desktop environment. The plan is to have authors of free window managers incorporate these extensions to make their programs become GNOME-friendly.

It is important to note that GNOME applications do not depend on those window manager extensions, but if they are available, they will get used.

3.2 Automation through CORBA

GNOME applications will export their internal functionality via CORBA, allowing other applications or scripting languages with CORBA bindings to make use of their functionality.

For example, when a script is run, it can use CORBA to request that a GNOME application be launched. It could then control the application's behavior and read, modify, and extend the information controlled by the application.

In the example below, a hypothetical Perl script is used to manipulate the GNOME spreadsheet. The GNOME spreadsheet is launched if required, or a reference to the currently executing spreadsheet is fetched. Then, a new empty page is created and filled with information for making a chart. Finally, a chart object is created with a plot of the information and Postscript output is returned in a variable for further Perl processing.

```
$spreadsheet = &gnome_find_impl ("Spreadsheet");
if ($spreadsheet) {
   $sheet = $spreadsheet->new_empty_sheet ();
   for ($i = 0; $i < 100; $i++) {
      $sheet->set_value ("A$i", $i);
      $sheet->set_value ("B$i", $i * $i);
      $chart = $sheet->new_chart ("A1..A100", "B1..B100");
      $chart->set_style ("lines");
      $psoutput = $chart->dump_postscript ();
   }
}
# $ <-- That is a TeX Hack for Emacs font lock mode
```

4 Development

The GNOME development model is pretty close to other free software projects; discussion takes place on public mailing lists, and developers are kept in touch about important changes by means of a private mailing list that has little noise. Developers get access to commit their code directly to the CVS tree. Becoming a GNOME developer is orders of magnitude easier than becoming an accidental parent in high school; just write some GNOME code and you get commit access. Anonymous access to our CVS server has been in-place since just after the project started.

GNOME uses the standard set of GNU development utilities. GNU `autoconf` allows us to easily deal with portability issues, and GNU `automake` takes care of writing correct Makefiles with little intervention from us, the developers.

The GNOME team at Red Hat Advanced Development Labs has implemented a system that does daily compilation and RPM packaging of the most advanced GNOME projects, fresh from the CVS server.

This has served two purposes. First, users have been given a chance to try GNOME on a daily basis and to report bugs, and see how fast we can fix them. Second, it has helped developers acquire the habit of committing working code to the CVS tree. Besides this, a daily status report of the bugs in the code and comments on the GNOME programs is posted to the GNOME web site to inform developers of the known problems in the code. We expect to set up a bug tracking system soon to better track these problems.

5 Current status

The GNOME desktop is currently usable; the desktop environment is now functional and a wide range of utilities have been written.

Effort is now directed towards the big applications and our C-based ORB. Some projects include: Seth Alves's and Chris Toshok's word processor, the presentations program, and the spreadsheet program. The rest of the productivity tools that are missing from free systems are slowly being addressed.

6 Availability

For more information on the GNOME project, you can visit the GNOME web page: `http://www.gnome.org`. GNOME is available in a number of ways:

- AnonCVS: Use CVS to grab GNOME from the following CVS address: `:pserver:anonymous@cvs.gnome.org:/debian/home/gnomecvs`

- FTP: `ftp://ftp.gnome.org/pub/gnome/devel` for the latest snapshots with precompiled binaries.

7 Acknowledgements

There are so many people that need to be acknowledged for various contributions to the GNOME project that we are going to leave someone out, we apologize in advance.

Both Debian and Red Hat have backed up the GNOME effort since the beginning of the project: Red Hat Advanced Development Labs is currently spending their resources into developing GNOME. An endless list of GNOME hackers, coders, beta testers, documentation writers, artists, and friends have made the GNOME project a reality. The GIMP/Gtk+ authors have helped the project a lot, and both Novare and VA-Research have provided GNOME with network resources to hold our CVS server. The Instituto de Ciencias Nucleares UNAM originally provided the FTP and mailing list service. Thanks also to Ariel Faigon at SGI and SGI for lending Miguel an Indy machine to write free software. Thanks to Nat Friedman for helping us with this paper and to Elliot Lee for starting a new project on a daily basis.

References

[1] William Clinger, Jonathan Rees, et al. "Revised^4 Report on the Algorithmic Language Scheme," 1991.

[2] The Common Object Request Broker: Architecture and Specification. The Object Management Group (OMG), Revision 2.0, July 1995 (`http://www.omg.org/corba`)

[3] The Davenport Group, maintainers of the DocBook DTD:
`http://www.oreilly.com/davenport/`

[4] Drepper, Ulrich. "Internationalization in the GNU project". Appeared in the *Proceedings of the First Conference on Freely Redistributable Software*, 1996.

[5] Eric Eide, Kevin Frei, Bryan Ford, Jay Lepreau, Gary Lindstrom. "Flick: A Flexible, Optimizing IDL Compiler". Appeared in the *Proceedings PLDI '97*:
`http://flux.cs.utah.edu/flux/flick/`

[6] The free software philosophy:
`http://www.gnu.org/philosophy/free-sw.html`

[7] Gdk_Imlib: `http://gnome.labs.redhat.com/imlib/`

[8] GNOME Style Guide:
`http://www.gnome.org/devel/sg`

[9] GNU Image Manipulation Program (GIMP):
`http://www.gimp.org`

[10] The GNUstep project:
`http://www.gnustep.org`

[11] The Gtk+ toolkit:
`http://www.gtk.org`

[12] The K Desktop Environment:
`http://www.kde.org`

[13] The LessTif project:
`http://www.lesstif.org`

[14] Microsoft OLE2 programmer's reference. Microsoft Press, 1993.

[15] The Midnight Commander:
`http://www.blackdown.org/mc`

[16] Object Management Group (OMG):
`http://www.omg.org`

[17] Open Source Software Guidelines:
`http://www.opensource.org/osd.html`

[18] Larry Wall and Randal Schwartz, "Programming Perl," O'Reilly and Associates, Inc, 1991.

[19] The Wine project:
`http://www.winehq.com`

[20] The XmHTML widget:
`http://www.xs4all.nl/~ripley/XmHTML`