GNOME Technologies

A Brief Description of The Technology Behind GNOME

David C. Mason
Red Hat, Inc.

dcm@redhat.com

Introduction

GNOME provides the user with a rich, mature desktop environment in which they can use their applications. Behind the graphical front end of this environment is a robust "structure" that allows developers to create applications in less time while still using tools that allow their applications to remain on the forefront of computer technology.

This paper will briefly detail the base technologies that GNOME provides for application developers. For all of these technologies there are whitepapers and reference documentation available that go into greater detail. You can find this documentation and other resources at http://www.gnome.org.

The technologies detailed in this paper are: GTK+, the relationship between GNOME and window managers, CORBA and ORBit, and the GNOME Imaging Architecture.

GTK+

GTK+[1] is a mature, efficient widget set. It provides a modern look-and-feel while remaining flexible and efficient. GTK+ provides the application developer with a powerful object system that has the ability to dynamically query and modify objects and widgets at run—time. GTK+ is an extensible toolkit; for example, GNOME has built-in, high-level widgets to speed up the GNOME application development cycle. The user will enjoy GTK+ because of the powerful theme engine that allows them to change the look of all GTK+ applications with one easy-to-use tool.

There are several reasons to use GTK+ including the licensing, the language bindings, internationalization, and the drag and drop support. Each of these are explored in more detail here.

The License

The license covering the GTK+ widget set is the Library General Public License (LGPL)[2]. This license is very similar to the General Public License (GPL)[3], except that you may link any application to the widget set no matter if that application is free or proprietary software.

Language Bindings

Because of the open nature of GTK+ development, many people have contributed language bindings for GTK+. There are already many languages besides C that can be used to create a GTK+ application. These languages include: C++, Perl, Python, Pascal, Objective C, Guile, and Ada.

Internationalization

GTK+ supports international[4] text display and input by utilizing the X Input Method extension. Because of such support, your applications can immediately be used with languages that require non-Roman character sets, such as those in Asia.

Drag and Drop

GTK's drag and drop[5] functionality supports both the Xdnd[6]and Motif[7]drag and drop protocols which allow for use of drag and drop functionality with other widget sets that support these protocols. Applications written in widget sets such as Motif or Qt that correctly support these protocols will also be able to share drag and drop functionality with GTK+ applications. This support is a breakthrough in providing the user backwards compatibility with their older applications.

GNOME, the Window Manager, and X

One aspect of GNOME that new users have a hard time understanding is the relationship between GNOME and the window manager and GNOME and X. To understand this relationship you must break down the three components.

The X Window System is the software that allows graphical elements to be drawn to the screen. This is the base of the whole system and provides the libraries and technologies to make the graphical display possible.

The window manager is the software that essentially defines the way windows are placed on the screen and how the window borders look and act.

GNOME is a desktop environment that runs on top of the X Window System and a window manager. GNOME was designed to not be dependent on any one window manager. Therefore, the user can have a choice in which window manager to use. GNOME provides a number of "hints" to let the window manager to work with it in a seamless manner. If a window manager contains these hints it is considered "GNOME Compliant.[8]" Currently there are a few window managers with all or some GNOME compliancy.

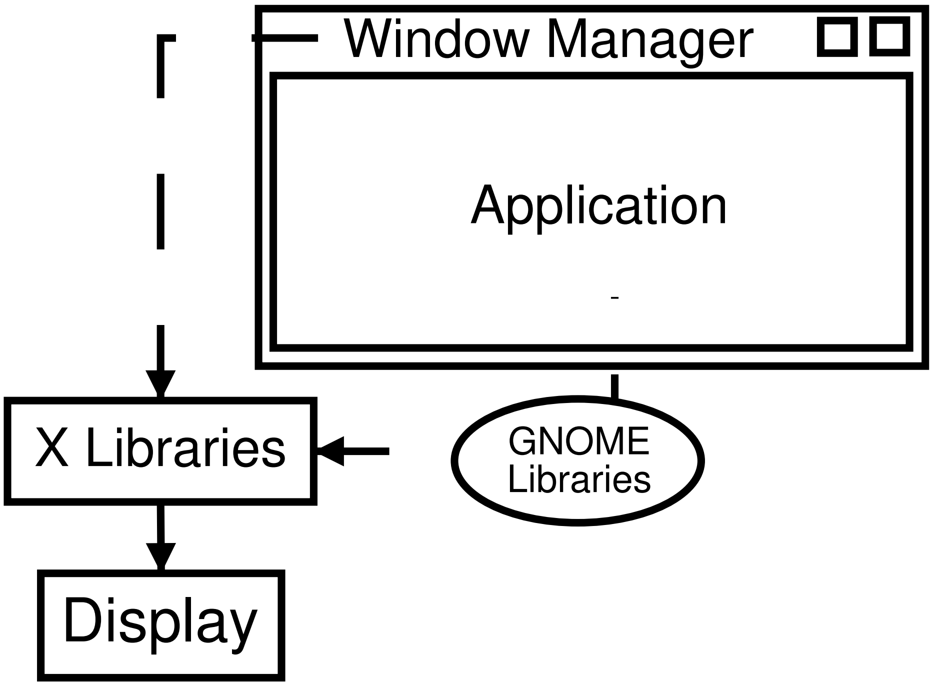

Figure 1. GNOME, X, and the Window Manager working together.

CORBA in GNOME

GNOME provides intercommunication for applications through CORBA. CORBA (Common Object Request Broker Architecture) was started by the Object Management Group [9](OMG) to allow interoperability among the many available hardware and software products.

CORBA is designed to allow applications to communicate with each other, no matter where they are located (local or network) or how they were implemented (language/operating system).

For example, a program written in the C language running on a Linux system would be able to use CORBA to send a request to a program written in Python running on a Windows system, without ever having to worry about the programming language or operating system in use.

For this communication to work there is an important piece that has to be present—the Object Request Broker (ORB). Simply stated, the ORB is the middleware that allows the communication to occur between the two applications.

ORBit

ORBit [10]is an ORB that was adopted by the GNOME project. When the decision was first made that CORBA would be implemented in GNOME, there were a few ORBs that could have been used. Most ORBs had licensing problems and were quickly left out.

Then, a couple of hackers wrote a small, lightweight ORB called ORBit. ORBit has developed rapidly, and once there was a stable release, many GNOME applications started to take advantage of it.

ORBit fit well into the GNOME project primarily because it does not have any licensing problems. Like GTK+, it is licensed under the GNU LGPL.

Secondly, ORBit is resource-friendly. It is a very fast implementation of CORBA and has proven to be one of the lightest ORBs available in terms of system resource usage.

Finally, because ORBit is a free software project, the development has been rapid enough to quickly meet the needs of the GNOME project.

GNOME Imaging Architecture

The GNOME imaging architecture provides developers with tools for creating interactive graphics displays, high-quality printing, and typographic-quality fonts. There are four components in the GNOME Imaging Architecture, and although not all of them are completed, they promise a development environment that has never been offered in the Linux and UNIX worlds.

The four components of the GNOME Imaging Architecture are: the Canvas, Libart, GNOME Print, and GNOME Fonts. At the writing of this paper the canvas and Libart are complete and GNOME Print and GNOME Fonts are still being developed.

GNOME Canvas

The GNOME canvas[11] is a high-level display engine for creating structured graphics. A canvas displays collections of items that can be lines, rectangles, elipses, and text.

The canvas is an extensible, high-level, object-oriented drawing widget. Applications can use the basic drawing primitives in the canvas to create interactive displays, or define their custom canvas item types for complex displays.

The GNOME Canvas can render the images in two different modes. One mode uses Xlib to render in a fast but lower quality method. The second mode uses Libart to render a very high quality image, but requires more system resources to do so.

Libart

Libart is a rendering engine that provides a superset of the PostScript imaging model, with the addition of alpha transparency and native anti-aliasing. Libart is a high-quality rendering engine that uses a combination of rendering technologies to enhance performance; for example, microtile arrays and sorted vector paths.

A microtile array is a grid-like data structure that lets the canvas keep an efficient representation of the regions that need to be repainted.

With Libart, the programmer can create a vector path or a Bézier path, and later ask the library to create a sorted vector path for performance. A sorted vector path is a vector path whose segments have been sorted in a monotonically-increasing Y order. This allows for more efficient rasterization or scan conversion.

Libart also utilizes GdkRGB, which creates a 24 bit rendering space for all images, no matter what bit depth the final display will be. If the image needs to be displayed in a lower bit depth it will still be calculated in a 24 bit, RGB space prior to display. The image then goes through a conversion to adjust it to the X visual depth. Using this method increases the performance in rendering. Performing the calculations in 24 bit depth allows for the highest quality image available to the user.

Conclusion

The GNOME Desktop Environment is more than just a way of displaying your applications within nice looking graphics. GNOME provides a firm foundation for application developers.

From the widgets provided by GTK+ to the intercommunication provided by CORBA, GNOME allows developers to create applications that can be on the front edge of Linux and UNIX technologies.

Notes

1. GTK+ homepage (http://www.gtk.org)
2. The GNU hompage (http://www.gnu.org)
3. The GNU hompage (http://www.gnu.org)
4. GTK+ Internationalization whitepaper (http://www.gnome.org/white-papers/i18n/gtki18n/)
5. GTK+ Drag and Drop Whitepaper (http://www.gnome.org/white-papers/DND/DND/)
6. The Xdnd protocol (http://www.cco.caltech.edu/~jafl/xdnd/)
7. Motif (http://www.opengroup.org/desktop/)
8. GNOME Window Manager Compliance (http://www.gnome.org/devel/gnomewm/book1.html)
9. OMG homepage (http://www.omg.org)
10. ORBit (http://www.labs.redhat.com/orbit/)
11. GNOME Canvas whitepaper (http://www.gnome.org/white-papers/Canvas/canvas/)