Study of Development Tools for Developing an Interactive Graphical Optical Music Recognition System

Michael Droettboom

April 26, 2000

First Draft

Abstract

This document examines four graphical user interface toolkits and assesses their suitability to our needs: the development of interactive, graphical optical music recognition system.

1 Introduction

There are many GUI toolkits available. On UNIX systems, a single standard has yet to emerge, unlike proprietary operating systems such as Windows and MacOS. Online, "The GUI Toolkit Framework Page" provides an exhaustive list of toolkits: past, present and future [Tai].

I have selected four toolkits for intensive study: GTK and friends, Qt, Java Swing and wxWindows. All of these toolkits are freely available, and some are open-source. The overall quality is extremely good: All of them have been used in the development of major applications. Graphical user interfaces have been around long enough that the basic set of features a toolkit must provide is more or less standard. It is therefore more useful to examine the fundamental issues of each system, rather than to perform a widget-by-widget comparison.

1.1 Library hierarchy

Within the general category of graphics toolkits, there is a heirarchy of three separate levels of libraries that this project requires.

1. **Graphics drawing**

   This is the lowest-level of the three categories. It provides tools for building complex graphics out of graphics primitives, such as lines, arcs and characters. This level should not be confused with very-low-level graphics libraries, such as XLib, which deal merely with getting pixels on the screen.

Since the present project will deal with the difficult and specific problem of Common Music Notation (CMN), this level is the most critical.

Libraries at this level include **Adobe Display Postscript**, **GnomeCanvas**, **Java 2D**, **Libart**, **OpenGL (Mesa)**, **QPainter**, **TkCanvas** and **wxPostscriptDC**.

All of these libraries can express roughly the same domain of graphics as Postscript. Of these, only **GnomeCanvas** and **TkCanvas** support structured graphics: treating graphics in terms of live objects rather than static one-shot drawings on a grid of pixels.

2. **Widgets**

   This is often referred to as the windows, icons, mouse and pull-down menus (WIMP) level. This level provides the user-interface elements that are consistent across virtually all GUI applications: listboxes, menus, textboxes, checkboxes, radio buttons etc. WIMP interfaces have been around long enough that there is a standard set of widgets that all of the toolkits reported here provide.

   Libraries at this level include **GLUI**, **GTK+**, **Qt**, **Swing**, **Tk**, **wxWindows**.

3. **Application framework**

   This is the highest-level. Application framework libraries are designed to minimize the work of creating complete GUI applications. This often helps applications conform to UI conventions, such as automatically creating pull-down menus with the conventional names in the conventional order. They may also provide compound widgets, such as wizards, calendars, spreadsheets or standard dialog boxes for opening and saving files. Some frameworks include facilities for printing, usually based on Postscript. An "application command framework" can ease the implementation of undo/redo. Lastly, application frameworks provide the means by which applications can communicate to other applications, though often only to applications created using the same application framework. Though not strictly necessary for our needs, "if we got 'em, let's use 'em."

   Application frameworks include **GNOME**, **KDE**, **wxWindows**, and **Java Platform**.

While some libraries are "hard-wired" to other related libraries, many can safely be mixed-and-matched. Tables 1 and 2 show the compatibilities of the various tools.

1.2 Object-oriented programming and the model, view, controller paradigm

While the jury on object-oriented vs. imperative languages is still out on some issues, when it comes to GUI development, object-oriented programming (OOP)

	GLUI	GTK+	Qt	Swing	Tk	wxWindows[1]
Display Postscript		•	•		•	○
GnomeCanvas		•				○
Java 2D				•		
Libart		•				○
OpenGL (Mesa)	•	•	•	•	•	•
QPainter			•			
TkCanvas					•	
wxPostscriptDC						•

Table 1: Compatibility matrix of graphics drawing libraries to widget libraries.

	GNOME	KDE	wxWindows	Java Platform
GLUI	•	•	•	
GTK+	•		○	
Qt		•		
Swing				•
Tk				
wxWindows			•	

Table 2: Compatibility matrix of widget libraries to application frameworks.

is the obvious choice: It is very natural for visual objects to inherit properties from other visual objects.

Note that OOP is a programming style, not a language feature. One could write in an object-oriented style in any language. However, for OOP to be truly useful, the language syntax needs to be expressive and convenient [Swi93].

The most powerful use of OOP in GUI progamming is to support the model, view, controller (MVC) paradigm. In an MVC application, every object is either a model (internal data representation), view (displays the model to the user) or controller (allows the user to manipulate the model). When the user engages a controller, it sends a message to its associated models. The models, in turn, send messages to their associated views. The power of the system is in the ways multiple models, views and controllers can be inter-related. In a music editor, there might be two different views of the same score showing: one in full score and one showing just an extracted part. Changes to one will automatically affect the other. One could also have multiple controllers operating on the same model: the MIDI keyboard, QWERTY keyboard and mouse could all manipulate the same data.

One of the considerations in selecting a GUI toolkit is how well is supports OOP and the MVC paradigm.

1.3 Programming methodologies

An increasingly common method of GUI development is to implement high-level features in a high-level dynamic scripting language, such as Python or Scheme, and implement low-level, performance-critical features in a system language such as C. This is often referred to as Rapid Application Development (RAD). Software developed in this style tends to be more flexible, have fewer bugs and be delivered sooner [Lut97]. The penalty in performance is minor in GUI front-ends, where most of the time is spent waiting for the user. In applications where the fundamental data structure is highly dynamic and heterogenous, performance can often increase by using a well-implemented scripting language [Lut97].

A real-world case study in this methodology is the UNIX vector graphics program Sketch by Bernhard Herzog [Her]. "Sketch is written almost completely in Python, an object oriented interpreted programming language. Python can easily be extended by modules written in C to increase performance and Sketch uses this fact to implement some time critical functions and Python objects in C." The original version used the Tkinter GUI toolkit, (which adds an additional layer of Tcl scripting!) though the current development version is written for GTK+. The program speaks for itself: it works and performs well. If that isn't enough, Herzog wrote an illuminating and convincing report about his experiences with this methodology [Her].

A compromise to RAD, when performance is paramount over flexibility, is Rapid Prototyping. In this methodology, a prototype of the system and its important algorithms are first written in a flexible, dynamic scripting language and then transitioned to a systems language. In this way, all logical bugs in the algorithms can be found easily and without the complications of debugging compiled code. Using a GUI toolkit that can be used identically in both settings would greatly ease the transitioning.

Support for scripting languages would be a major asset to our choice of GUI toolkit.

1.4 Building dialog boxes

All of the toolkits here have interactive dialog builders, some better than others. The use of such tools is a matter of personal taste. Personally, I prefer coding dialog boxes manually: I find it makes them easier to update. To aid in that approach, all of the toolkits here also support automatic reformatting of dialog boxes when they are resized.

2 The toolkits

This section examines each family of toolkits in turn, discussing history, licensing, portability, documentation, support and language issues. A summary is presented in Table 3.

2.1 GNU Tools

GNU Network Object Model Environment (GNOME) [GNO], Gimp Toolkit (GTK) [GTK] and Libart are all officially part of the GNU's Not Unix (GNU) [GNU] project. GTK+ was originally developed for the widgets in the Photoshop clone, The GNU Image Manipulation Program (Gimp). It was later chosen as the official widget library for the GNOME project. GNOME aims to create an open-source desktop environment for UNIX[2]. Real-world applications developed using these tools include the Gnumeric spreadsheet, Gimp and Sketch. In recent years it has become the standard among open-source afficianados.

While GTK+ has been ported to Microsoft Windows, GNOME and Libart currently only compile under UNIX.

> **I'm running Microsoft Windows, can I get GNOME?**
>
> Currently the answer is no, GNOME hasn't been ported to the Windows platform, there is an effort going on to make such a port possible. If you really really need a version for Windows, count on doing lots of work and coordinating with related efforts.
>
> Tor Lillqvist has been porting several graphical libraries used by GNOME to the Win32 (Microsoft Windows 95/98/NT) platform. More info on his efforts can be found at http://www.gimp.org/~ tml/gimp/win32. He's already ported glib, gtk+, gdk_imlib, and a few other libraries involved in generating graphics to Win32.
>
> His goal is to get Gimp working fully on the platform (it already works partially), but his work brings GNOME much closer to being portable to Windows. A number of GNOME applications need just his libraries (possibly with some more debugging), libpng, gnomelibs and ORBit. If someone were to port these, many GNOME applications could be compiled and run on Windows. *[From the GNOME FAQ.]*

The GnomeCanvas, for structured graphics, is based on Libart, which would be very difficult to port because it is so heavily optimised for the X-Windows client/server model. Note also that no ports to MacOS have even been discussed.

GNOME and GTK+ are distributed under the Lesser GNU Public License (LGPL)[3]. This allows developers to freely use its source-code for open-sourced or commercial products. However, all modifications to the library itself must remain open-source.

The documentation of is rather patchy. While a great tutorial introduction exists [Pen99], The GNOME Reference Documentation Project is just getting started. As with most open source projects, the only real source of information

[2] In this document, 'UNIX' refers to all UNIX variants currently in wide-spread use, Linux, Solaris, AIX, IRIX etc., running X-Windows.

[3] I'm not a lawyer, so I stand corrected on any of this licensing stuff

is mailing lists and newsgroups. Commercial support is available from Helix Code and RedHat Labs.

GTK+ and GNOME are written in a controversial proprietary object-oriented style of C. Not only is this style a major coding headache compared to proper object-oriented languages[4], it is less efficient than C++ since all object types are dynamically bound. The only advantage of the system is the ease with which support for new languages can be added. In fact, the number of supported languages is unparalleled: All the important ones are there, (C, C++, Python, Scheme, Perl) as well as some more esoteric ones (Pike, Haskell, Ada95, Eiffel etc.). Full GNOME, however, can currently only be used with C, C++, Python and Scheme. GTK–, adds proper C++ object-orientivity to GTK. MVC is not built-in to the system and would have to be implemented manually.

2.2 Qt and KDE

Q Toolkit (Qt) [Qt], from Troll Tech, began as a commercial toolkit for cross-platform development. It is the official widget library of K Desktop Environment (KDE), a project to create an open-source desktop environment for UNIX.

After years of pressure from users and heated competition with GNOME, Troll Tech has released the UNIX version of the Qt source-code under a restricted open-source license. The 'QPL' allows use of the Qt source-code by developers of other open-source software. All modifications to the source code become the property of Troll Tech. In the event that Troll Tech discontinues their support of Qt, Qt will fall under the open BSD license. The Microsoft Windows version of Qt remains a commercial product, costing $1550 USD. Portability is achieved by drawing all widgets directly, rather than relying on "native" widgets. Qt also abstracts non-graphical operating system features to ease porting across platforms.

Real-world Qt applications include an entire office suite, KOffice, and Netscape Communicator 4.x.

Since Qt is maintained by a private company, the documentation is clean and thorough. Commercial support is also available.

Qt is written in C++, but requires a special preprocessor to handle its unique and convenient event system. This is not a problem in UNIX, where it is easy to preprocess source code before compiling, but adds some headaches when using Microsoft Visual C++. MVC is intergral to the design of Qt. PyQt adds Qt support to Python.

2.3 Java

Java [JFC] is very well-suited to networking and cross-platform development. It's original GUI API, the Abstract Windowing Toolkit (AWT), took a lowest-common denominator approach to cross-platform compatibility, and thus was not very powerful. The new Swing API, is one of the most powerful and flexible

[4] A point was made on the on-line tech-news site, Slashdot, that one could write in object-oriented assembler, but that doesn't make it right.

GUI toolkits ever created. The lower-level API, Java 2D was developed jointly by Adobe and Kodak. It runs identically, including look-and-feel, on all platforms with a Java 1.2 virtual machine. This currently includes UNIX, Microsoft Windows and MacOS.

All of the development tools necessary to use Java Swing are freely available. The source-code was recently made available after complaints that Sun Microsystems was exercising too much control over Java.

A good example of a complex JFC application is Borland's JBuilder 3 IDE, written entirely in open-sourced Java.

The documentation is excellent, and there are loads of third-party books on the subject. Support is available from Sun Microsystems and IBM.

The well-designed Java language itself is probably JFC's biggest asset and biggest liability. It is the most flexible of all the tools reviewed here because of the ways Java allows you to build object heirarchies. MVC is fully utilized throughout the system, but is optional. Scripting of JFC is also available by using JavaScript or JPython. However, interest in Java has wained in recent years due to its poor performance and the increasing control of Sun Microsystems.

2.4 wxWindows

Like Qt, wxWindows also aims to be a cross-platform solution. It began as an experiment in 1992 by Julian Smart at the University of Edinburough [wxW]. Unlike Qt, it only provides thin wrappers around other widgets: Under UNIX, it uses GTK+ widgets. On Microsoft Windows and MacOS it uses the standard "native" widgets. In its initial release, wxWindows was plagued with the same problem as Java's AWT: it only supported widgets that were common across all platforms and it was difficult to inherit from and modify them. Release 2 changes all that, with additional widgets implemented from scratch where necessary. Like Qt, it also abstracts basic operating system features across platforms.

wxWindows is fully open-sourced under the LGPL.

JAZZ, a MIDI sequencer for Linux, was implemented using wxWindows.

The documentation is complete, if at times too concise. There is no official support available, only mailing lists.

wxWindows is written in and supports pure C++. It uses the Document-View model, a Microsoft bastardisation of MVC. wxPython provides Python support.

2.5 Side option: OpenGL

OpenGL[Ope] is a graphics library specification from Silicon Graphics. While commonly used for 3D graphics, OpenGL can handle sophisticated 2D graphics just as well.

Mesa is an open-source implementation of OpenGL licensed under the GNU Public License (GPL).

GLUI is an open-source widget set that uses OpenGL as its rendering engine.

	GNU	Qt	JFC	wxWindows
License	LGPL	QPL (UNIX); $ (MSWin)	Sun Source	LGPL
Platforms	UNIX; MSWin (GTK+ only)	UNIX; MSWin	UNIX; MSWin; MacOS	UNIX; MSWin; MacOS
Case studies:	Gnumeric, Gimp, Sketch	KOffice, Netscape Communicator	Borland JBuilder 3	JAZZ
Documentation	Incomplete	Superb	Superb	Adequate
Support				
Mailing list	YES	YES	YES	YES
Newsgroup	YES	NO	YES	NO
Commercial	Helix Code; RedHat Labs	Troll Tech	Sun, IBM	NO
Language				
Base language	C with object-oriented style	C++	Java	C++
Systems languages	Ada95, C, C++, Eiffel, Objective-C, Pascal	C++ (with proprietary preprocessing)	Java	C++
Scripting languages	Scheme (Guile), JavaScript, Perl, Python	Scheme (Guile), Python	JavaScript, JPython	Python
MVC	NO	YES	YES	YES

Table 3: Summary of GUI Toolkits

As much as the potential is there, I could not find a single reference to a 2D graphics application written using OpenGL.

3 Conclusion

If portability beyond UNIX is not required, GNOME/GTK is the best choice, if only for GnomeCanvas' structured graphics. Some annoyances remain, such as poor documentation, a silly object-oriented syntax (which can be avoided with GTK–), and lack of built-in MVC.

If portability is required[5], JFC and wxWindows are both excellent choices. Their lack of structured graphics, however, means a lot more effort would be involved in implementing the core notation engine. Experiments with embedding the GnomeCanvas in wxWindows look promising, but do not change the fact that GnomeCanvas is inherently difficult to port.

Qt has no advantage over the other toolkits, particularly due to its price.

References

[GNO] Gnome website. http://www.gnome.org/.

[GNU] Gnu project website. http://www.gnu.org/.

[5]MacOS support would be useful if we choose GUIDO as our music language and wish to interface with NoteAbility Pro.

[GTK] Gtk+ website. http://www.gtk.org/.

[Her] Bernhard Herzog. Sketch drawing program. http://sketch.sourceforge.net/.

[JFC] Java foundation classes website. http://www.java.sun.com/jfc/.

[Lut97] Mark Lutz. *Programming Python*. O'Reilly and Associates, Sebastopol, CA, 1997.

[Ope] Opengl website. http://www.opengl.org.

[Pen99] Havoc Pennington. *GTK+/Gnome Application Development*. New Rider, Indianapolis, IN, 1999. Open-source, available on-line.

[Qt] Troll tech website (home of qt). http://www.troll-tech.com/.

[Swi93] Robert Switzer. *Eiffel: An Introduction*. Prentice Hall Object-Oriented Series. Prentice Hall, 1993.

[Tai] Andy Tai. The gui toolkit, framework page. web page. http://www.geocities.com/SiliconValley/Vista/7184/guitool.htm.

[wxW] wxwindows website. http://www.wxwindows.com/.