Reprinted from the

Proceedings of the Linux Symposium

July 23rd–26th, 2008
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
Matthew Wilson, *rPath*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
Eugene Teo, *Red Hat, Inc.*
Kyle McMartin, *Red Hat, Inc.*
Jake Edge, *LWN.net*
Robyn Bergeron
Dave Boutcher, *IBM*
Mats Wichmann, *Intel*

Choosing an application framework for your Linux mobile device

Shreyas Srinivasan and Phaneendra Kumar
Geodesic Information Systems
shreyas@geodesic.com

Abstract

Application Frameworks define the user experience. For consumer mobile devices, choosing a feature-rich and high-performance application framework becomes a premium. Creators of Linux mobile devices have a range of application frameworks (gtk/qt/efl) to choose from, but this choice also makes it hard to pick a framework which suits a specific set of requirements.

This paper evaluates various open source application frameworks and their underlying technologies. It also explains performance benchmarks of the frameworks on different types of hardware and the capability of the framework to use specific hardware features to improve performance.

The Application frameworks which will be evaluated are Gtk/Gnome, QT/KDE, Clutter/Tidy, and EFL/E. The talk will present performance benchmarks of these frameworks on Omap, Freescale, and Intel mobile processors.

1 Introduction

User interfaces have become an important factor to decide the popularity and success of a consumer device. The launch of the IPhone has showed the importance of a well designed and intuitive user interface, and has heightened expectations of users all over the world. Increasingly, most processors have built-in floating-point processors and support open standardes like Open GL ES, so the ability to support fluid interfaces and animations exists.

Processors for embedded devices have largely been dominated by ARM-based processors, but with X86 processors improving rapidly, it is important to understand the current features and capabilities of each of these architectures.

ARM

The ARM architecture has always dominated the embedded device market due to its low power consumption. There have been various versions of ARM over time, but with performance improvements plateauing, most ARM processors have chosen to add co-processors to provide specialized performance to applications. Some of the widely used embedded processors based on ARM are:

- **OMAP**
  The Omap series of processors are based on ARM Cortext A8. Different families can have co-processors like PowerVR SGX 530 2D/3D and a DSP Video accelerator. The OMAP 2430 processor range is used by Nokia for their internet tablets.
- **Freescale I.MX**
  The I.MX range of processors by Freescale is based on an ARM 1136JF-S core. Different families have options of different co-processors like the VFP11 numeric processor, IPU, H.263/MPEG4 encoding accelerator, and ARM MBX R-S graphics accelerator. The I.Mx31 processor is used in the Microsoft Zune.

X86

The X86 architecture has always been seen as the one valid for desktops but unsuitable for embedded devices, mainly due to power consumption. With the Menlow family of processors, X86 processors can finally compete with ARM ones, even in power consumption.

- **Pentium Mobile**
  The Pentium mobile range of processors are X86-based processors with 1 GHz processor, a built-in FPU, and power consumption of 14.44 Watts.

These are mainly used for laptops, but some devices like the Founder also use this chip because of its very high performance.

- **Menlow**
  The Menlow (Atom) series of processors are the first of Intel's chips to foray into low power for embedded hardware. The major difference is removal of predictive instruction execution; this reduces power consumption. The Menlow range of processors range from 800 MHz to 1.6 GHz, have a built-in floating point unit, and consume just 5W of power.

The trends of hardware means different requirements for application platforms depending on the underlying architecture.

2 Characteristics of a good Application Framework

Timeline and Animation Support. Creating intuitive and fluid interfaces requires a state-aware canvas which can move different objects over a sequence of coordinates with regards to time.

Hardware support. The ability to use specific hardware features to increase performance is of premium importance. Embedded hardware-based rendering has been standardized around OpenGLES. This is particularly important on the ARM architecture, where graphics performance can be considerably improved by using the Graphical Processing Unit and conserving the relatively low computational power of the CPU.

Multi Language Bindings. Multi Language bindings make the application framework viable to a wide variety of programmers.

Email Libraries. Email is one of the core applications. Email libraries which support a range of protocols like POP, IMAP, and Exchange are of great importance.

Browser Support. Support for rendering and embedding web pages is a powerful feature which enables applications to enrich the user experience by supporting local and cloud-based applications.

Multimedia. Capability of rendering video and audio.

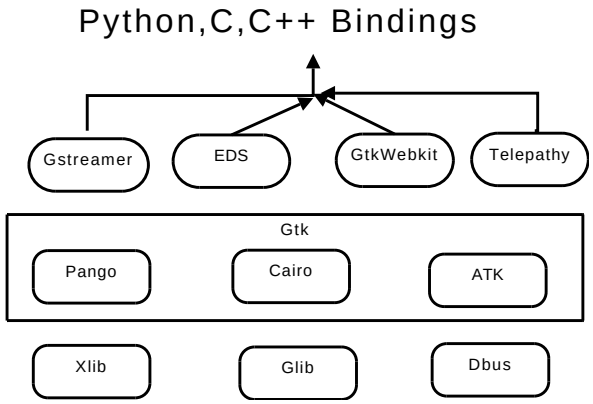

Figure 1: Architecture of Gnome Mobile Application Framework

Inter Process Communication. Communication between various applications helps build a good user experience.

This paper evaluates multiple application frameworks and their performance in some of the aforementioned categories.

3 Application Frameworks

Gnome Gtk

The GNOME Mobile Platform is a subset of the proven, widely used GNOME Platform. The platform definition represents components that are currently shipping in production devices.

- **Components**

  1. **Cairo**
     Cairo is a 2D graphics library with support for multiple output devices. Currently supported output targets include the X Window System, Quartz, Win32, image buffers, PostScript, PDF, and SVG file output. Experimental backends include OpenGL (through glitz), XCB, BeOS, OS/2, and DirectFB.
  2. **EDS**
     Evolution Data Server is a PIM server which manages access to calendar, addressbooks, and tasks. All these items are served over Dbus.

3. **GtkWebKit**
   WebKit/GTK+ is the new GTK+ port of the WebKit, an open-source web content engine that powers numerous applications such as web browsers, email clients, feed readers, and web and text editors.
4. **GStreamer**
   GStreamer is a library that allows the construction of graphs of media-handling components, ranging from simple Ogg/Vorbis playback to complex audio (mixing) and video (non-linear editing) processing. Applications can take advantage of advances in codec and filter technology transparently. Developers can add new codecs and filters by writing a simple plugin with a clean, generic interface.

- **Advantages**
  1. Existing precedent of devices which ship with this platform.
  2. Well defined roadmap and enthusiastic developer community.
  3. High profile industry support.
  4. Focus on minimal footprint.
  5. Non-free codecs can be licensed on top of gstreamer and shipped legally.

- **Disadvantages**
  1. No current support for offscreen rendering.
  2. Cairo OpenGL backend is extremely unstable.
  3. Gobject API has a steep learning curve.

EFL E

- **Components**
  1. **Evas**
     Evas is a hardware-accelerated canvas API for the X Window System that can draw anti-aliased text, smooth super and sub-sampled images, alpha-blend, as well as drop down to using normal X11 primitives such as pixmaps, lines, and rectangles for speed if your CPU or graphics hardware is too slow.

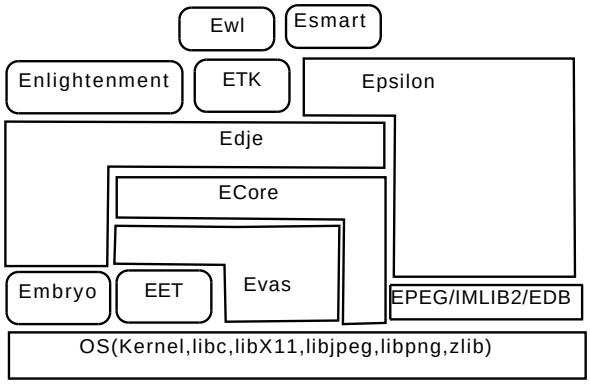

Figure 2: Architecture of Enlightenment Foundation libraries

  2. **Ecore**
     Ecore is the core event abstraction layer and X abstraction layer that makes doing selections, Xdnd, general X stuff, and event loops, timeouts, and idle handlers fast, optimized, and convenient. It's a separate library so anyone can make use of the work put into Ecore to make this job easy for applications.
  3. **Edje**
     Edje is a graphical design and layout library based on Evas that provides an abstraction layer between the application code and the interface, while allowing extremely flexible dynamic layouts and animations.
  4. **EWL**
     The Enlightened Widget Library (EWL) is a high-level toolkit providing all of the widgets you'll need to create your application. The expansive object-oriented-style API provides tools to easily expand widgets and containers for new situations.
  5. **Emotion**
     Emotion is a library providing video-playing capabilities through the use of smart objects. Emotion is based on libxine, a well established video playing library, and so supports all of the video formats that libxine supports, including Ogg Theora, DiVX, MPEG2, etc.

- **Advantages**
  1. Small Memory footprint
  2. Evas is a state-aware canvas which supports timeline-based animations.

3. Evas has an OpenGL backend and hence can be hardware accelerated.

- **Disadvantages**

1. Long release cycles.
2. Rapidly changing mainline; makes it hard to keep up.

Clutter

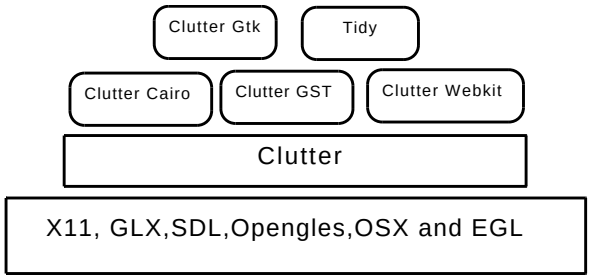

Figure 3: Architecture of Clutter

- **Components**

1. **Clutter**
   Clutter uses OpenGL (and optionally OpenGL ES for use on Mobile and embedded platforms) for rendering, but with an API which hides the underlying GL complexity from the developer.
2. **Clutter-GST**
   Clutter-GStreamer (clutter-gst) is an integration library for using GStreamer with Clutter. GStreamer is a streaming media framework, based on graphs of filters which operate on media data. Applications using this library can do anything from real-time sound processing to playing videos, and just about anything else media-related.
3. **Clutter-Webkit**
   Clutter Webkit is an integration library which allows HTML rendering on GL textures. This could also provide hardware acceleration for video rendering through the browser.

- **Advantages**

1. Clutter has a OpenGLES backend, which makes it suitable for ARM-based devices with a GLES-based GPU.
2. Most hardware provides gstreamer-based libraries for hardware codec support.

- **Disadvantages**

1. Clutter is not production-ready.
2. Tidy, the toolkit built on top of clutter, is still nascent and does not have a comprehensive set of widgets.

4 Benchmarks and suitability

Frame Rate

Frame rate, or frame frequency, is the measurement of the frequency (rate) at which an imaging device produces unique consecutive images called frames. The term applies equally well to computer graphics, video cameras, film cameras, and motion capture systems. Frame rate is most often expressed in frames per second (FPS) and in monitors as Hertz (Hz). To create a fluid interface, the underlying framework should at least output between 25–30 frames per second. This section benchmarks the frame rate across various hardware.

1. Intel Mobile Processor
   The Intel Mobile Processor range consists of high-performance chips which are mainly used in ultra-mobile PCs.
   - **Hardware Specifications**
   - Processor: Intel Mobile 1 Ghz
   - Memory: 1 GB
   - Power Consumption: 14.44 Watts
2. Freescale IMX.31
   The Freescale chip consists of an ARM 1136JF-S core with an onboard GPU which supports OpenGLES.
   - **Hardware Specifications**
   - Processor: 533 MHz
   - Memory: 128 MB
   - Power Consumption: 6.5 Watts

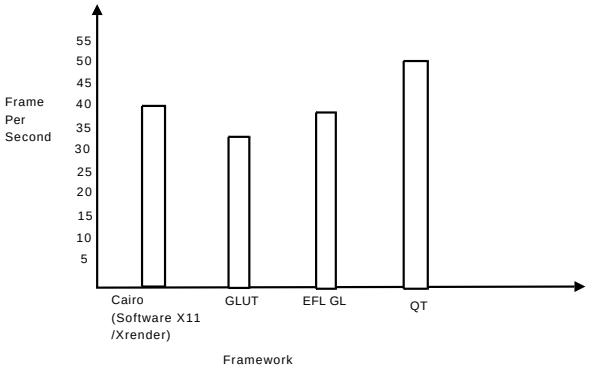

Figure 4: Frame Rate on Intel Mobile Processors

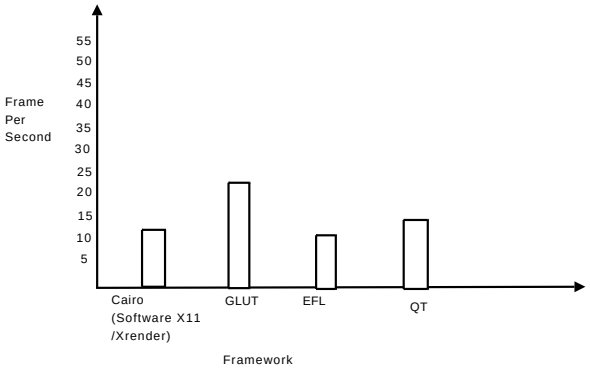

Figure 5: Frame Rate on Freescale I.MX 31

Quality vs. performance

Pixel-perfect drawing is necessary for accurate event processing and coherent visual representation. Current hardware access abstractions like OpenGL/GLES don't provide pixel-exact hardware aliasing. This may result in substantial pain when trying to deal with constant user input and interaction.

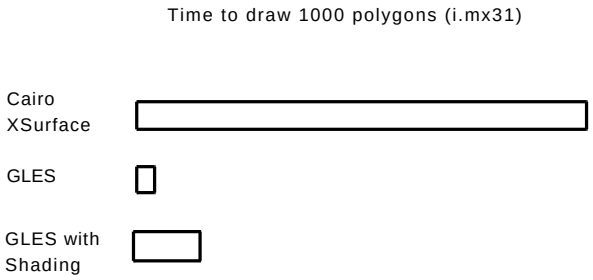

Figure 6: Time taken to draw 1000 polygons

To understand the tradeoffs involved in rendering versus quality, this test draws shaded polygons using OpenGL and compares the time taken to draw 1000 such polygons against a software-only API provided by Cairo. We show the output of both tests to understand quality.

5 Future

As we compete with application frameworks like Cocoa, FOSS application frameworks need to accomplish the following:

1. **Look nicer and feel intuitive**
   Implement aspects of physics and 3D to create a new intuitive paradigm which feels natural and exciting.

2. **Easier to develop, extend, and deploy**
   Learning lessons from web development are extremely necessary, a bridge needs to be built between application developers and graphic artists.

3. **Flexible design to handle multiple interaction modes**
   Increasingly new methods to interact with the computer are gaining popularity. An open design which can easily handle multiple interaction modes makes it easier to build quick support for new input devices.

4. **Established method for hardware acceleration**
   The FOSS application frameworks continue to have a varied approach to hardware acceleration. An accepted approach preferably using X (DRI) would help in making it easy to differentiate one framework from other on a functional basis while also allowing them to work well in unison.

6 Conclusion

This paper analyzes and benchmarks various application development frameworks. There are a lot of application development frameworks which one can use right now, but the ability to build cutting-edge, hardware-accelerated interfaces is still in its infancy. We are still seeing a multitude of approaches, all of which are works in progress. Issues such as hardware acceleration, animation support, and multi-input handling all need more

work to challenge current market leaders. The benchmarks presented cover most of the important usage scenarios and various directions currently being pursued.

Making a decision on choosing an application platform continues to be a subjective decision over a purely objective one. The current bout of approaches need to stabilize for us to make a complete set of benchmark tests which can help you decide one way or the other, for sure!

References

[Keith] Keith Packard, *Getting X Off The Hardware*, http://keithp.com/~keithp/talks/xserver_ols2004/xserver-ols2004-html/

[Zack] Zack Rusin, *Benchmarking tesselation*, http://zrusin.blogspot.com/2006/10/benchmarks.html

[Michael Dominic] Michael Dominic, *OpenGL Shaders and Cairo*, http://www.mdk.org.pl/2007/8/6/vector-drawing-opengl-shaders-and-cairo

[OpenGLES] OpenGLES, *OpenGLES 1.1 and 2.0 specification*, http://www.khronos.org/registry/gles/specs/1.1/es_full_spec.1.1.12.pdf

[Tim] Tim Janik, *OpenGL for Gdk/Gtk+*, http://blogs.gnome.org/timj/2007/07/17/17072007-opengl-for-gdkgtk/

[QT and OpenGL] Qt and OpenGL, *QT and OpenGL*, http://doc.trolltech.com/3.3/opengl.html

[GTK+3.0] GTK+3.0, *Imendio's GTK+ 3.0 vision*, http://developer.imendio.com/sites/developer.imendio.com/files/gtk-hackfest-berlin2008.pdf

[Carl] Carl Worth, *EXA*, http://cworth.org/tag/exa/