WIND RIVER

Graphical User Interface Resources in Linux for Automotive

Table of Contents

Executive Summary

With its desktop heritage and wide deployment in mobile handsets and consumer electronics applications, Linux offers automotive original equipment manufacturers (OEMs) and integrators an impressively wide range of graphical user interface (GUI) technologies and toolkits. The GUI of intelligent in-vehicle systems presents OEMs and integrators with the greatest opportunities for differentiation. Navigation applications, entertainment, wireless telephony, Internet connectivity, remote productivity, and other in-car applications all require GUI capabilities with appropriate and scalable performance, resource utilization, and capability sets.

This paper provides an overview of commercial and free software GUIs available for automotive OEMs and integrators and heuristics for making design and deployment decisions.

Use Cases and Applications

The primary use cases for GUI technology in Linux-based automotive applications include the following:

- Vehicle dashboard and back-of-seat displays with medium to high resolution
- Navigation systems with 2D and 3D map displays, audio output, text-to-speech, and so on
- Entertainment systems for local media playback and streaming from edge, 3G, satellite, and other wireless Internet connections
- Single and multiplayer games
- Desktop-type applications for reading/sending email, opening email attachments, calendaring/scheduling, and being productive while in traffic
- Vehicle maintenance and operational feedback (e.g., vehicle reverse video camera)

Core Automotive UI Requirements

To support this gamut of applications, GUI technology for automotive applications presents a short list of core requirements:

- 2D and (optional) 3D acceleration; OpenGL ES (embedded systems) APIs
- Portable and lightweight application framework
- Ready-to-use UI elements and widgets
- Customizable full-screen application palette with branding/skinning
- Screen resolutions from VGA (video graphics array) (640x480) upward
- Touchscreen support
- Customizable, international input methods
- Speech output[1]
- CPU support for automotive architectures: ARM, Renesas SH, and Intel Architecture
- Driver support for in-car network protocols and peripherals such as CAN, MOST, global positioning systems (GPS), etc.

Unlike the majority of legacy embedded OSes, Linux also supports a range of desktop applications. This broader set of use cases lets Linux offer embedded developers a surfeit of options for user interface design. Extensive embedded deployment, moreover, has helped to optimize and rearchitect UI resources to fit the resources and performance needs of automotive applications.

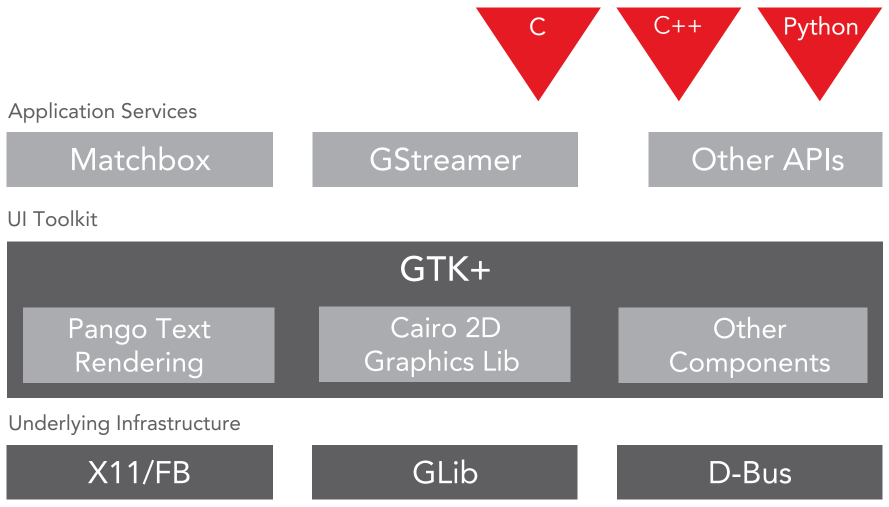

Figure 1: GNOME user interface components

GNOME

GNOME is the most popular of the Linux desktop environments; it is the default desktop for Novell/openSUSE, Red Hat/Fedora, Ubuntu, and other distributions, and also powers many UNIX workstations. Today, GNOME also enjoys the largest and most active developer community for deployment in mobile, automotive, and other embedded use cases. Learn about the main GNOME project at http://www.gnome.org.

GNOME offers automotive OEMs and integrators a rich toolkit with the following component subsystems.

GTK+

GIMP (GNU Image Manipulation Program), or GTK (GIMP Toolkit), is a toolkit and API set for creating applications with graphical user interfaces. GTK is best known and most widely deployed in its role as the foundation for the GNOME desktop on Linux-based workstation computers. GTK is also the underlying framework for a range of Linux-based mobile devices, in particular mobile phones and mobile Internet devices (MIDs) running LiMo middleware and the Moblin platform. Learn more at http://www.gtk.org/.

GTK is written in C with bindings to C++, Python, and C#. GTK is licensed under both LGPLv2.1 and MPL 1.1 and can be freely deployed in commercial applications without reciprocal disclosure requirements. Its broad desktop use has enabled development and customization for a wide range of use cases.

GTK itself has few dependencies beyond the standard GNU libraries. It can render using both X11 and frame buffer interfaces and uses D-Bus for communication among GTK-based application and also to/from other D-Bus publishers/subscribers.

GTK offers developers a rich set of APIs for graphical presentation, with a number of notable subsystems including Cairo and Pango, described in the following sections, and the Accessibility Toolkit (ATK).

Cairo 2D Graphics

Cairo is a 2D graphics library designed to produce consistent results across output media. Currently supported output targets include the X Window System, Quartz, Win32, image buffers, PostScript, PDF, and SVG file output. Back ends currently under development include OpenGL (through glitz) and DirectFB. Learn more at http://www.cairographics.org/.

Cairo APIs provide drawing operators that include stroking and filling cubic Bézier splines,[2] transforming and compositing translucent images, and antialiased[3] text rendering. All drawing operations can be transformed into any affine transformation[4] (scale, rotation, shear, etc.).

With the release of GNOME version 2.1, Cairo APIs were enhanced to include integer-only interfaces to avoid the overhead of earlier desktop-centric floating point architecture.

Pango Text Rendering

Pango is a library for laying out and rendering text, with an emphasis on internationalization. While able to run freestanding, Pango primarily runs in the context of the GTK+ widget toolkit and forms the core of text and font handling for GTK+ 2.x.

Pango is modular and the core Pango engine can be used with different font back ends: client-side fonts using FreeType and fontconfig libraries; native fonts on Microsoft Windows; and native Mac OS X fonts.

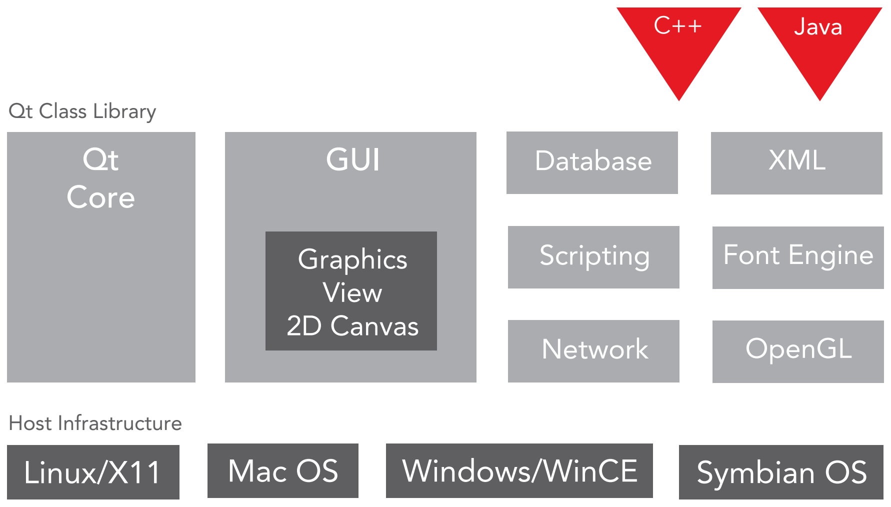

Figure 2: Qt user interface components

Pango supports virtually all major scripts and ships with a wide selection of modules, including Hebrew, Arabic, Hangul, Thai, and a number of Indic scripts. Learn more at http://www.pango.org/.

GNOME Application Services

To support the global ecosystem of cross-platform GNOME applications on Linux, UNIX, Mac OS, and Windows, the GNOME developer community has evolved it to include application-enabling modules and middleware, most notably the following:

- **Matchbox window manager:** Small, resource-efficient windowing library for mobile devices; learn more at http://matchbox-project.org/
- **GStreamer:** GNOME multimedia framework;[5] learn more at http://gstreamer.org/
- **GConf:** GNOME configuration management; learn more at http://www.gnome.org/projects/gconf/
- **GnomeVFS:** Virtual file system for consistent local and remote storage access by GNOME applications; learn more at http://library.gnome.org/devel/gnome-vfs-2.0/

In addition, the GNOME developer community has provided integration with other applications and middleware, such as the following:

- **BlueZ:** Bluetooth stack/tools; learn more at http://www.bluez.org/
- **Evolution Data Server:** Calendar/contact manager; learn more at http://www.go-evolution.org/EDS_Architecture
- **Telepathy:** IM/presence; learn more at http:// telepathy.freedesktop.org/wiki/
- **Avahi:** Service discovery; learn more at http://avahi.org/

Qt Platform for Embedded Linux and Qt Extended

Qt from Qt Software, a division of Nokia, is a C++ GUI toolkit that provides functionality for building graphical user interfaces. Qt supports multiple platforms, spanning desktop systems running Windows, Mac OS, legacy UNIX, and Linux to embedded systems running OSes from Microsoft, Symbian, and, of course, embedded Linux. On the desktop, Qt forms the basis for the popular K Desktop Environment (KDE) project and is a standard installation option on Novell/SUSE, Red Hat/ Fedora, and the main desktop for the Kubuntu distribution.

Qt is an application platform for devices based on embedded Linux, targeting PDAs, mobile phones, web pads, and automotive applications.

Q includes two broad versions:

- **Qt for Embedded Linux:** This enables creation of fixed-function Linux-based devices and is optimized for memory efficiency.
- **Qt Extended:** This is an advanced application platform and user interface for embedded devices such as Linux-based mobile phones. It includes preintegrated applications, allowing OEMs and designers to build feature-rich phones.

Qt for Embedded Linux is offered under both open source (GPL versions 2 and 3) and commercial licenses, and Qt Extended is offered in a commercial license. The commercial branches of the Qt for Embedded Linux/Qt Extended projects include a range of development tools for UI design and layout, internationalization, and cross-platform build.

Enlightenment

Enlightenment is an open source window manager for the X Window System, which can be used alone or in conjunction with a desktop environment such as GNOME or KDE. Enlightenment is also used as a substitute for a full desktop environment.

Enlightenment enjoys deployment in “consumer desktop” Ubuntu-based (Debian) Linux distributions such as gOS and OpenGEU. It also forms one configuration of the OpenMoko mobile phone stack.

The Enlightenment project itself comprises three main components:

- **Enlightenment DR16:** The original window manager project
- **Enlightenment DR17:** A complete rewrite of DR16 intended to function as a complete desktop shell; also known as e17
- **Enlightenment Foundation Libraries:** Graphical software libraries from the Enlightenment window manager project

e17

The e17 environment offers embedded application developers and device OEMs a range of features and capabilities:

- Animated, interactive desktop backgrounds, menu items, iBar items, and desktop widgets
- Window shading, iconification, maximizing, and sticky settings
- Full theming and skinning
- Integrated file manager
- Desktop icon support
- Virtual desktop grid
- Modular design (available modules include pager virtual desktop switching; iBar launcher; iTask application dock; drop shadow for windows; analog clock; battery; CPUFreq; and temperature monitors)
- Support for NetWM, ICCCM, XDG, and other standards
- Internationalization

Enlightenment is licensed under Berkeley Software Distribution (BSD) and so imposes a copyright disclosure requirement on OEMs integrating Enlightenment code into commercial devices.

FST FancyPants

GTK, Qt, and Enlightenment all exist as open source projects with various commercial integrations of each for different markets. There also exist purely proprietary options for building mobile and automotive applications on embedded Linux, such as FancyPants.

FancyPants is a graphics and multimedia platform for developing and deploying embedded Linux applications in consumer, commercial, and industrial devices, including automotive applications. Theming and scripting (with XML and Lua) lets designers create dynamic layouts while an optimized run-time delivers performance with or without hardware acceleration. Through efficiency and minimal resource requirements, FancyPants brings multimedia graphics capabilities to embedded devices, without inducing new hardware requirements. Learn more at http://fluffyspider.com/.

Java

Java made early strides in automotive applications, especially through the efforts of IBM Global Services. As with Java in other mobile devices, automotive OEMs and integrators have struggled with both performance and interoperability issues in building and deploying Java-based automotive systems over the long term. In particular, OEMs and independent software vendors (ISVs) have faced fragmentation in Java profiles/JREs (CDC, J2ME, J2SE, etc.) and APIs as well as “too many choices” for Java UI frameworks.

Today there exists a range of increasingly standardized UI options for Java in embedded systems, with Sun and others making introductions in the past year. The major paths include the following:

Abstract Window Toolkit

The Abstract Window Toolkit (AWT) is the original Java platform-independent windowing, graphics, and UI widget toolkit, dating back to JDK 1.0/1.1. AWT is now part of Java Foundation Classes (JFC), the standard API for providing a GUI for a Java program. It is widely used and deployed in embedded applications and represents the “least common denominator” for Java graphics and UI resources. AWT is also the GUI toolkit for several mobile Java profiles. For example, Connected Device Configuration (CDC) profiles require Java run-times on mobile telephones to support AWT.

AWT widgets provide a thin abstraction over an underlying native UI. Creating an AWT object (e.g., a check box) causes AWT to call the underlying native routine to create that object. Many application developers prefer this model because it provides fidelity to the underlying native windowing system and better integration with native applications. Learn more at http://java.sun.com/products/jdk/awt/.

Swing

Swing is a widget toolkit for Java and is part of the JFC. Swing provides a more sophisticated set of GUI components than AWT. In particular, Swing provides a native look and feel that emulates several platforms and also supports a pluggable look and feel that allows applications to have a unique look and feel independent of the host platform.

Swing has been included as part of the Java Standard Edition since version 1.2, but notably not with J2ME. For this reason, and because of additional compute and resource requirements, Swing has not taken hold in mobile and embedded applications. With the advent of more powerful, multicore embedded CPUs and enhanced desktop interoperability (as with Intel Atom), Swing can be a more attractive UI toolkit for automotive. Learn more at http://java.sun.com/j2se/1.5.0/docs/guide/swing/.

SVG Tiny and JSR 226

Scalable Vector Graphics (SVG) is a W3C specification with an XML-based grammar that defines instructions for rendering rich, interactive graphics and multimedia applications and content. In response to input from the mobile ecosystem, the W3C SVG working group defined two profiles to target resource-constrained devices: SVG Basic and SVG Tiny.

JSR 226 can be thought of as Java wrappers/APIs for SVG Tiny and has become the standard for interactive/animated 2D graphics on the J2ME platform. Developed within the Java Community Process (JCP), JSR 226 acknowledged the mobile industry standardization of SVG Tiny and built its APIs on top of them. JSR 226 offers simple methods to load and display SVG Tiny files on the go, as well as manipulate SVG Tiny content or create it from scratch on a mobile device.

JSR 226 offers developers the option of prototyping with full SVG using Swing on the desktop and subsequent refactoring onto SVG Tiny for resource-constrained mobile/automotive devices.

LCDUI

LCDUI (Liquid Crystal Display UI) is the Java ME (MIDP) UI class library. LCDUI API provides a small set of display primitives common to mobile device user interfaces: List, Alert, TextBox, Form, and Canvas. Learn more at http://java.sun.com/javame/reference/apis/jsr037/.

LWUIT

LWUIT (Lightweight User Interface Toolkit) is a UI library designed for deployment together with applications. LWUIT helps content developers to create compelling and consistent Java ME applications. LWUIT supports visual components and other UI elements including theming, transitions, and animation. Learn more at https://lwuit.dev.java.net/.

Browser-Based UI

The previous sections on native and Java UI schemes focus primarily on the display of application output and content local to a device. An alternative paradigm is to abstract the difference between local applications and content and remote equivalents by using a web browser as the primary UI display mechanism. Browser-based UI design builds on HTML/XML documents and programming in JavaScript, PHP, and CGI (server-side Common Gateway Interface) scripting instead of relying on toolkit APIs for rendering and display. Documents and content that constitute the device UI are then served by a local embedded web server (e.g., thttpd, an open source software web server by ACME Laboratories; Boa; or even Apache) or by remote servers over WAN and other connective paths. A further advantage of a browser-based UI paradigm is the ability to leverage AJAX code and tools for embedded UI applications.

A browser-based UI typically eschews the "window frame" around the browser. Many browsers encountered on the desktop have embedded versions that offer full-screen functionality for UI applications. Some rely upon host UI toolkits (e.g., GTK+) while others (optionally) include their own graphics primitives, calling X or frame buffer routines directly.

Linux hosts perhaps the broadest array of browsers of any OS in embedded or enterprise IT. Available browsers include Mozilla/Firefox (and the Gecko rendering engine), Konqueror, Opera, and multiple WebKit derivatives.

Supporting Legacy In-Car Applications

The automotive marketplace features long-lived hardware and software platforms. OEMs and integrators demand that legacy applications be able to migrate from prior-generation implementations to new Linux-based automotive systems (and even beyond). These legacy systems were built on a range of OSes, middleware, and languages; fortunately, Linux and other free and open source software (FOSS) projects provide multiple rehosting paths for such legacy code. Examples include the following:

- **"Classic RTOS" (Wind River's VxWorks, pSOS, etc.):** Multiple projects and toolkits offer options for rehosting on Linux, using RTOS emulation and/or embedded virtualization. Information on the topic is available from Wind River.[6]
- **Embedded UNIX (QNX, LynxOS, etc.):** These legacy OSes share more than 90% of their APIs and architectures with Linux, easing porting and reuse of legacy automotive code.
- **Java:** All profiles of Java host and run on embedded Linux, making the transition nearly transparent.
- **Legacy GUI:** In-house and commercial off-the-shelf (COTS) proprietary UI stacks can usually be rehosted in Linux-based virtual machines running over available embedded hypervisors. The only onerous requirement is to encapsulate ad hoc use of frame buffers and video memory into a driver abstraction for safe virtual machine integration and/or driver paravirtualization.

Conclusion

The goal of this paper is to elucidate the wealth of options for building and rehosting new and legacy user interface code and content for automotive application on embedded Linux, helping to design and speed automotive applications to market.

Linux-based development for automotive applications enjoys an extremely rich toolbox of mature solutions, platforms, and point technologies, thanks to the following:

- Five-plus years of burgeoning deployment of Linux in mobile and GUI-intensive consumer electronics applications
- More than 15 years of Linux desktop development and deployment in enterprise, technical workstation, and consumer desktop systems
- Almost three decades of graphical workstation investment in and around X11 (X Window System)

Notes

1. *Multimedia Resources in Linux for Automotive*, Wind River, http://windriver.com/whitepapers/.
2. Curves interpolate between two endpoints, with additional parameters governing the shape determined by two ``control points.''
3. The technique of minimizing the distortion artifacts known as *aliasing* when representing a high-resolution signal at a lower resolution in digital signal processing.
4. Any set of translation, rotation, and scaling operations in the two spatial directions of the plane.
5. See note 1 above.
6. Dan Noal and Tim Fahey, *Effective Linux Migration Processes*, Wind River, http://windriver.com/whitepapers/.