RTC

THE MAGAZINE OF RECORD FOR THE EMBEDDED COMPUTING INDUSTRY

Android for Embedded: The Good, the Bad and the [insert adjective here]S

As with embedded Linux before it, the complexity inherent in building and shipping and monetizing Android-based designs is not stopping or even slowing adoption.

BY BILL WEINBERG, BLACK DUCK SOFTWARE – FEBRUARY 2014

Android, Google's mobile phone platform, has now climbed to the top of the embedded OS heap, where it reigns together with other versions of Linux and FreeRTOS. The ascendency of this mobile operating system may surprise embedded industry veterans—Google's mobile OS boasts a large resource footprint, it's not particularly well suited to serve real-time response requirements, and its functions are very display-centric.

Objections notwithstanding, Android's popularity is large and still growing for applications as diverse as automotive (OEM and aftermarket IVI), wearable computing (wristwatches, head-up helmet displays and smart glasses), robotics (for domestic and telepresence applications), multimedia (TVs and media players), special-purpose tablets (most notably, the Amazon Kindle) and even near-earth satellites.

Device manufacturers are designing in Android not because doing so is (merely) stylish, but because the OS reflects the confluence of technology and market trends building over the last five years:

- Increasing deployment of free and open source software (FOSS) in intelligent devices
- Ubiquitous device connectivity, especially over TCP/IP networks
- Cost-effective availability of multicore silicon for most or all types of intelligent devices
- Ever-declining costs from DRAM and flash memory, with higher integrations of both in existing form factors
- Similar (if less precipitous) downward trends in display and touchscreen prices
- Differentiation through total user experience (UX) vs. core device functionality (even previously headless devices now often sport attractive UIs)
- A burgeoning Android applications marketplace that drives developer fluency in Java and the Android programming framework
- OEMs' desire to leverage Google Play, both for the million+ apps offered there and as a distribution mechanism for OEMs' own device-specific applications
- "Business-friendly" Apache 2.0 licensing of the Android software stack

To evaluate it as an embedded operating system is to examine if and how Android really supports these trends, and to call out a few other areas where the OS may or may not fit the embedded applications bill including cost and licensing, differentiation, the applications marketplace and security.

Cost and Licensing

Android, like other FOSS, is free in the sense of free speech—the vast majority of the Android platform is released under FOSS licenses that promote and preserve the free circulation of underlying source code (see below). Acquiring that code is also free as in free beer—Google provides versions of the majority of the platform in repositories.

Costs for Android come in the form of multicore CPU, GPU, DRAM, Flash and other resources required for an acceptable user experience (UX). And, while costs for these and other components continue to fall, it's still very possible to underspec devices. Insufficient hardware provisioning and accompanying UX degradation is a key cause of staggeringly high return rates for Android handsets, watches and other gadgets—reportedly 30% or more for some devices. The moral is that it's worth not skimping on a solid BoM for your Android-based design. Other costs are not a mystery to OEMs—integration, customization, QA, etc., which are no different with Android than with Linux or other embedded platforms.

The Android project presents developers and OEMs with a tempting and liberal licensing regime. The project licensing states,

The preferred license for the Android Open Source Project is the Apache Software License: Version 2.0 ("Apache 2.0"), and the majority of the Android software is licensed with Apache 2.0.

The Apache 2.0 license is indeed OEM-friendly, in that many or most device manufacturers still consider low-level hardware interfaces to be proprietary. Indeed, the Apache 2.0 license section "4. Redistribution" states,

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium: with or without modifications, and in source or object form, provided that you meet the following conditions, which include redistribution of the license itself; notification of modification in source files if/when redistributed; retention of copyright, patent, trademark and attribution notices in source files and a human-readable NOTICE of the above. Yes, redistribution of (modified) source code is completely optional.

However, Apache 2.0 is not the whole licensing story. The Android Licensing page goes on to say:

> *While the project will strive to adhere to the preferred license, there may be exceptions that will be handled on a case-by-case basis. For example, the Linux kernel patches are under the GPLv2 license with system exceptions.*

In fact, the Android sources include code distributed under at least 19 different open source licenses, including those with reciprocal and liberal requirements—from GPL to LGPL to BSD to Public Domain, and points between.

This diversity is not a bad thing per se—it reflects the multiplicity of projects upon which Android is built, including the Linux kernel (GPLv2), Webkit (BSD/LGPL), SQLite (Public Domain with other licenses for tools/scripts) and others. However, the panoply of licenses does defy the perception that Android platform is “all Apache” with minimal compliance requirements.

Given that OEMs frequently customize Android “from the bottom up,” adding modifications to the underlying Linux kernel—starting with device drivers—Android is no more (or less) business friendly than any other Linux-based platform with comparable compliance requirements.

Differentiation

Adding an attractive UI to a hitherto headless device or one with a ho-hum interface provides an alluring upgrade and an opportunity to differentiate a product in crowded markets. Android can certainly provide a path to differentiation, not only with splashier visuals but by turning a mono-function device into an “applications platform.” This re-invention is evident in devices that already boast attractive UIs—DTV and IVI in particular.

But Android is not the only “path eye-candy,” and it’s certainly not the least costly to implement—numerous proprietary and open source UI graphical and UI frameworks (Qt, e17, PEG, FancyPants, etc.) can deliver the same (or better) visual oomph with lower-end graphics and application processors, usually without GPU.

The application platform aspect of Android—the ability to run apps from OEMs, channel partners and diverse third parties—is quite attractive, but only if those third parties are actually likely to offer apps for your device or device-type. Moreover, it’s important to ask if a device use case truly accommodates third-party apps. For example, would you want to play games or risk malware on Android-based medical or industrial devices?

Finally, while it is an advantage to be the first OEM “on the block” to offer an Android-based device in a given vertical market, competitors and copycats will soon follow, at which point your device will be “just another Android.” Unless, of course, you can further customize and brand the Android look-and-feel. Fortunately, Android is quite amenable to reskinning (much more than Microsoft Windows Embedded, Windows Phone and family). Unfortunately, many OEMs and app developers strive to customize Android in ways that require use of Android native APIs (via Android NDK), forking the platform and/or creating apps that only run on select devices, thus limiting interoperability.

Application Marketplaces

Given that Android-based devices have the potential to run third-party apps, it behooves OEMs and developers to consider exactly how they will leverage Google Play and other Android apps markets, and/or will instance their own device or brand-specific app stores as Amazon has done. Some device types will benefit greatly from a broad selection of existing third-party apps—especially tablets and DTVs, and of course mobile phones. Other types will only benefit from app stores as streamlined distribution channels, with limited need or capability to run most of the million+ apps on Google Play.

The real benefit of building your next device with Android comes not from existing apps but instead from the global community of Android apps and platform developers. The lure of a hit app has created a worldwide gold rush, with hundreds of thousands of developers being trained to code for the platform, and ISVs and development houses building practices to deliver ready-to-deploy and custom apps to OEMs and enterprise alike.

But two bits of caution should temper OEM enthusiasm over leveraging what is truly a vibrant market. The first is that apps developers are usually not platform developers—they can create software for your device only once it is stable and market-ready. Embedded systems programmers, for Android as for Linux as for RTOSs, are still in high demand and low supply. The second is that just because you release an Android-based device, developers will not spontaneously craft apps tailored for it. You still have to create and nurture a developer community that reflects the particulars of your device and vertical marketplace.

Security

Android suffers from a less-than-stellar reputation when it comes to security. In particular, the bazaar-like nature of Google Play and other app stores has resulted in wholesale distribution of malware itself and new vectors for it. Some estimates for infected or spoofed Android applications run as high as 90+%, with the U.S. Department of Homeland Security attributing 79% of all mobile malware to the Android platform.

Putting aside applications for the moment, Android itself enjoys a fairly robust security architecture. Android networking, network security, file systems and user security model are shared with Linux, whose architecture and development community boast highly effective technical and procedural mechanisms to prevent and respond to a range of exploits. Moreover, Dalvik (Google's cleanroom Java VM for Android) has not suffered from the same security woes as Oracle's original implementation.

For both the underlying Linux kernel and for Dalvik, it is key to enable timely updates to system software for Internet-facing Android-based systems, which is to say, practically all of them.

The application's front requires a rather more draconian approach to secure the platform. While you may have chosen Android for its openness, you are best served security-wise by locking down applications on your device, sourcing them through known secure channels, and/or using secondary solutions, third-party containers and other isolation mechanisms to wall off apps of uncertain provenance.

In weighing the aspects of Android's suitability for today's embedded designs, there is clearly the good—cost, mostly liberal licensing, high functionality, ample applications and multiple apps channels, and a global and energetic developer community. But there's also the bad—the need for heftier BoMs, Android fragmentation and the malware morass. But more than good or bad, designing and deploying with Android, like most things in life, is complicated—the double-edged nature of differentiation with the platform, finding the right strategy for leveraging app stores and developer communities, and the need for comprehensive license compliance in spite of the "Apache Inside" label on the cute green box.