Linux Pundit
Open Source Opinion & Analysis
Business & Technical Consulting

WHITE PAPER

Uniting Mobile Linux Application Platforms

July 31, 2008

Prepared by

Bill Weinberg, LinuxPundit.com

TABLE OF CONTENTS

Abstract

This white paper outlines the commercial and technical landscape around deployment of Linux-based mobile applications platforms. It examines ecosystem forces at play in (re)defining “mobile Linux” and the diversity of paths to deploying open source in mobile devices. It documents the roots causes for and locus of fragmentation among Linux-based mobile software and posits a forward-looking approach for crossing this rocky landscape and reaching a unified platform on the “other side”.

In particular, this white paper explores how ISVs, OEMs and operators, rather than be deterred by a diversity of approaches and emerging standards, are shipping Linux-based mobile devices in ever-greater volumes.

Uniting Mobile Linux Application Platforms

Introduction

The mobile telephony market presents a tantalizing opportunity to software platform suppliers, mobile operators, OEMs and increasingly, to application developers. The rapid evolution of mobile device capabilities and services offerings, combined with a global market of over one billion units annually[1], constitutes an irresistible siren song to new players and new technologies. For their part, handset OEMs (Original Equipment Manufacturers) and mobile operators themselves constantly search for shorter and less expensive paths to market and to volume, and for differentiation in a crowded and often commoditized environment.

Starting in 2002, Linux entered this dynamic marketplace as a software platform for enabling well-provisioned smart-phones and select feature phones. It offered OEMs and their operator customers a range of benefits, including

- Lower software BoM (Bill of Material) impact than proprietary solutions
- Options for UI customization, extension and differentiation
- Native support for IP networking and desktop peripheral interfaces
- A locus for innovation from OEM, operator and community efforts
- Higher reliability over legacy RTOS-based platforms
- A bridge to reusing desktop/enterprise software and to end-to-end application development

In the last five years, Linux use in mobile has matured and expanded into the mainstream. Today. Linux is being deployed in 50-60M[2] units from handset vendors that include Motorola, NEC, Panasonic, Samsung and a range of regional players like China's Datang, Haier, Huawei and ZTE. Moreover, Linux-based mobile deployment has been growing at over 400% CAGR and is expected to garner at least 25% of the smartphone segment and a significant piece of the middle tier by 2010[3].

For all the advances made in targeting and refining Linux for mobile, and despite the strong and growing deployment trend for the open source OS, those fielded phones have little in common except the ARM architecture Linux kernel and core run-time libraries. First generation Linux-based handsets variously deployed both commercial and RYO (roll-your-own) kernels from the 2.4 Linux source tree, with accompanying versions of **glibc** and other base enabling libraries and tools. A number of devices/models shipped with Qt[4] as the application framework, but an equal number integrated and deployed UI (user interface) frameworks specific to OEMs and ISVs (operating system vendors, e.g., China MobileSoft, Mizi Research, and indeed Purple Labs).

With the appearance of the 2.6 Linux kernel in 2004, both ISVs and OEMs began integrating that newer base OS and embedded platforms based upon it. The biggest change post-2.6 was the advent of a number of frameworks: GTK+[5] as a graphical UI framework, gstreamer as multimedia framework and WebKit as browsing framework.

---

[1] Source: Gartner 2007.

[2] Source: Informa 2007/2008.

[3] Source: Diffusion 2007. In China, Linux already powers at least a third of all shipping smartphones.

[4] Qt - the graphical UI framework that forms the basis for KDE, the K-Desktop Environment. Qt is dual-licensed for commercial use by Trolltech, now a division of Nokia.

[5] GTK – GIMP Tool Kit (GIMP is the GNOME Image Manipulation Program). GTK forms the basis of the GNOME desktop and is freely licensed (under LGPL) by the GNOME community for desktop and embedded applications.

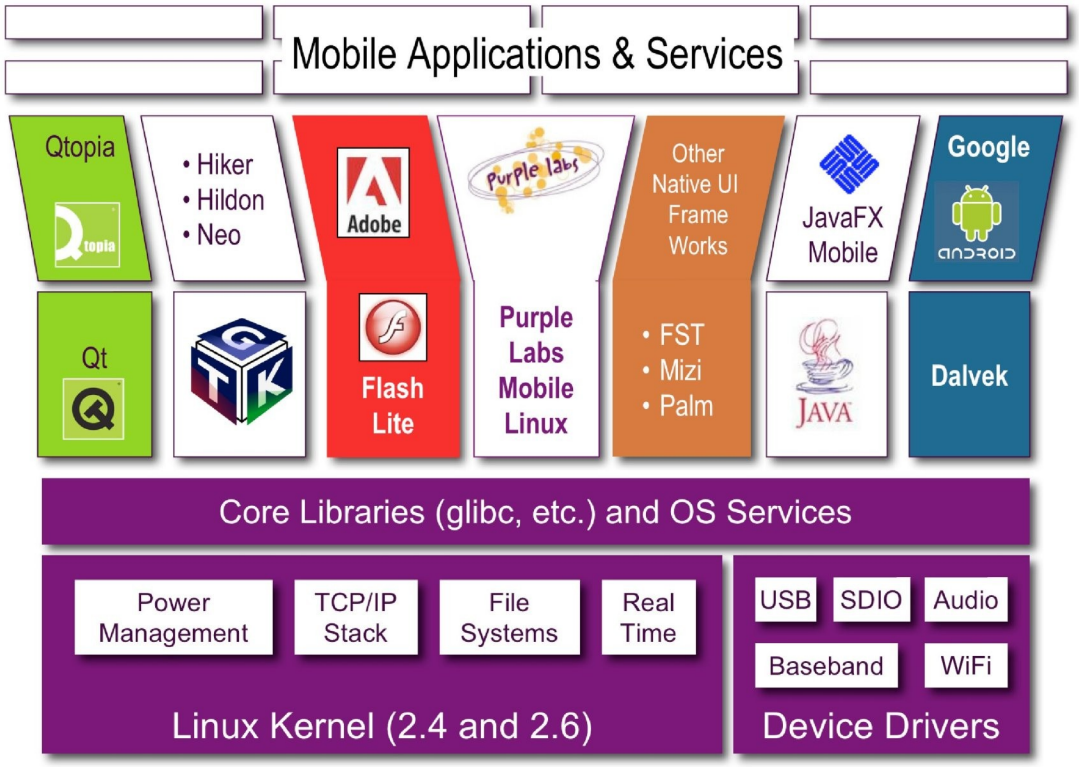

Figure 1. – Competing / Complementary Mobile UI / Application Frameworks over Linux

By the end of 2007, OEMs had shipped three to four-dozen different Linux-based handsets and announced more to come. For all the investment in application framework enablers by OEMs, ISVs and also by standards bodies and consortia, the locus of interoperability for these devices remains firmly rooted in Java (as it is for non-Linux phones), and not in emerging Linux native APIs and middleware.

Organic Defragmentation

In discussions of open source software business models[6] (and in other businesses as well) there exists a balance between innovation and shared investment in infrastructure. In more conventional terms, this balance hangs between value-added IP plus value-enhancing activities vs. investments in (shared) commodity areas.

In the last decade, Linux and other open source software have excelled at commoditizing base platform software (OS, libraries, etc.). Community-based FOSS (Free and Open Source software) effectively "breaks down the doors" at suppliers of these infrastructure elements, internally and externally. Rethinking of how platforms are built, deployed and maintained has radically changed the OS landscape, by deconstructing traditional vertically-integrated views of software value-added (cp. Microsoft, Apple and Sun). This deconstruction supports the current strong position of Linux and other FOSS in server, network infrastructure, mobile and other embedded applications.

The term applied to this division between value-added and commodity is the "Value Line". Broadly speaking, technology below the *Value Line* is best delivered through shared and prefer-

[6] E.g., GOLDEN and WEINBERG [2006-2008].

ably open source implementation (in particular, the LAMP stack[7]), while areas above the line present opportunities for adding value, independent of implementation (e.g., Oracle, Adobe, and commercial mobile application stacks).

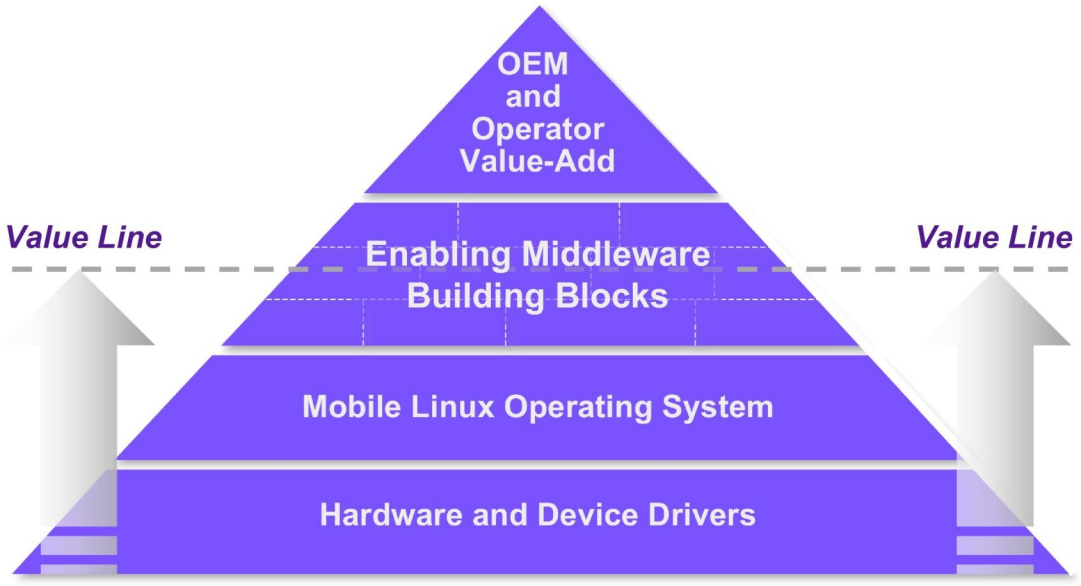

Figure 2. – The Value Line separating commodity software from differentiating IP

The Quest for Interoperability

At the time of writing this white paper, the status quo for Linux-based mobile phones could be summed up by two trends

- Rapidly growing, adoption, deployments and opportunities
- OEMs, ISVs and consortia trying to fight fragmentation among phone platforms by proposing and promoting their own Linux-based platforms

These two movements, rather than heading towards unification from ubiquity of mobile Linux, have given the *appearance* of growing fragmentation derived from their diverse architectural approaches (as in Figure 1.). A key message of this White Paper is that these approaches differ greatly when viewed from the *top-down*, but actually exhibit growing unity and increasing interoperability from the *bottom-up*.

Interoperability and Market Opportunity

The Windows-based PC market has thrived and grown not merely from the *push* provided by Microsoft, but from the ecosystem and end-user *pull*. This structural demand is engendered and amplified by the tens of thousands of available applications and device drivers available for Microsoft Windows platform. In particular, Microsoft partners branding[8] gives consumers and enterprise users confidence that off-the-shelf applications, hardware and services will function "out of the box" on Windows-based workstations and severs they provision.

Mobile/embedded computing has enjoyed analogous interoperability programs with nearly comparable impact – PalmOS/Garnet compatibility programs for Palm Treo and legacy hardware as-

[7] Linux OS, Apache web server, MySQL database and Perl/PHP/Python scripting

[8] "Designed for Windows" (software) and "Certified for Windows" (hardware)

sure availability and functionality of some 30,000 applications, with similar claims made today for Windows Mobile, SymbianOS and other software platforms.

Interoperability is a laudable goal. Ideally, interoperability gives operators and consumers greater choice of software and services they can deploy and use. A common software platform would simplify roll-out of operator programs and services on heterogeneous fleets of handsets. And unification of mobile phone OSes and APIs could hold down costs for ISVs (Independent Software Vendors) targeting the phone market.

Lacking a shared platform today, a forward-looking quest for interoperability poses challenges to the mobile phone industry:

- Waiting for a viable common platform slows adoption of mobile Linux (perhaps even pausing actual deployment of current Linux-based phones)
- Choosing an existing platform (e.g., Windows Mobile) force-fits interoperability and severely limits the ability of OEMs and operators to differentiate their offerings
- A rush to a least-common denominator platform stifles innovation and occurs at such a low level that it does not actually foster interoperability, only the appearance of it

The Linux Interoperability Challenge

Circa 2002-2003, the mobile industry greeted the arrival of Linux with great fanfare. Linux-based mobile was touted both as a native platform for higher-performance applications and as being able to leverage the wealth of desktop of server applications, utilities, and tools. However, Linux and FOSS entered a marketplace with little traditional support for interoperability and no viable technical means to achieve it

This basic set of historical facts should temper concerns around mobile Linux fragmentation today. However, pundits worry that Linux could become irrelevant waiting for a unified stack with real-world interoperability to appear and to mature. This worry is compounded by the perception that mobile Linux ecosystem players are not actually working *towards* interoperability, at least not yet. Why? Because operator requirements for applications and APIs are being met with preloaded code; operator programs are still rolling out on Java, not native Linux (nor any other majority platform, yet).

Interoperability becomes imperative only when the ecosystem both defines what interoperability really means (Java or native or another paradigm) and when operator programs and end-user buying habits create pull and demand for it. Left to their own devices (mobile or otherwise), OEMs and ISVs implement to serve their own business goals and technical preferences, not for the abstract goal of platform unification.

Native Linux Platforms

Native mobile programming has by far the richest potential to deliver high performance and rich end-user experiences. To date its practitioners have mostly been developers on Microsoft Windows Mobile and Symbian Series 60 devices. On Linux, native programming is the most hyped, and the most complex discussion.

Platforms and Stacks

The native mobile Linux programming marketplace teems with self-styled solutions, advanced platforms and applications stacks. Examination sorts these offerings into three basic categories:

LSP – Linux Support Packages[9]

Horizontal embedded Linux suppliers[10] have optimized versions of their embedded development kits to target mobile designs. This optimization focuses on "abstract" mobile capabilities like kernel footprint, kernel boot time (to **init**), chipset enablement (e.g., drivers for peripheral sets on TI OMAP and FreeScale MX), low-level device power management, and support for flash and RAM file systems. While essential and ubiquitous in shipping mobile devices, this type of platform is not "application enabling"

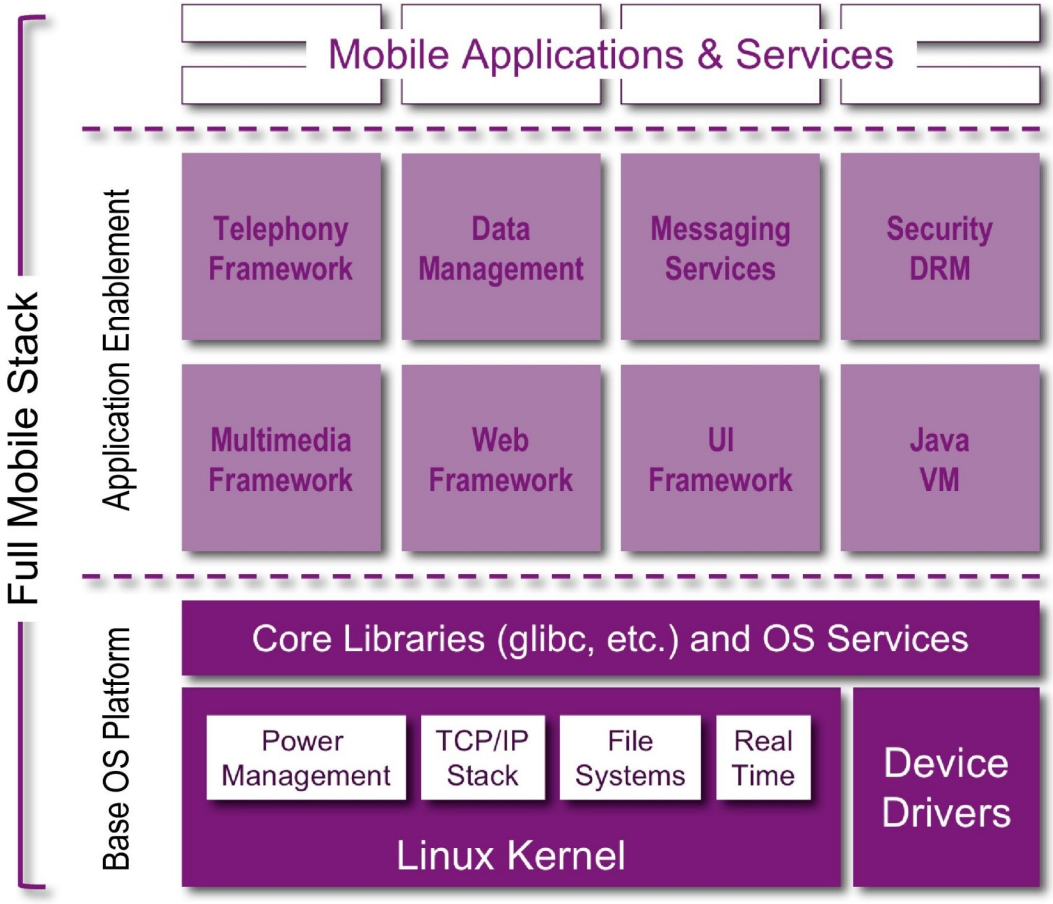

Figure 3. – Mobile Platforms and Stacks Hierarchy

Mobile Applications Platforms

A mobile application platform ideally presents APIs and services (i.e.. *middleware*) designed to enable development and deployment of actual applications visible to the end-user. Such platforms comprise multiple frameworks for key functions like telephony, data management, messaging, security/DRM, multimedia, web, user interface, and also Java.

Consortia like the Linux Phone Standards Forum (LiPS) and the LiMO Foundation are working to standardize this layer and these frameworks, but few commercial entities[11] limit their offerings

[9] LSP arises from the desire to differentiate Linux board support from low-level RTOS board support packages, or BSPs

[10] MontaVista, Wind River and a few others

[11] The ecosystem does include very specialized suppliers of telephony functions like Bluefin, of multimedia applica-

to enablement only.

Applications Stacks

Commercial phone ISVs and also handset OEMs themselves by necessity deliver both applications platforms and a set of applications that run on them. At a minimum these are the "hygiene" or "preload" applications that constitute the core expected end-user applications: dialer, phone book, messaging client, email, browser, media player, etc.

This area is today the most crowded and includes ISVs like Purple Labs, ACCESS, A La Mobile, Azingo, Mizi Research, OpenMoko and Trolltech, and handset manufacturers Motorola, NEC, Palm and Panasonic. Beyond Linux are, of course, Microsoft Windows Mobile and SymbianOS.

UI FrameWorks

Underlying mobile applications stacks and the enablement platforms beneath them are native user interface frameworks. UI frameworks are the most visible locus of mobile Linux[12] fragmentation.

Qt and Qtopia

The majority of Linux-based phone shipped to date employ the commercial version of Trolltech's Qt graphical user interface. Five years ago when OEMs began experimenting with, developing and deploying Linux-based mobile devices, Qt was state-of-the-art. Qt offered an easy to prototype tool kit, and fast time-to-market with off-the-shelf support for PDA "hygiene" apps in Qtopia. Qt also featured bridging to the K-Desktop application ecosystem built on both commercial and free versions of the same Qt libraries.

Today, Qt appears to be in decline as a mobile development framework. OEMs with tight bills of material seek a royalty-free solution; developers complain of complexity in the Qt C++ programming model, weak global support, inappropriateness of the Qtopia application set and limited opportunities for differentiation on Qt/Qtopia-based devices. Also, the acquisition of Trolltech by Nokia in Q12008 has further distanced the framework from Linux-centric development.

Finally, in response to input from open source community members, OEM pricing concerns and desires for greater interoperability, consortia like LiMo[13] and even the Linux Foundation have specified GTK over Qt in their public specifications.

GTK, GNOME and Mobile FrameWorks

Community-based GTK and Gnome components like **gstreamer** and Cairo window manager have enjoyed substantial investment by both commercial and community forces. Most ISVs today support GTK and **gstreamer** as part of their offerings (e.g., ACCESS, Purple Labs and Azingo); consortia have endorsed GTK (LiPS, LiMo, et al.) and OEMs have invested in GTK or plan to migrate to it shortly.

When OEMs began going to market with Qt back in 2002, using GTK in mobile was still considered *experimental*. In particular, the Cairo window manager still needed optimization for mobile deployment and no single viable widget set ran on top of it. As such, commercial entities and community entities felt free to experiment with it, resulting in several divergent use models and widget sets; the two most prevalent are Hildon (employed by Nokia maemo.org for web pads) and Hiker (developed by ACCESS for ALP, the ACCESS Linux Platform).

---

tions enablers like Sky Mobile, of man-machine interfaces (MMI) like Digital Airways, and of course of mobile Java Virtual Machines like Esmertec.

[12] And also desktop Linux fragmentation

[13] And LiPS before it

Other Native UI Frameworks

Beyond the two-way Qt/GTK split are several other frameworks: Enlightenment/e17, Mizi Research PRIZM, Palm's Linux port of the legacy Garnet UI, and Fluffy Spider FancyPants. In China in particular there are other legacy frameworks from smaller ISVs and OEMs ad hoc integration and regional implementations.

This range of native UI frameworks reflects the particulars of resource-constrained embedded development. Historically, handset OEMs and ISVs eschewed off-the-shelf desktop-derived graphics middleware, preferring customized and highly-optimized graphical frameworks to conserve limited memory and CPU power. With today's better-provisioned mobile devices, there exists new temptation to look to desktop display middleware (Java, Flash, etc.) but OEMs consistently return to native UI implementation. The difference today is that standards-based native graphics can deliver needed performance in a mobile-ready footprint.

Other Mobile Applications Paradigms on Linux

How fragmented is the mobile Linux landscape, anyway? Do fissures really split the platform in two, into a dozen, or as some claim, almost two dozen jagged pieces? Let's take a moment to examine the reality of fragmentation fears, starting with the divides documented in Figure 1.

Java on Linux

Most shipping Linux phone designs deploy a Java Virtual Machine as a means to accommodate both pre-load (OEM) and post-load (operator and after-market) applications. In general, because Linux-based phones are fairly well provisioned and have somewhat more generous BoM budgets, OEMs choose to deploy J2ME as their Sun Java profile of choice and can run both MIDlets through full Java applications. Most major OEMs license Java run-times directly from Sun, although a few also treat with Sun "coffee cup" licensees. As with Java on non-Linux phones, OEMs still need to customize the Java run-time to suit the particulars of actual phone hardware, placing Java-on-Linux phones squarely in the fragmented mainstream.

MOTOMAGX and JavaFX

Two branded Java platforms have emerged in this space. The first, MOTOMAGX (originally L+J) is actually a native platform (based on Qt). The supplier, handset manufacturer Motorola Mobile, has only published Java APIs while it slowly transitions the platform toward GTK+.

The second, Sun JavaFX Phone Edition, sets out to reinvent mobile applications development on Linux as a pure Java experience. With the exception of the Linux kernel itself, Sun's JavaFX delivers a completely Java-based mobile stack, including base OS libraries (i.e., no **glibc**), enabling middleware and pre-load applications. To date, Sun has not announced any shipping devices or design wins based on JavaFX and is currently emphasizing JavaFX applications and services rather than the platform itself.

Android / Dalvik – Java on Linux Redux

In Q42007, Google announced with great fanfare that it was entering the fray, with the Linux-based Android platform. Structurally and thematically, Android should be grouped with Java on Linux platforms, except that the underlying *Dalvik* virtual machine seeks to re-invent the Java run-time model (and conveniently side steps Sun licensing requirements and community process). As such, it is mostly comparable to JavaFX Mobile, except that

- It is or will be available as open source
- The base platform is (or will be) royalty-free

- Instead of a community process to control divergence, OHA (Open Handset Alliance) invokes a hopeful anti-fragmentation pledge
- Dalvik is lacking a native security model and provision for application signing

Most importantly, unlike Sun with JavaFX Mobile, Google is not seeking revenue from deployment of the stack itself, but presumably has plans to realize advertising and services revenues from future deployed Android phones and the networks on which they will operate.

Flash and Other Scripting Paradigms

Adobe and its ecosystem partners have for several years attempted to (re)position Flash and related technologies as not merely a multimedia deployment media for desktop computing but as a new paradigm for all user interfaces. With the release of FlashLite, and with open source initiatives at Adobe (e.g., the opening of the Flash SDK[14]). Adobe has sought to position its technology as a mobile applications platform in general and an ideal one for Linux in particular.

Adobe is not the only purveyor of a scripted multimedia approach. Most notably Microsoft has introduced and positioned Silverlight as a ubiquitous multimedia platform that crosses mobile/desktop, OS and browser boundaries (http://www.microsoft.com/silverlight/). There also exist GNASH, a free software Flash work-alike (see http://www.gnu.org/software/gnash/) and proprietary scripted platforms like FST FancyPants (see http://www.fluffyspider.com), with underlying programming occurring in scripted Lua.

Web Applications

The shortest path to platform interoperability may actually involve redefining or limiting what you mean by *platform*. The best example of this semantic and practical shortcut lies in web applications. Very much like Java, web browsers supply

- ubiquitous mark-up and programming languages (HTML/CSS, Javascript and php)
- a well-understood execution model (HTTP, Javascript and now AJAX)
- a native security model (SSL).

Unlike Java, web browsers offer inherently scalable display canvases and a finite (and diminishing) number of permutations (MS-IE, Mozilla/FireFox, Opera, WAP and increasingly, Webkit). A *web app* approach is of course advocated by browser suppliers like Opera, by web content and services companies like Google, by low-end PC ventures like gOS, and, when previously lacking a native SDK, as a short path for iPhone applications. A recent validation of *web apps* as a unifier can be found in Mozilla.org joining the LiMo Foundation.

With both today's EDGE and 3G and tomorrow's 4G connectivity and bandwidth, web APIs, AJAX and browser widgets present a very short path to unification of the mobile programming space. Linux, as an enterprise-class OS, is well positioned to take advantage of this paradigm: Linux-based server-side software can easily discover client-based browser attributes, and practically all browsers[15] on the market already target the Linux OS.

Approaches to Unification

Choice and diversity are the hallmarks of open source software. Diversity is good, but mobile Linux can have the appearance of diverging at nearly all levels: Linux kernel and library versions, user interface (UI) and application frameworks, IPCs, telephony and multimedia APIs, etc.

[14] But, importantly, not the Flash run-time code.

[15] Excepting Microsoft Internet Explorer!

If choice is the glass half-full, then fragmentation is that same glass half empty. Fragmentation in a technology marketplace is undesirable because it can

- Dilute development efforts and applications on any single platform
- Limit interoperability among devices based on diverse s/w platforms
- Raise costs across the ecosystem for development and support
- Slow or inhibit adoption of underlying common technology

Conversely, choice and diversity in both open source and traditional commercial software have even greater virtues by:

- Providing a low-risk "laboratory" for experimentation, prototyping and peer review
- Helping to reduce costs over time as "bottom up" consolidation spreads costs among community members
- Fostering adoption of both common and competing technologies through openness
- Ultimately enabling the "best in class" technology to win on merit, not consortium politics

As illustrated in the following sections, there exist multiple approaches to unification, but no single straightest path to platform unity. The approaches are not mutually exclusive, and do face challenges, but also have much to recommend them:

Traditional Standards

The global mobile/wireless industry is built on standards. Mobile users can roam the globe and communicate reliably at home because of GSM, CDMA and other networking standards. Organizations like OMA (Open Mobile Alliance) and OMTP (Open Mobile Terminal Platform) promote standardization of mobile applications interfaces for synchronization (SyncML), telephony, mobile video, etc. The convergence of mobile and fixed wireless entails bridging to standards in the 802.11 family; rich peripheral support in today's smartphones encompasses implementations using USB, Ethernet, SDIO, and other interconnect standards.

Challenges to Traditional Standardization

The importance and success of wireless industry standardization notwithstanding, critics call out three factors that can limit the utility of standardization as a force for unification:

Open and Closed Standards – The dominant standards in mobile/wireless today are not open. Even with open standards, ecosystems often lean towards closed implementation: openness is still a relatively new idea to this industry, and some government regulation[16] can appear to favor closed over open. However, existing telecoms standards are successful, widely adopted and can provide a model for mobile Linux. The good news is that in telecommunications and in other industries, standards are opening up over time (e.g., the ongoing WiMax rollout).

Leading vs. Trailing Compliance – The presumed outcome of a successful standardization effort is multiple implementations, each cleaving to the public standard specification via certification and compliance testing. However, real-world pressures can limit the scope of compliance regimes, which by definition and design arrive later in the standardization life cycle (a standard must exist before implementations can comply to it). As such, many compliance disciplines are "works in progress", leaving wiggle room for commercial implementers. The result can be divergent implementations that lock-in adopters to commercial implementations, or vague specifications and broad testing. The upside of delayed certification is that it offers real opportunities for

[16] E.g., U.S. F.C.C. regulations limiting changes to device drivers in certified radio devices.

implementers to add value and to help the specification better track real-world usage.

Pace – Despite attempts at "fast track" standardization, traditional efforts suffer from the "3 Cs": Committee, Consensus, and Compliance – definition work takes place in *committees* and work-groups, usually as a part-time members activity; divergent interests in those committees must reach *consensus* and often do so by choosing the least controversial option under consideration; *compliance* regimes with challenges described above. One should not judge this stately process too harshly – early rigid standardization can stifle innovation and runs counter to today's dynamic community development processes.

OSS and Standards – Bridging a Language Gap

Standards organizations are trying to change methods to leverage fast-moving, shared open source as a new medium. However, standardization and open source appear to speak different languages. Deliberation around traditional standards focuses on *requirements*, *specification* and *compliance*. OSS dialogue centers on *implementation* in actual source code.

Philosophically, standardization and open source must bridge a gap between them. Vendors build standards from commercial interest, guided by large participants' agendas. Standards body members implement and license specifications, derived IP and conformance regimes to third parties, By contrast, OSS communities focus on free sharing of implementations and APIs.

OSS is not anti-standards: free and open source software has a long history of standards-based implementation (e.g., POSIX and TCP/IP RFCs in Linux; HTTP, HTML, and CSS in Apache and Mozilla, etc.). However, standardization of OSS has a mixed history. Efforts such as the Linux Standards Base (LSB) began as "trailing" standards, describing commonality among enterprise distributions; today LSB tries to provide a leading definition for the Linux OS. Community voices initially rejected specifications like Carrier Grade Linux as unfunded mandates; instead CGL requirements made their way, piece by piece, into mainstream Linux code.

However, unlike building proprietary software, OSS development doesn't routinely employ (proprietary and expensive) compliance suites. Preferred is community-based use testing. Compliance is a recent and challenging requirement only recently comprehended by OSS developers.

Ad hoc Standardization by Companies and Consortia

De facto standards build on consortium-based and vendor-sourced implementation. Consortia posit "get the right players in the room, give them code, and everything will be fine." Vendor vision entails building devices so ubiquitous that developers beat a path to suppliers' doors and mailing lists.

Both approaches face challenges. Consortia suffer from member inertia, reluctance to share IP and competing interests. Code quality can also be an issue – do members share their best code or merely *leftovers*? Vendor-sourced implementations risk lock-in: can companies really aggregate and support communities around stacks and handsets?

Also worrisome are "IP Clubs" that restrict access to useful code. "Openness" shouldn't mean selectively exposed APIs, meted out to developers as marching orders. Similarly, ISV stacks and OEM reference handset designs can be hampered by closed code, licensing restrictions, cost and availability. Optimally, consortium and vendor implementations bridge to open source by sharing code, using suitable licenses, accepting patches and cooperating with related projects. And increasingly, consortia and consortium members are doing just that.

A Pragmatic Path to Unification

The preceding description of the status quo in mobile Linux and options moving forward might seem grim. A pessimistic assessment points to eschewing the mobile Linux *status quo* and choosing to *wait and see*, or giving up and developing for other platforms.

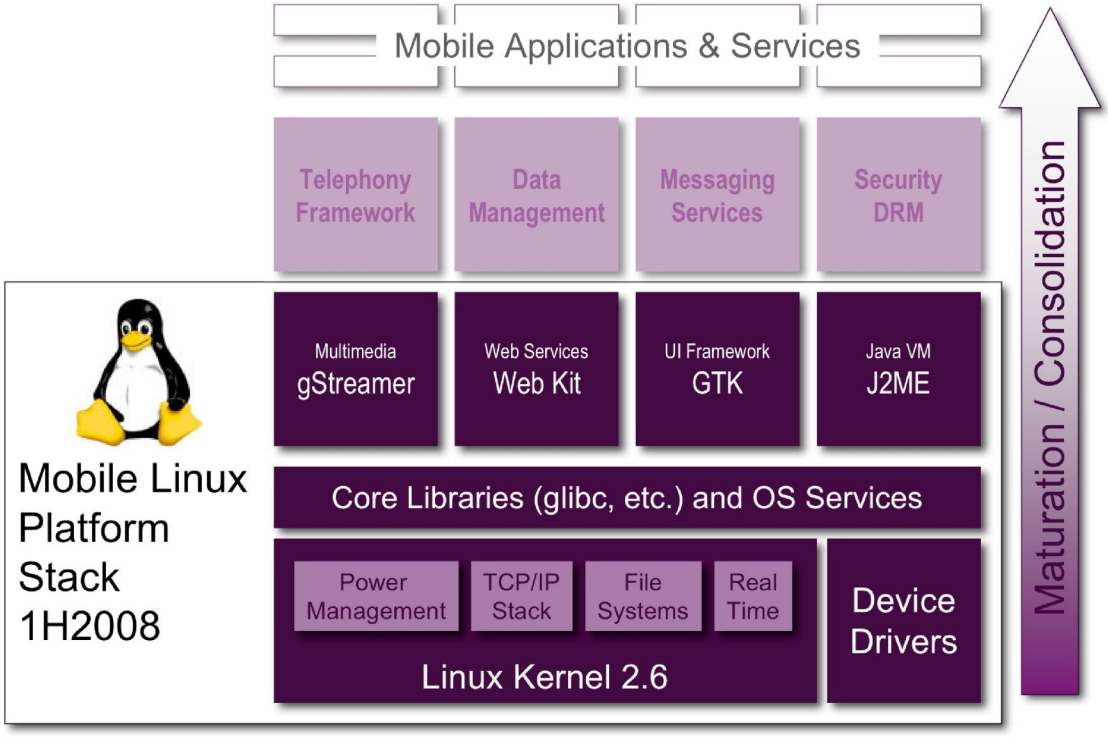

Figure 4. – The state of evolution in the mobile Linux stack

Actually, mobile developers, OEMs and other players, have more latitude within the scope of mobile Linux. The sponsors of this white paper point to the need for a pragmatic approach, compatible with the realities of shipping products on schedule, and emphasizing close ties to the mobile ecosystem.

Incremental Optimism

The operational premise of this white paper and of its sponsor is a more pragmatic and less doctrinaire approach, and certainly a more optimistic one. It is based on the following principles:

- While Linux and FOSS are generally very mature after 15 years of development and deployment, Linux in mobile is still evolving
- This evolution is occurring rapidly and from the bottom of the stack "upwards"
- The only way to leverage the power of the emerging mobile Linux platform is to proceed incrementally

The good news is that this increasingly rich base platform now includes, off-the-shelf, a mobile-optimized Linux kernel, most of the needed device drivers, core libraries and frameworks and services for multimedia, web, user interface and growing choice of build and prototyping tools. The less-good news is that key enablers for telephony, data management, messaging, security and DRM remain in flux or do not yet exist at all in deployable form.

Steps for Success

The realities of the state-of-the-art dictate the following steps for OEMs and ISVs to follow. Indeed, these steps describe the typical activities of these companies in supporting their customers – handset OEMs and operators.

1. Start with Stable Base

Working at or below value line, choose either a commercial platform base (as in Figure 4. Above) or integrate your own from readily available open source components: Linux kernel, drivers, libraries, GTK, gstreamer, webkit, etc. The result will likely not be monolithic and your team will have to enumerate the "holes" in the base platform: missing drivers, CODECs, etc. Bridging these gaps, off-the-shelf, constitutes a strong added-value from and incentive for working with commercial mobile Linux suppliers.

2. Bridging Gaps

You, your partners or subcontractors will next need to work to fill holes in the base platform, mend API seams where components meet, and start bridging gaps on top (additional application enablement and applications themselves) with a practical mix of proprietary IP and/or commercially enhanced OSS components. Again, if your organization sees its primary expertise and value-added in manufacturing, applications, channel, etc., you will value working with a mobile Linux ISV.

3. Track Community Progress

The next steps are open-ended. New developments in both community and commercial technology proceed at their own pace. Your team will have to monitor and preferably work with community developers on creating and fostering advances in emerging OSS code. This activity gives your team a unique opportunity to innovate and influence the direction of the mobile Linux platform, but confers no guarantees of short-term success or controlled ROI. For many companies – OEMs, ODMs and also for operators – this type of activity may fall outside their comfort zone, expertise and available budget. For this reason many commercial mobile companies choose to work with ISVs rather than "rolling their own" platforms and dedicating resources to community interface and tracking.

4. Evolve the Base

As you discover and help to create a richer base platform, you, your partners and community maintainers will need to integrate new elements into the evolving platform. In some cases you will need to "back port" new components onto the maturing base; in other cases you and others may end up porting and migrating platform components forward to mesh with new kernels and versions of key software components (e.g., gstreamer).

Don't try to "go it alone" – costs to build and maintain proprietary patches to community code will soon outweigh any short-term benefit realized from "hoarding" the differentiating code. And don't attempt to move too far ahead: stability trumps any positioning advantage perceived in leading/bleeding edge adoption.

5. Repeat as needed!

Like all software development, and especially like open source development, technology providers need to iterate, to keep the emerging platform "alive".

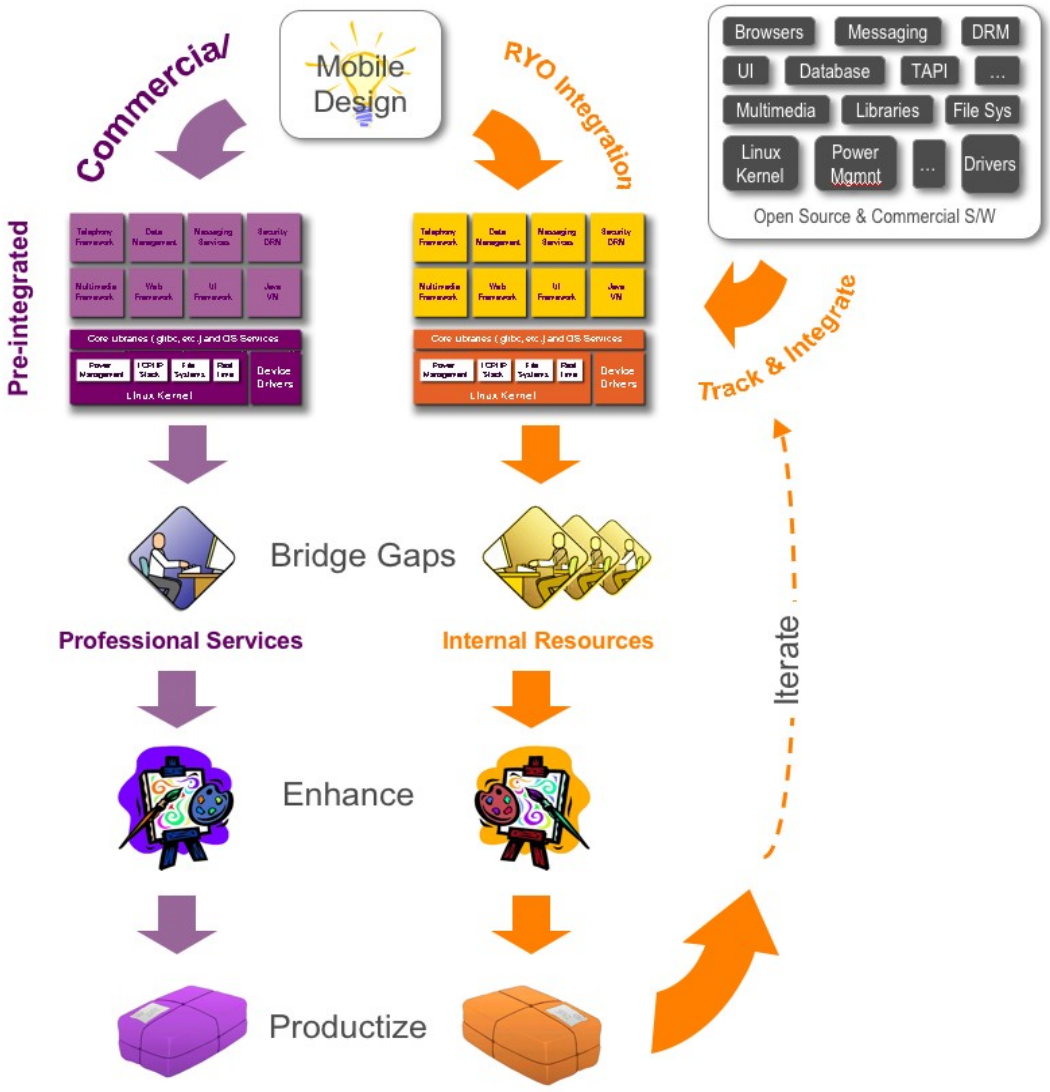

Figure 5. – The Life Cycle of a Mobile Linux Platform

Conclusion

For mobile Linux, as with enterprise Linux, Apache, Mozilla, GNU, Eclipse, other successful open source technologies and projects, an incremental approach yields the best results over time; smaller deltas mean lower risks and more opportunities to align and work with community and ecosystem forces. Despite the feeling that Microsoft is "breathing down our necks" with Windows Mobile, the announced open source Symbian Foundation, and growing numbers of SymbianOS design wins and shipments, the mobile Linux ecosystem still has time to experiment, and the platform time to mature, to blossom.

Individual platform developers, ISVs, OEMs and other interested parties can have tremendous impact on the evolving platform, but not alone and not overnight. They must gain expertise and credibility with the platform building blocks and the glue that binds them together. And platform developers must have the patience, even the humility to change course to follow community and ecosystem adoption trends: even as you invest in a project or an implementation, you must be prepared to change course to follow the prevailing trend. "Swimming upstream" will only exhaust individual and shared resources.

ISVs and OEMs have an advantage over purely free software development: as community process participants, they can combine IP and services *on purpose,* to build products, not just tinker with platforms. In the short term, this kind of integration focuses on getting to market; in the longer term, it will deliver an increasingly universal, interoperable, shared mobile Linux platform.

References

GOLDEN, Bernard, and WEINBERG, William [2006-2008]. "Open Source and the Entrepreneur: Real Value, Real Business." *Proceedings of SD West 2008 Conference*.

KINGMAN, Henry [2007]. "Pundits weigh in on Android". LinuxDevices.com, November 6 edition.

WEINBERG, William [2007]. "The F Word Fragmentation of Emerging Mobile Linux" *Wireless Week*, September 25 edition.

WEINBERG, William [2006]. "Mobile Phones: the Embedded Linux Challenge" *Linux Journal*, July edition.