OSBR.CA

The Open Source Business Resource

MARCH
2010

MARCH 2010

PUBLISHER:
The Open Source Business Resource is a monthly publication of the Talent First Network. Archives are available at the website:
http://www.osbr.ca

EDITOR:
Dru Lavigne
dru@osbr.ca

ISSN:
1913-6102

ADVISORY BOARD:
Tony Bailetti
Leslie Hawthorn
Chris Hobbs
Rikki Kite
Thomas Kunz
Michael Weiss

"Today's modern handset represents a 'melting pot' of communications and multimedia technologies."

R. Wietfeldt, Texas Instruments

Mobile is the editorial theme for this issue of the OSBR. This month's authors provide an overview of how open source fits into the world of handheld mobile devices, discussing everything from the hardware to the software applications running on the device. Their discussion is not limited to mobile phones as they cover other aspects of the complete mobile system, including transmitters and receivers. It is our hope that their insights prompt you to think about open source the next time you reach for your mobile device.

As always, we encourage readers to share articles of interest with their colleagues, and to provide their comments either online or directly to the authors.

The editorial theme for the upcoming April issue of the OSBR is Cloud Services and the guest editor will be Mike Kavis. Submissions are due by March 20--contact the Editor if you are interested in a submission.

Dru Lavigne

Editor-in-Chief

Dru Lavigne is a technical writer and IT consultant who has been active with open source communities since the mid-1990s. She writes regularly for BSD Magazine and is the author of the books BSD Hacks, The Best of FreeBSD Basics, and the Definitive Guide to PC-BSD.

Open source software and hardware has become an accepted way of developing new and interesting applications in many information and communication technology domains: operating systems, databases, Web infrastructure, and applications. It's not surprising that with the increasing popularity of mobile handheld devices, users and researchers have explored the power of open approaches to providing innovative new applications and services in this domain. However, unlike personal computers and the Internet, mobile handsets were tightly controlled by mobile network operators (MNOs) who developed a vertical ecosystem by integrating the communication infrastructure, the handheld device hardware, and often the applications installed on those devices. The software and protocols running the mobile communications infrastructure and devices are often standardized by membership-only bodies, where large MNOs and manufacturers have a predominant influence. These players invest significant financial resources into shaping the industry along their vision to gain a competitive advantage. A current example is the ongoing battle about the dominant radio access technology for 4G cellular systems: LTE vs. Wimax.

These trends have changed recently. Companies such as Google, Nokia, or Openmoko and Industry Alliances such as the Open Handset Alliance are providing the core building blocks, both in hardware as well as software, of increasingly open mobile devices. This issue of the OSBR reviews the relevant trends in the open mobile platform space from a number of perspectives. As the articles in these issue show, there is a lot of exciting ongoing work that brings the power of open source development to the mobile space. This trend is not just confined to the mobile devices as there are also efforts in the development of open mobile infrastructure elements and whole systems.

Andreas Constantinou is the Research Director at VisionMobile. His article discusses the importance of governance models to understand the dynamics of an open source product, constrasting it to the better understood role of licences. Using the mobile industry as an example, he demonstrates how governance models can be used by open source sponsors to control the development of open source products, and argues for more education and clarity on governance models.

Jason Kridner is the open platforms principal architect at Texas Instruments Incorporated. His article discusses the challenges and successes in establishing a vibrant ecosystem around the BeagleBoard, a low-cost, fan-less single-board computer. The efforts within this community have allowed the BeagleBoard to become a versatile and powerful open embedded device.

David Burgess of the OpenBTS Project discusses the project's experiences, which will probably become the first case of a free software GSM basestation in a public cellular network. The article focuses on the challenges of the project, as well as the advantages of having followed the open source route.

François Lefebvre leads the Mobile Multimedia Broadcasting team at Communications Research Centre, Canada. His article surveys CRC's attempt to increase collaboration and innovation in the field of mobile broadcasting by developing and offering complete end-to-end free and open source software toolsets.

Carl B. Dietrich, Jeffrey H. Reed, Stephen H. Edwards and Frank E. Kragh discuss OSSIE, a university-based open source Software Defined Radio project at Virginia Tech. OSSIE software has proven useful for rapid prototyping by industry as well as for published research and education of hundreds of graduate and undergraduate students. In addition to examples of OSSIE's successes, the project's challenges and approaches to mitigating and overcoming them are described.

Hal Steger, Vice President of Marketing at Funambol, inc., introduces the cloud computing paradigm as a way to deliver mobile applications and data. His article discusses trends that are driving the adoption of the mobile cloud, important components of mobile cloud infrastructure, and the role of open source.

Bradley M. Kuhn is the Policy Analyst and Technology Director at the Software Freedom Law Center. He briefly reviews the history of free software in the mobile device space, focusing on both software and hardware. A review of the available alternatives to-date leads him to conclude that users, while able to access open code bases from major companies, are at the mercy of these companies. For a number of reasons, true software freedom on mobile devices is, as yet, an elusive goal.

Thomas Kunz, François Lefebvre
Guest Editors

Thomas Kunz received a double honours degree in Computer Science and Business Administration and the Dr. Ing. degree in Computer Science from the Technical University of Darmstadt. He is currently a Professor in Systems and Computer Engineering at Carleton University. His research interests are primarily in the area of wireless and mobile computing. The main thrust is to facilitate the development of innovative next-generation mobile applications on resource-constraint, hand-held devices, exploring the required network architectures, network protocols, and middleware layers. He authored or co-authored close to 150 technical papers, received a number of awards, and is involved in national and international conferences and workshops. Dr. Kunz is a member of ACM and the IEEE Computer Society.

François Lefebvre joined the Communications Research Centre, Canada, in 1999 to lead its Mobile Multimedia Broadcasting team. Since then, he has contributed to numerous national and international standardization efforts and R&D projects. His recent work has focused on creating and developing open software building blocks for next-generation mobile broadcasting networks, devices and applications. With his team, he launched the CRC mmbTools and Openmokast open source software projects. He writes about the future of broadcasting on his blog Broadcasting 2.0 (http://www.broadcasting20.org). Mr. Lefebvre graduated from Laval University in Electrical Engineering where he also completed his M.A.Sc. in 1989.

"Open source licenses tell only half the story. The governance model, the implicit rules defining transparency and influence into an open source project, is the small print that determines the power dynamics around that project."

Andreas Constantinou

Openness is a much-misunderstood word. It represents a kind of good-will moniker to which people attach an impressive variety of definitions: open source code, open standards, open handsets, openness as in transparency, shared roadmaps, open application programming interfaces (APIs), open route to market, and so on. It is a very forgiving term as far as definitions go.

One of the mobile industry's favourite facets of openness is open source code. Since 2007, tens of mobile industry giants and consortia have embraced open source in some form or other: the Symbian Foundation (http://symbian.org), LiMo Foundation (http://limofoundation.org), Google's Android (http://android.com), Nokia's Qt (http://qt.nokia.com), Apple's WebKit (http://webkit.org) and Nokia+Intel's Meego (http://meego.com) are the initiatives that have hit the industry front pages. On the surface, these initiatives use open source licenses, but that only tells half the story. Behind the scenes, Google, Apple, Nokia and others use restrictive governance models and control points that effectively detract the very freedoms that open source licensing is meant to bestow. This article discusses the many forms that governance models can take, and how they are used in the mobile industry to tightly control the roadmap and application of open source projects.

What is Open Source?

Open source licensed software carries four basic freedoms that provide the right to access, modify, distribute and contribute to the software (http://gnu.org/philosophy/free-sw.html). These freedoms have been expanded into ten cardinal points that form the criteria that every open source license must adhere to, and which are defined by the Open Source Initiative (http://opensource.org/docs/osd).

A handful of different licenses (http://opensource.org/licenses/alphabetical) are used in the vast majority of open source projects; namely the GPL, LGPL, APL, EPL, MPL, BSD and MIT. Interestingly, the GPL which is known as a strong copyleft license (http://en.wikipedia.org/wiki/Copyleft), is most often used in personal computer and Internet projects, but is rarely used in the mobile industry. Instead, the licenses used most often in mobile software are weak copyleft, such as the EPL, or permissive licenses, such as the APL, due to handset manufacturer concerns for downstream liabilities.

Licenses are Half the Story

What's often missed in open source discussions is how open source licenses tell only half the story. The governance model, the implicit rules defining transparency and influence into an open source project, is the small print that determines the power dynamics around that project.

In practice, mobile open source initiatives use a variety of control points - such as trademarks, private lines, distribution of derivatives, ownership of reviewers, gravity of contributions and contributor agreements - to turn an egalitarian governance model into an authoritarian one. Such control points can detract the very freedoms that open source licenses are meant to bestow, examples of which are shown in Table 1.

Table 1: Control Points Used in "Open" Mobile Projects

Open source criteria	Mitigation (control) points
Software source code can be accessed freely and easily	Android uses private lines which are feature-wise 6+ months ahead of the publically released SDK. It would be impossible for OEMs to release competitive handsets by relying on the public SDK alone.
Software can be modified freely	In Android, OEMs can modify software freely, but they must rely on Google exclusively to introduce that modification back into the main codebase.
The license must not discriminate against any person or group of persons	In many cases, the product roadmap or latest features are only available to members of a paid-for or invite-only club.
No discrimination against fields of endeavour	Sun and Google have secured Trademarks on the Java and Android brand names respectively; this implies that even if software can be distributed freely, it cannot be called Java or Android unless by prior agreement.

In general, open source licenses pertain to the use of the source code in a purist sense, while governance models relate to the use of a product in a practical sense. Governance models are much more fundamental in determining the ecosystem dynamics around that product.

Mapping Licenses vs. Governance Models

Governance models can be simplified to indicate the democracy of transparency and influence on an open source product. On one end of the spectrum are autonomous trust communities where opinion leaders influence the direction of the product through a chain of trusted reviewers. An example of this type of model is Linux. On the other end are single-sponsor communities where the product roadmap, private lines, contributions and trademarks are controlled by a single company. Based on this simplified governance model, we map the most popular mobile open source projects, as seen in Figure 1.

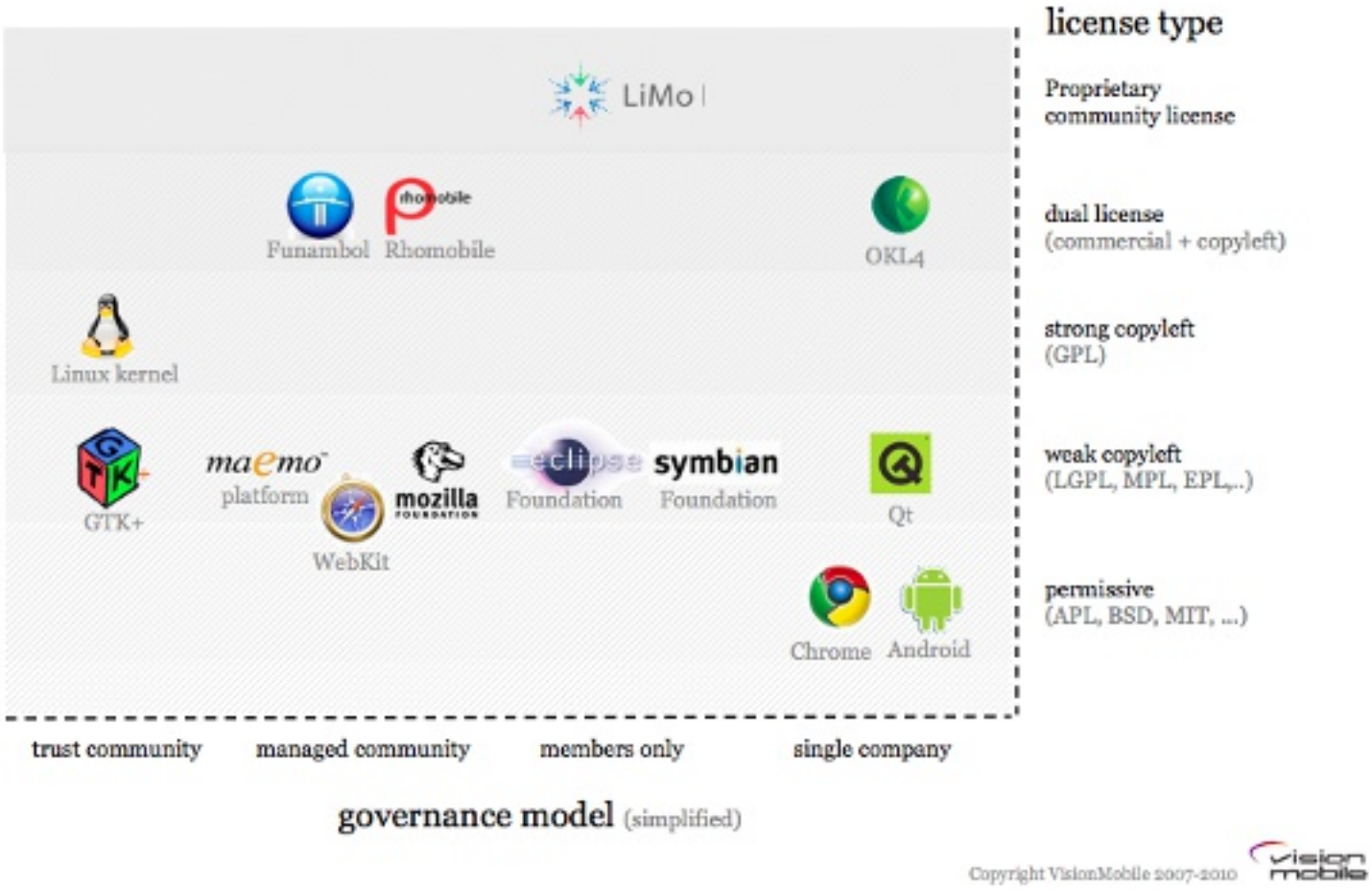
The picture that emerges is one where:

- open source licenses (the large print that covers source control) are widely used, converged and well understood, while
- governance models (the small print that governs product control) are proprietary, diverging and poorly understood

The Maze of Governance Models

There have been many attempts at classifying governance models, but there is really no universal dictionary, no certification body, and an excessive amount of open marketing hype to help obscure rather than enlighten the mobile industry. This is further exacerbated by the fact that governance model particulars are usually available under confidentiality terms. They are rarely critisized and debated in public, in the way, for example, an intrusive privacy policy would be.

Figure 1: License vs Governance Models



How do open source initiatives control access to a product, manage influence mechanisms, or intellectual property (IP) rights for contributions? We have attempted to provide some of the key questions that should be asked when determining the level of openness of a governance model:

- Is the latest source code publically available or are feature-packed private code lines available discriminately to selected parties?
- Is the product roadmap publically available or to members only?
- How is the roadmap formed and who has voting rights?
- Is there a transparency of contributions in terms of the level and type of contributions by partipating companies?
- Is the amount of contributors and reviewers/maintainers balanced among the community or does one company form the gravity centre of contributions?
- Can anyone become a code reviewer or maintainer? (upstream influence)
- Can contributions be deemed as mandatory? (downstream influence)

- Are derivatives available widely, or through a proprietary route to market, such as seen in the Android Market?

- Can you fork the software and buy the service from somewhere else?

- Is there a trademark applying to the use of the project name and embodiments of products?

- Do contributions require copyright assignment or patent grants?

- Are there any safe harbour provisions (http://en.wikipedia.org/wiki/Safe_harbor) for contributors to the source code?

We believe that governance models are the key determinant of the ecosystem dynamics, and the level of innovation around open source projects.

A Call for Education

What the mobile industry needs is not more marketing hype around the benefits of openness, but more education and clarity on governance models. The industry also needs a benchmark – a sort of openness index - for determining the true dynamics of an open source product, and for pushing the corporate sponsors to play fair. At VisionMobile (http://www.visionmobile.com) we have been quietly working towards developing an openness index and are keen to hear from companies who want to make this happen.

Governance is one of the most understated topics in the open mobile industry today, yet one of the most fundamental in the direction the industry will be taking.

Andreas Constantinou is the Research Director at VisionMobile where he oversees the company's research, advisory and industry mapping projects. He has ten years experience in research, development and strategy in mobile, specialising in the handset ecosystem, software strategy and open source. Andreas has worked on several product and marketing strategy projects for clients including Sony Ericsson, RIM, Microsoft, France Telecom, T-Mobile, OMTP, Qualcomm, ST Ericsson, Gemalto and Trolltech and authored numerous research reports for analyst firms Informa, Ovum and ARCchart. Andreas also teaches the Mobile Open Source workshop, part of VisionMobile's 360 degree training courses on complex industry sectors. Prior to founding VisionMobile, Andreas spent 3 years at Orange's Research and Innovation division, including serving as a technology lead for the Orange-Microsoft relationship. His interests include uncovering under-the-radar industry trends and pursuing human-centric design. When not hopping on planes, Andreas spends his time in Athens, Greece. Andreas holds a Ph.D. in Image & Video Compression from the University of Bristol, UK.

"If you're a developer, you want to be where you can fulfill your vision. If you're a consumer, you want to be where you can do what you want to do. If you're a handset manufacturer, you need to be where that innovation happens."

Troed Sångberg
http://blogs.sonyericsson.com/troedsangberg/speed-of-innovation

There should be little doubt that any given area of technology will eventually be occupied by open solutions. Rather, it is a question of "who" and "when." In mobile and embedded platforms, there is not a single dominant proprietary solution to displace, unlike desktop software. Instead, open software produces interface and compatibility experiences on par with proprietary software.

At points where no proprietary solution is clearly differentiated from open solutions and the barriers to participation are kept low, open innovation thrives at the forefront of the technology. This article introduces BeagleBoard.org, a project that creates powerful, open, and embedded devices based on the BeagleBoard hardware, a low-cost, fan-less single-board computer. By lowering the barriers to participation and making commitments to support and sustain the architecture to preserve the innovation from that participation, BeagleBoard.org has built an active and growing community of hobbyists and professionals advancing the state of the art in low-power embedded computing.

What do People do with a BeagleBoard?

I am asked frequently if I am surprised by the rapid growth of the BeagleBoard.org community. What does not surprise me is the number of people who quickly recognize that there is something different about the BeagleBoard – it is something accessible and capable – and then want to get involved.

However, I am certainly surprised by what people do with their BeagleBoards. Some examples include:

- an Iron Man costume with digitally controlled air-rocket launchers
- amateur radios that analyze wave contents using software
- robots that know how to stand and walk on their own
- three dimensional digital cameras
- autonomous flying vehicles that recognize objects around them to avoid collision and redirect their path
- miniature presentation projection systems that utilize 3D objects and animation to keep the audience interested
- high-definition (HD) media centers
- car computers with touch-screens showing video and virtual instruments with data logging
- touch-screen computers that hang on a refrigerator

This is just a small pool of the hundreds of ongoing projects driven by thousands of hobbyist and engineering individuals and team members. People want to do more with their mobile or low-power embedded computing platforms than place phone calls. They want to collect and process data from the physical environment and manipulate that environment. They want to move data over a variety of media and in a variety of formats.

When given a computing platform that is sufficiently high-performance, low-power, expandable, affordable, small, and supported by tools and software stacks, developers with the capability to

extend that computing platform, either through hardware or software, will simply do so. BeagleBoard.org creates an open innovation ecosystem that advances everyone using the OMAP™ 3 processor-based BeagleBoard by encouraging open source developers to utilize all the available tools wherever it makes sense in their designs.

Understanding the Community Landscape

A key benefit to collaborating with open source developers is that they are vocal about their specific wants and needs. One of the best ways to attract these developers is to simply listen and meet their demands.

LugRadio (http://lugradio.org), which was a bi-weekly audio download series that featured the Canonical Ubuntu community manager Jono Bacon (http://www.jonobacon.org), nicely summarized the primary characteristics and motivations of open source developers. Each of these characteristics is a way in which the developer or innovator is able to gain some aspect of control or influence. The types of open source developers can be summarized as:

Community Participants: are characterized by enjoyment of attention and being part of something big, where they can rub elbows with influential people and improve their own profile within their desired sphere.

Tinkerers: are characterized by a desire to know how something works and have enough control to change how it works or how it is assembled.

Underdog Fans: are characterized by a desire to see the dominant influences in a realm of technology challenged by up-and-comers that limit the control of any single entity.

The Cheap: are characterized by the desire for access to be universal or at least have access be of minimal cost to them.

Freedom Crusaders: are characterized by seeking prevention of any control over them.

We discuss each of these participants further below and describe how they benefit from the BeagleBoard ecosystem.

Community Participants

To meet the demands of Community Participants, it has to be clear from the beginning that the community is large enough to justify their time. Given that the BeagleBoard is based on the OMAP 3 platform (http://en.wikipedia.org/wiki/Texas_Instruments_OMAP), the first broadly available device with an ARM® Cortex™-A8 CPU, little convincing was required to reflect that there would be a large pond of participants.

That same processor will be used as the basis for the next generation Maemo.org device. This signaled to the Maemo open source developer community that they would get early access with the confidence that their time was being spent in an area where plenty of other developers would notice their work. An English-language mailing list was created to be a focal source of announcements and queries about community developments. A single RSS feed was created to aggregate the news published on blogs of developer activities. Today, there are thousands of mailing list subscribers and thousands of articles about BeagleBoard activities with many answering complex questions about how to design something with the BeagleBoard. The RSS feed provides a centralized location to view the latest articles, and several new articles are published every day.

The Tinkerers

Many are tired and frustrated with the way computers work, or don't work. Tinkerers were born to do something about it. They love source code, but source hardware is even better. The source code to the operating system and the source hardware design files describe how the BeagleBoard operates and how it is built. The source allows Tinkerers to not only understand the inner workings, but to alter them. Dozens of systems that are open for adding more software or hardware have been built using the BeagleBoard as a starting point, with each adding some value beyond the BeagleBoard while growing the shared software base that enhances what the BeagleBoard can do. The BeagleBoard is based on a processor with multiple vendors, which further extends the sources that can be used by Tinkerers to build and experiment with software. Today, there are several OMAP 3 processor-based platforms built from the BeagleBoard with the flexibility to add new capabilities, such as those seen in Table 1:

Developers have the ability to add many other physical world interfaces, all in a variety of form-factors, and many much smaller than the BeagleBoard itself. The BugLabs Bug 2.0 (http://buglabs.net) provides a plugin module architecture with an online store for people to sell their own modules that utilizes OMAP 3 and BeagleBoard support to run the Android operating system.

Underdog Fans

Underdog Fans are attracted to the challenge that BeagleBoard represents to the entrenched virtual monopoly held by the processors used in typical desktop computing. As embedded microprocessors utilizing multi-vendor ARM (http://en.wikipedia.org/wiki/ARM_architecture) processor cores move up in performance at much lower power points, lower prices, more integration and specialized processing resources for some markets, many designs that would have used a traditional PC architecture are now choosing to switch. Fans help notify those who have not considered such a switch that a viable solution exists. With the influence of these Fans, who are frequently key influencers in medium and large organizations and developer communities, the BeagleBoard is being picked up by people migrating from: i) desktop or server operating systems like Ubuntu, Debian, and Gentoo; ii) boot firmware; iii) media center software; and iv) many other components not traditionally targeted at embedded processors.

The Cheap

The needs of the Cheap are partially satisfied by the bring-your-own peripherals approach taken by the BeagleBoard. Instead of being constrained to a single configuration where everyone needs to pay for all the possible features of the system, the BeagleBoard makes use of off-the-shelf expansion via standard inter-

Table 1: Possible BeagleBoard Capabilities

touch-screens	WiFi	Bluetooth
3G modems	multi-axis accelerometers	digital compasses
3D air mice	eyeglass monitors	keypads
wired industrial serial/network interfaces	various RF demodulators/modulators	RF communications
temperature sensors	various audio/signal converters	storage expansions
motor servo controllers	power monitors	digital cameras
general purpose I/Os	global positioning systems	projection displays

faces, allowing developers to utilize components that they already own or allowing them to choose the components they prefer. There has been some resulting complexity for users, yet the lower-cost generates a far greater reach and lower threshold for purchasing decisions. The limitations created also open doors for value-added resellers, such as Special Computing (http://specialcomp.com), who become participants in the community. Although Special Computing has been successful creating some bundled kits with the BeagleBoard, the overall success and adoption of preconfigured kits has been limited. The wide variety of demands could be to blame, and improvements to the BeagleBoard.org website in positioning bundles could be a resolution.

What has taken hold is the use of the BeagleBoard in a large number of demonstrations for various hardware or software products. Some examples include demonstrations of hardware products interfaced to the BeagleBoard:

- Hillcrest Labs Freespace (http://hillcrestlabs.com/products/freespace.php) in-air pointing devices
- TI DLP Pico Projector development kit (http://focus.ti.com/dlpdmd/docs/dlpdiscovery.tsp?sectionId=60&tabId=2235)
- TI eZ430-Chronos wireless watch development tool (http://focus.ti.com/docs/toolsw/folders/print/ez430-chronos.html) with many personal and environmental sensors

Other examples include demonstrations of software products or concept demonstrations:

- Android implementations from more vendors than I can count, including the ARowboat.org project
- MontaVista's (http://www.mvista.com) MVL6 and Montabello
- Timesys's LinuxLink (https://linuxlink.timesys.com)
- ARM's (http://www.arm.com) ARM Linux Internet Platform and D5
- Halcon machine vision from MVTec Software GmbH (http://mvtec.com)
- MPC Data's (http://mpc-data.co.uk) Windows® Embedded CE
- QNX (http://www.qnx.com)
- Ingenient's (http://ingenient.com) 720p TMS320C64x+ processor-based video codecs in Android
- ARM-based video codecs from Visual On (http://www.visualon.com)
- Softkinetic-Optrima (http://softkinetic-optrima.com) OptriCam
- ASTC's (http://www.astc.org) Beagle Board simulator
- RidgeRun (http://www.ridgerun.com)

The Freedom Crusader

One challenge that must be met to keep the Freedom Crusader happy is to ensure BeagleBoard is not locked down to a single vendor. A key difference between the BeagleBoard and other ARM Cortex-A8 development platforms today is that all of the components are available in low quantities through the catalogs of electronics distributors, and the documentation for the critical components is available for free online. Also, since ARM processors can be licensed by just about any silicon manufacturer or vendor, it is conceivable to even replace the core processor.

There are programmable elements available to the ARM processor for which there are programming guides and freely available tools. Thanks to the open programmability of the digital signal processor, which is optimal for processing video and audio data, projects such as the open source Ogg Theora (http://.theora.org) video decoder thrive and provide the Crusaders with something to crusade about.

Building a Community Around Active Participation

The great success of building a community on the BeagleBoard comes not only from meeting the needs of the open source and open hardware developer communities, but by having core developers act as key participants in both the culture of quality products and the culture of open. Whenever tough decisions need to be made, the default choice is the one that involves getting out of the way of innovators and letting the community lead. The more an individual participates and contributes, the more ownership is required on the direction of the project.

Further, continued improvements and innovation are required to keep people excited. This summer, the next revision, the BeagleBoard XM, will be running at 1GHz or higher with enough RAM to natively rebuild its entire application stack. This is an important milestone in the evolution of any computing hardware. The choices made in the design of the BeagleBoard XM reflect the guidance of the community members. Despite having some features that have an effect on the cost for the new product, the commitment to maintain the existing product at the existing price point is being held. This sort of commitment to the existing community is absolutely required.

The readily apparent innovations in the available commercial and/or open source products look great, but how do we understand the full value of community involvement? Perhaps the best indicators are the customers who have already seen the BeagleBoard and likely played with it before the first sales call is ever made for a new design.

Jason Kridner is the open platforms principal architect at Texas Instruments Incorporated. He is passionate about pervasive and accessible computing platforms. Kridner graduated from Texas A&M with a bachelor's degree in Electrical Engineering and was drawn by the allure of digital signal processing to TI in 1992. He began as a hardware developer, working on board, FPGA, and ASIC designs. Utilizing software experience prior to TI, Kridner transitioned to lead software development of low-power media software, audio processing, file systems, USB drivers, digital rights management, and video codecs. He now defines software architectures that enable a broad body of developers on TI's ARM and DSP based catalog processors.

"What would the world look like if cellular networks were open-sourced?"

Alec Saunders

http://saunderslog.com/2009/09/01/opensource-meets-gsm-at-the-burning-man

In mid-2007, Kestrel Signal Processing, Inc., a small software radio consulting shop in northern California, started writing an implementation of a GSM (http://en.wikipedia.org/wiki/Gsm) basestation. The initial developers were Kestrel cofounder Harvind Samra and myself. Our goal was to create a new kind of lightweight cellular network that could be built out inexpensively in remote and sparsely populated areas. Our software-radio GSM system, now called OpenBTS (http://openbts.sourceforge.net), was released publicly under the GPLv3 license in September 2008 and will be used in pilot deployments with small operators by the time this article goes to publication. This will probably be the first use of a free software basestation in a public cellular network, where both network operators and subscribers can download and read the full source code of the GSM protocol stack that connects their handsets to the rest of the world and where the operators will be free to modify the system to meet their specific needs. This article introduces the goals and evolution of the OpenBTS project.

History

To understand the relationship between OpenBTS and open source development, you must first understand the project's history. Initially, we planned to fund the development of low-cost cellular technology through the standard Silicon Valley startup process. Our process started in July 2007 with a meeting of the project founders and our first senior advisor, Glenn Edens, a former executive at both AT&T and Sun. We had breakfast along the San Francisco waterfront, laid out our story for Glenn, and asked how we should start our search for funding. After listening and considering, he said, "You don't need money at this point. You need a movement. Do this as a public open source project and develop a following, then the money will be much easier." Glenn also suggested using the Burning Man festival (http://burningman.com) as a technical proving ground for the project, for the exposure that it would produce, and because it would be a lot of fun. This one meeting set the direction of our activities for the next 18 months.

Despite the name, OpenBTS is not a natural candidate for a public open source project. For such a project to have mass appeal, it needs to be something that is easy to use and can provide significant functionality early in the development process. A GSM network stack lacks both of these features. First, the stack must be mostly complete in layers 1 and 2 (http://en.wikipedia.org/wiki/Osi_model) to do anything useful and these are the most complex layers of the system. Second, a developer needs about US$1,300 worth of radio equipment, a legal way to operate that equipment, some test phones, and a working knowledge of how GSM networks and handsets interact. Granted, US$1,300 is a remarkably small sum for a cellular basestation development kit, but this cost would be a significant barrier for many potential contributors. Given these circumstances, we knew that we would not make a public announcement of our project until layers 1 and 2 were complete. We assumed that open source participation would be limited to people who already owned the required hardware, the Universal Software Radio Peripheral (USRP, http://en.wikipedia.org/wiki/USRP), for other projects. We would have to write a lot of OpenBTS ourselves, see it work and then release it, and that would take several more months.

Even after that, we did not expect a lot of outside contributions.

In May 2008, just as the software was becoming complete enough for a public release, a former consulting client sued Kestrel and me personally over the OpenBTS project, claiming that the yet-unreleased source code was misappropriated from work I had done for him in the period 2005-2007. The work in question was a GSM stack for an IMSI-catcher, a device used to perform false-basestation and man-in-the-middle attacks on cellular networks. The client filed this suit having seen only a small fraction of the OpenBTS source code and knowing that I had written other GSM protocol implementations for other defense-sector clients prior to working for him. Among other things, he accused me of violating the trade secrets act, even though GSM is a publicly available specification and IMSI-catchers are the subject of patents in the UK and EU. This client eventually claimed the text of the source code itself as a trade secret, even though we did not use that client's source code in the OpenBTS project. It was our position that the case had no merit. We laid off our two junior employees to free up money for legal fees and continued moving forward as well as we could.

In September 2008, we ran a GSM test network at the Burning Man festival in Nevada. Our funds and equipment were limited, but the system actually worked. We immediately followed the test with a public announcement and the first public release of the source code. We made our initial announcement in the GNU Radio project's mailing list (http://www.gnuradio.org), a large forum for USRP developers.

News of the project spread quickly in the USRP and GNU Radio communities and within a few weeks John Gilmore, one of the founders of the Free Software Foundation (FSF, http://fsf.org) and a backer of GNU Radio, expressed interest in the project. John suggested that we transfer the copyrights for the OpenBTS source code to the FSF for release under the GPL license. We soon realized that the FSF's standard assignment contract preserved nearly all of our rights as developers through the grant-back of a blanket license. We formalized our FSF copyright assignment on October 24, 2008 and made a GPL release through the GNU Radio web site within a few days.

In early November 2008, after tens of thousands of visitors to the OpenBTS web site and hundreds of downloads under GPLv3, the ex-client and his lawyers discovered the public distribution of OpenBTS. They immediately petitioned the court for an injunction blocking distribution of the software, not against the FSF, but against the project founders. The court granted a more limited injunction requiring us to preserve the "names and internet addresses" of those who received technical information from the project. While that compromise seemed reasonable to the court in principle, the practical reality was confusing. We could only distribute the source code to individuals through e-mail, but the FSF, not being a party to the case, continued to run their anonymous access servers unaffected. We could continue to develop privately, but could not contribute to the public distribution of the project we had started. While this arrangement distorted the project's management, it was clear that the only way anyone could really shut down the OpenBTS project would be to sue the FSF directly.

Another effect of the injunction was greater public interest in the project. News of the injunction became a starting point for many online discussions of intellectual property law. As the public profile of the lawsuit rose, the client was becoming a well-known member of an industry that avoids publicity. The case was settled and the injunction lifted in August 2009, after several months of negotiations. While the settlement and negotiations were confidential, we have no doubt that our public release through the FSF had a tremendous influence on the early settlement of the case.

OpenBTS Today

OpenBTS continues to make GPLv3 releases through the FSF GNU Radio project. To date, the public and commercial releases of OpenBTS are identical in functionality, differing only in licensing. Our long-range plan is to follow a model similar to that of Digium (http://digium.com) and Asterisk (http://www.asterisk.org), with a free public release under the GPL and a commercial private release under a different license. We will also have a contributor's license agreement that protects the rights of contributing developers while transferring the copyrights for their contributions to Kestrel. We do not do this expecting other developers to write our system for free. A great majority of the current code has been written by Kestrel employees and paid contract developers and we do not expect that to change. There are, however, several practical reasons for maintaining a free public release.

Free as in Freedom

Free software and open source software are not the same thing, although OpenBTS's public release is both. One of the great lessons of our lawsuit experience is that widespread public distribution is an excellent way to protect your right to continue producing that work.

Public Image

One of the best reasons for a public release is to maintain the public visibility of the project. By making the source code available to developers and students for inspection and evaluation, we make the project real and exciting in the minds of a lot people. Since our long term goal is to provide cheap communications in poor and remote areas, it is counter to the spirit of the project to tell students and development workers that they must pay thousands of dollars to use the software in their experiments. From a business standpoint, there is little to be gained by denying groups that can not afford large licensing fees access to the software. The direct business advantage is that their work may lead to new applications and markets that we would not have been aware of otherwise. There is also a business advantage in creating a public image of the project as supportive of groups with socially important goals.

Testing and Documentation

While we do not get a lot of no-cost contributions from outside developers, we do get a lot of useful bug reports and debugging discussions. By following the typical sequence of questions from a new user and reading the material posted to the public wiki and discussion lists, we develop better documentation. The feedback we get through the public release is valuable in development and in the preparation of training materials and documentation.

Recruiting

As our project attracts more funding and we are required to hire staff, we expect the public work on the project to be a useful recruiting tool.

While we do not depend on outside developers to move the project forward, such developers do exist. Most would contribute more if they could be paid to do so and some would be glad to have full-time jobs on the project. From a hiring standpoint, these developers are known quantities. We have been coordinating with them and already have samples of their work. We have had good results hiring contract developers this way and hope to continue and expand that practice in the future.

Summary Comments

Simply put, an open source release may have saved our project. Now that the litigation threat is over, open source work continues to serve our commercial interests.

David Burgess is a Partner at Kestrel Signal Processing, Inc. and Co-Founder of The OpenBTS Project. David has nearly 15 years of experience in signal processing system development and scientific computing. Much of his work in recent years has been in the areas of signals intelligence, radiolocation, and navigation. He has also worked in electronic warfare, image processing, high-fidelity audio processing, and DSP system design. He holds an M.S. degree in computer science, and a B.S. degree in electrical engineering, both from the Georgia Institute of Technology.

"By His Genius Distant Lands Converse And Men Sail Unafraid Unto The Deep."
Epitaph on the memorial tablet of the grave of Canadian radio pioneer Reginald Fessenden

The Communications Research Centre Canada (CRC, http://www.crc.gc.ca), the federal government's primary laboratory for advanced telecommunications research and development (R&D), has been at the forefront of new developments in mobile digital broadcasting technologies since their inception in the late 1980s. During this time, digital replacement technologies have been standardized in an effort to rationalize spectrum use and enhance broadcasting applications with datacasting services and associated program information. Eureka DAB (http://en.wikipedia.org/wiki/Digital_Audio_Broadcasting) was the first all-digital mobile broadcasting technology to be conceived, developed and deployed. It became a widely adopted standard for digital radio in many countries around the world in the mid-1990s. DAB was officially launched in Canada in 2000.

This paper presents CRC's attempt to increase collaboration and innovation in the field of mobile broadcasting by developing and offering complete end-to-end Free, Libre and Open Source Software (F/LOSS) toolsets for the transmission and reception of DAB and FM/RDS (http://en.wikipedia.org/wiki/Radio_Data_System) applications and services.

Background

Radio has always been mobile. AM and FM receivers were introduced in cars in the 1930s, and later, with the emergence of transistor receivers, into portable devices. Radio was one of the first wireless telecommunication applications available to the masses.

Since then, there has been a growing interest in mobile broadcasting services. This has resulted in the proliferation of new broadcaster-led standards as well as standards developed by the mobile industry. These industries recognize that physical layer broadcasting (as opposed to pseudo-broadcasting like multicast on the Internet) is efficient, both spectrally and in terms of infrastructure, for delivery that targets large audiences. This is particularly attractive for media-rich content like radio and TV, which are very expensive to deliver through 2.5 or 3G infrastructures. To mobile network operators (MNOs), this translates into lower delivery costs and relief on their one-to-one networks, which remain dedicated to high-margin personal communication services.

For broadcasters, mobile broadcasting is a natural extension of their traditional mandate and expertise: cultural content, radio, public services like weather forecasts, traffic conditions, emergency information and so on. New opportunities like mobile TV and datacasting are also of great interest.

Open Eureka DAB Transmitter

Broadcast transmission equipment is expensive due to low sales volumes and relatively high technical complexity. This creates a high barrier to entry that limits the potential for innovation in broadcasting.

At CRC, we realized that application innovation for Eureka DAB was difficult when using typical commercial transmitter equipment. The earliest issues appeared with the service multiplexer. In DAB, multiple services are combined in real time to form a single bitstream or multiplex. All services available are announced through a specific sub-channel in the multiplex so that receivers know where to find a service and how to decode it.

When we created non-standard applications, there was no way to signal their presence in the multiplex. Application signalling was hard-coded in commercial multiplexers. This was the motivation to develop a more R&D and innovation friendly multiplexer. Over the years, this project evolved to become a full-blown F/LOSS DAB multiplexer, now known as CRC-DABMUX.

Another key component of the DAB transmission chain is the modulator. Again, these types of devices used to be quite expensive. However, with the emergence of software defined radio and accessible platforms, the implementation of a DAB software modulator has become straightforward. The USRP (http://en.wikipedia.org/wiki/Universal_Software_Radio_Peripheral) and the GNU Radio open source framework (http://gnuradio.org/redmine/wiki/gnuradio) were invaluable F/LOSS tools that permitted the integration of our CRC-DABMOD modulator in a very short time frame.

Together, these two components can now generate DAB-compliant signals at an unprecedented accessibility level. We pushed this one step further by putting most of our DAB transmission tools on an Ubuntu-based live CD which we distribute for free from our projects' Web sites. The CRC mmbTools Live CD (http://mmbtools.crc.ca/content/view/30/54/) can be launched without prior installation on the host PC to produce DAB signals in real time with a simple mouse click and a connected USRP.

We also used our software to provide DAB audio encoding and multiplexing functionality over the Web. These tools replicate typical functionality of physical devices and are known as Web appliances or WAPPs. For example, our WAPPs provide the functionality of DAB audio encoders and multiplexers. CRC-DABMUX is used here to produce multi-

plexed bitstreams according to parameters and content provided by the users. Since their launch, the WAPPs have been used steadily by various members of the industry and this process turned out to be an excellent mechanism to test our own software. Sometimes, users inform us about issues that they have. Often, we are able to fix the problems and upload updated software components to our WAPPs server in just a few hours.

Openmokast

The context for the democratization of mobile broadcasting at the receive end is different. In a time when independent developers and users are empowered to create mobile apps on iPhone and Android (http://android.com) platforms, broadcast apps development remains in the hands of a few players. More importantly, there are still no clear signals that digital broadcast chipsets will be integrated into mass market mobile devices.

The stagnation of mobile broadcasting technological advances could be explained by the innovative and competitive wireless communications ecosystem that is thriving today. Several kinds of new wireless communications technologies are emerging in the quest to reach mobile users, wherever they are, with maximum throughput. It appears as if mobile broadcasting is standing at a juncture between broadcasters and MNOs who are driven by distinct objectives and business models. Broadcasters are naturally inclined to pursue and extend their current free-to-air services which are monetized by public funding, licensing fees and advertising. MNOs, on the other hand, plan to deploy mobile broadcasting services to generate new revenue streams through cable-like subscriptions and pay-per-view models.

Openmokast (http://openmokast.org) was developed at CRC to address the lack of support of mobile broadcasting on emerging open handset platforms and to catalyze an application-driven ecosystem. As with Android, third party developers would be able to create new apps long before compatible receivers actually reach the market.

Openmokast is a complete software stack that provides a high-level application programming interface (API) for the control of a DAB receiver connected to a generic computer or smartphone. It demultiplexes the received bitstream and forwards selected raw sub-channel content to upper-layer decoding and rendering applications. It currently offers decoding libraries for standard applications like DAB and DAB+ audio, DMB (http://en.wikipedia.org/wiki/Digital_Multimedia_Broadcasting) video, Slideshow and Visual Radio. It is compatible with the Openmoko FreeRunner (http://wiki.openmoko.org) and the GNU/Linux operating system.

A physical extension was built to seamlessly integrate a USB-based receiver and its antenna at the back of the FreeRunner. The resulting prototype was the first open programmable handset to integrate the reception of live digital radio, video and data services with typical smartphone functions such as mobile telephony, wireless Internet and GPS positioning. An Android application was developed to showcase the usability of Openmokast in a mobile Wi-Fi broadcast hotspot configuration. This application is offered on the Android Market, as a GPL open source project, and is included on the CRC mmbTools live CD. Software-defined-radio demodulation of the DAB signal would represent another interesting approach to get mobile broadcasting reception on a mobile device without the need for specialized chipsets. However, today's smartphones still lack the low cost wideband front-ends plus the processing power to perform heavy lifting

tasks like signal demodulation.

Unencumbered Audio Codecs

Many of the tools provided on our CD are offered as F/LOSS to promote innovation and new developments in mobile broadcasting and DAB digital radio technology. However, some key components are missing as the free and unrestricted distribution of standardized DAB audio codecs is not possible. For these codecs, controlled distribution and payment of royalties are required.

Since audio is important in a broadcast radio system, we decided to integrate a royalty-free (RF) audio option with our tools. Interestingly, new developments in the area of RF or unencumbered codecs were being launched at the same time. For example, the IETF had just created a new working group to standardize an unencumbered Internet wideband audio codec. There is also a growing interest in the integration of the unencumbered Theora (http://www.theora.org) codec within the various Internet browsers to provide a free default option for the new HTML 5 video tag. The Open Video Alliance (http://openvideoalliance.org) promotes this solution.

We selected the CELT (http://en.wikipedia.org/wiki/CELT) audio codec for our implementation. CELT is a new full-band audio codec being developed and optimized for low-latency Internet applications. Its technical features make it attractive for broadcast applications. Like other free and unencumbered technologies promoted by the Xiph.Org Foundation, CELT requires no royalties and no special licensing. We designed and implemented a new transport protocol for CELT over DAB and adapted the C library provided by the Xiph project to our live CD and Openmokast client applications. This library now compiles on the various platforms that we use for our projects: Ubuntu, Android/G1 and Openmoko/FreeRunner. On Ubuntu, our CELT library will run as a plug-in for GStreamer (http://www.gstreamer.net), the F/LOSS media framework. Of course, none of the commercial DAB receivers can decode CELT radio as CELT is not a DAB standard.

Unencumbered codecs have the potential to become widely available on most types of Internet media platforms, which happen to be the same platforms broadcasters are targeting for mobile broadcasting reception. If this happens, broadcast radio systems may have to support such codecs in order to remain competitive with Internet radio on handhelds or other media devices.

Enabling Hybrid Radio

In many regions of the world, where the transition to digital radio appears to be stalled, FM radio could prevail for many years. In this context, how will radio evolve to provide all those multimedia enhancements envisioned for high capacity radio systems like Eureka DAB?

One option would consist of delivering radio over Internet-streaming networks instead of specialized physical layer broadcast infrastructures. While this appears to be a reasonable scenario for broadband-connected locations, live and uninterrupted audio delivery over mobile Internet today simply does not compare to FM. Consequently, a hybrid broadcast/broadband approach for radio could represent a more realistic evolutionary path toward its all-digital future. Current FM infrastructures would be leveraged to provide reliable and dependable audio-service components while optional multimedia elements,would be provided out-of-band through the mobile Internet and he Radio Data System (RDS). RDS is a communications protocol standard for embedding small amounts of digital information, including program informa-

tion, time and station identification, in conventional FM radio broadcasts.

In the logical continuity of our mmbTools and Openmokast work, we have integrated and developed new approaches and tools to foster the development of innovative hybrid FM/RDS/Internet radio applications and services. One important finding was that most of the required building blocks were already available as F/LOSS projects. A functional implementation was integrated in a few days using the same USRP and GNU Radio platforms that were used for our DAB projects.

While many new smartphones are already equipped with FM and RDS receivers, most of these platforms do not provide official APIs to third party and independent developers to control the broadcast receivers. For example, some new Android-based HTC (http://htc.com) products have FM and RDS receivers inside them, although no Android API provides access to this functionality and an official Android representative has mentioned that there are no plans to include an API. It seems that handset manufacturers will be the only ones able to provide FM tuning and RDS decoding apps.

We found some unofficial mechanisms to access FM and RDS functionality on some Windows Mobile handsets. Special libraries were developed by hacker communities who reverse-engineered the FM APIs. Thanks to these developments, we produced a portable RDS decoding software library that we will use to demonstrate hybrid radio application prototypes on commercially available smartphones.

Conclusion

The CRC mobile broadcasting F/LOSS projects have generated a lot of interest among the DAB community. Several dozen DAB industry players, including broadcasters, universities, manufacturers and application developers, have downloaded the CRC mmbTools live CD. Our online Wapps are visited steadily. An independent online community of enthusiastic users (http://opendigitalbroadcasting.org) is actively building a knowledge base around our tools. Our compact end-to-end system triggers interesting discussions about the future of mobile broadcasting whenever we introduce it at international events and trade shows (http://www.youtube.com/crcmmb).

While the business models for broadcasting are blurred by new content distribution options and by new media applications, it is difficult to estimate the real impact of the CRC mobile broadcasting F/LOSS projects. We think that by democratizing mobile broadcasting technologies, we increase their chance to remain competitive and succeed in the future.

François Lefebvre joined the Communications Research Centre, Canada, in 1999 to lead its Mobile Multimedia Broadcasting team. Since then, he has contributed to numerous national and international standardization efforts and R&D projects. His recent work has focused on creating and developing open software building blocks for next-generation mobile broadcasting networks, devices and applications. With his team, he launched the CRC mmbTools and Openmokast open source software projects. He writes about the future of broadcasting on his blog Broadcasting 2.0 (http://www.broadcasting20.org). Mr. Lefebvre graduated from Laval University in Electrical Engineering where he also completed his M.A.Sc. in 1989. He pursued his carreer in Europe, mainly in Germany, where he worked for ten years as engineer in R&D laboratories and as freelance supervisor of software developments on emerging multimedia and Internet platforms.

"The engineer's first problem in any design situation is to discover what the problem really is."

George C. Beakley

This article briefly describes OSSIE (http://ossie.wireless.vt.edu), a university-based open source Software Defined Radio (SDR) project based on the U.S. Department of Defense's Software Communications Architecture (SCA, http://en.wikipedia.org/wiki/Software_Communications_Architecture). The OSSIE software has proven useful for rapid prototyping by industry as well as for published research and education of hundreds of graduate and undergraduate students and short course participants. In addition to examples of OSSIE's successes, the project's challenges and approaches to mitigating and overcoming them are described.

Introduction

SDR is a flexible approach to radio design that allows a radio to support new communications standards by changing the radio's software. SDR is becoming increasingly prevalent in commercial as well as military arenas, with examples that include Vanu's AnyWave cellular basestation (http://www.vanu.com/solutions/anywave.html), Apple's iPhone, and the U.S. Department of Defense's Joint Tactical Radio System (JTRS, http://jpeojtrs.mil).

The OSSIE project, based at Virginia Tech, provides open source SDR software based on the SCA. The software, collectively known as OSSIE, includes a core framework or infrastructure software, as well as rapid prototyping tools and building blocks for developing SDR applications or waveforms. The SCA is an open SDR architecture associated with JTRS and other U.S. government programs, and has also been used to implement commercial communications standards.

The SCA's military and commercial relevance has given rise to a community of SDR developers who can benefit from SCA-based open source software.

History and Role of Open Source in SCA-based Software Defined Radio

Prior to OSSIE's initial development by IC postdoctoral fellow Max Robert and a team of Jeff Reed's students at Virginia Tech in 2003, the Communications Research Centre Canada (CRC) developed SCARI open (http://crc.gc.ca/en/html/crc/home/research/satcom/rars/sdr/products/scari_open/scari_open), a Java-based open source reference implementation of the SCA .

The OSSIE core framework was developed in the C++ programming language to facilitate portability to embedded platforms. OSSIE also includes readily mastered tools, processing blocks, device interface code, and documentation that enable basic tasks of SDR development. OSSIE fills a niche as a free resource for SCA-based SDR research, education, development, and rapid prototyping. OSSIE implements a subset of the SCA sufficient to build working waveforms, which can be ported to run with commercial SCA frameworks. OSSIE is licensed under the GNU General Public License (GPL) and Lesser General Public License (LGPL) and can be downloaded at no charge. In contrast, full-featured commercial SCA frameworks and development tools, while providing close fidelity to the SCA and optimization for operational use under demanding conditions, are expensive and can run into the tens of thousands of dollars per seat or per copy.

The high costs associated with commercial SCA software were the initial motivation for developing OSSIE as a tool for

research and education in resource-limited university settings. At the same time, OSSIE's low cost has proven advantageous in industry. OSSIE is used for SDR rapid development and proof of concept implementation, as well as to introduce new users, developers, or customers to SCA concepts. OSSIE's licensing is attractive because it allows users to customize the software for their own applications.

Successes of the Project

The open source approach has benefited SDR education, research, and development at Virginia Tech and elsewhere. Accomplishments of the OSSIE project include:

- widely distributed open source SDR software
- over 20,000 estimated total downloads of source code and ready-to-run live DVD and VMware images
- confirmed use by students and engineers at over 20 universities, companies, nonprofit research centers, and government laboratories
- eight free laboratory/tutorial exercises, suitable for classroom use or self-paced study, developed with the Naval Postgraduate School
- additional documentation via an 80+ page user and installation guide
- over 20 graduate and undergraduate students supported through work on OSSIE related projects
- more than 10 graduate theses and projects at Virginia Tech, the Naval Postgraduate School, and elsewhere that have used the software
- more than 10 short courses and tutorials, serving over 200 participants
- three peer-reviewed journal articles, four online articles, and more than twenty conference papers resulting from the project or using the software
- over US$3,000,000 in related sponsored and gift-supported research

OSSIE is used by projects in industry, government, and university settings. Aalborg University (http://cdsr.net.dynamicweb.dk/Projekter/SDR_implementering_af_Inmarsat_BGAN_transceiver_baseret_p%C3%A5_SCA_standard.aspx) and Gate House (http://www.teknologisk.dk/_root/media/34853_5_gh_bsdr_wireless_vitae_09.pdf) each reported its use. OSSIE is being used in a proof of concept of the Government Reference Architecture (GRA), a U.S. government architecture for above 2 GHz communications terminals. Other studies have focused on performance of the OSSIE software (http://eprints.nuim.ie/1415/1/PALOMOA.pdf), used the software to explore effects of waveform granularity (http://rta.nato.int/pubs/rdp.asp?RDP=RTO-MP-IST-083), and demonstrated and documented porting and interoperability of waveforms between OSSIE and CRC's commercial SCA framework and toolset (http://data.memberclicks.com/site/sdf/sdr09-02-Singh.pdf). Also noteworthy is OSSIE's inclusion in the OpenCPI initiative (http://opencpi.org).

Challenges

Potential challenges to a university-based open source project such as OSSIE include:

- managing competing university, developer, and stakeholder priorities
- maintaining open communications with and among stakeholders

- maintaining continuity of student funding throughout each student's degree program

- maintaining project continuity in an environment where high turnover among project personnel is desirable as part of the university's primary, educational mission

- development, testing, maintenance, and configuration management of the software itself

Addressing the Challenges – Lessons Learned

As mentioned earlier, Virginia Tech has benefited from the project in terms of students supported, degrees completed, and support for related research including use of the software in other radio design efforts. Like many universities, Virginia Tech has a center that manages licensing and commercialization of technologies developed by its students, faculty, and staff, a mission that could be seen as in tension with the open source approach. Over the course of the project, we have kept Virginia Tech Intellectual Properties (VTIP) informed of the software's status. VTIP has been very supportive of the OSSIE project and recognizes the benefits of the open source model for the university and the broader community, while also providing the option to negotiate alternative licensing for those desiring it.

Developers on the OSSIE project are students, whose main academic priority is completion of their degrees, including coursework and research that are not directly related to development and enhancement of the OSSIE software. OSSIE's growing capability and improved ease of use over the past few years have made it more useful for thesis research, increasing students' motivation to further improve the software.

Students are also motivated by the opportunity to work on software that is unusual and widely used within the SDR community. This enthusiasm on the part of the students and Dr. Robert sustained the project through its early period in which there was no external support.

The project has stakeholders that include university, industry, government users, funding companies, and government agencies. Interest in and preferences for particular capabilities or characteristics of the software varies depending on the stakeholder's immediate and longer term goals, so it may not be possible to support or accommodate all stakeholders equally. Tension exists between the desire for rapidly enhancing the software and maintaining stability and backward compatibility, supporting new operating system releases, supporting a wider variety of hardware platforms, and supporting SDR research and general wireless communication research, all within project resource constraints. As the project progresses, it is increasingly desirable to maintain open communication with and among stakeholders. This is partially achieved through use of a mailing list and a wiki that includes planned project milestones, bug fixes, and enhancements. At the suggestion of some of our stakeholders, we plan to initiate a users' group to encourage and formalize this interaction, possibly including periodic teleconferences.

Continuity in every sense is important to the project. Although challenging, effective solutions have been found. New and related research projects are required to support students through two or more years of graduate school. To date, this has been possible due to multi-year projects and new projects that build on past successes or explore new application areas.

Maintaining project funding conducive to both software development and presentable, publishable research is a long-term challenge. Continuity of project staff is maintained by research faculty, by students who stay or return for additional degrees or postdoctoral fellowships, involvement of undergraduate researchers over multiple years, and by frequent interaction of new students with more experienced students. Students and postdoctoral fellows who come from other institutions where they have used the software provide another potential source of continuity. The software, as well as documentation of installation procedures, must also keep pace with updates to software dependencies and the operating system to remain useful over time. This requires extra effort on the part of student developers. User reports on the project mailing list or wiki help identify incompatibilities and often users identify solutions as well.

Additional challenges relate to software development itself. OSSIE development began and continues in an electrical and computer engineering department. As the project has progressed, we have involved faculty and student developers from our computer science department as well as computer engineers and communications engineers. Some work, such as development of rapid development tools and user interfaces, does not require specialization in communications theory or computing hardware and is often more easily and better developed by computer science students. Electrical and computer engineers can now concentrate on the core framework, development and optimization of digital signal processing components and waveform applications, and porting the software to work with new digital and radio frequency hardware. Configuration management is achieved using the subversion (http://subversion.tigris.org) revision control system, connected to a Trac (http://trac.edgewall.org) wiki, while bug reporting occurs via a mailing list and the wiki.

OSSIE benefits from use of other open source software. OSSIE runs in Linux and has been ported to embedded platforms such as an OMAP starter kit (OSK). We have avoided reimplementing major functionality where possible; however, it is also desirable to minimize the number of dependencies to simplify maintenance. The current version of OSSIE uses omniORB (http://omniorb.sourceforge.net), an open source CORBA implementation. OSSIE was recently updated to use the GNU Radio (http://www.gnuradio.org) interface to the Ettus Research (http://ettus.com) Universal Software Radio Peripheral (USRP), a popular low cost radio frequency front end. The OSSIE rapid prototyping tools leverage the Eclipse.org open source integrated development environment and use Jython.org to interface with legacy Python.org code. It is possible that future versions of the OSSIE Eclipse Feature will be implemented entirely in Java to simplify maintenance.

Conclusion

The OSSIE project provides an open source resource for SDR education, research, and rapid prototyping. Development and use of OSSIE has led to multiple publications and presentations and several graduate and undergraduate students have been supported on related research projects. The software has been downloaded over 20,000 times and each of the accompanying labs, suited for university or self-paced study, are downloaded at a rate of about 1,000 per year, while well over 100 graduate and undergraduate students and 200 professionals have attended semester, quarter, and short courses that feature hands-on experience with the software.

The project faces interesting challenges and opportunities due to its specialized but heterogeneous user base and reliance on student developers supported by funded research projects and grants. Use of open source tools for configuration management, bug tracking, and communication has proven valuable to the project.

OSSIE is supported in part by the National Science Foundation under Grant No. 0520418. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Recommended Resource

Why are the latest smart phones using SDR?
http://groups.sdrforum.org/p/bl/et/blogid=20&blogaid=20

Carl B. Dietrich is a Research Assistant Professor in the Bradley Department of Electrical and Computer Engineering at Virginia Tech, where he completed Ph.D. and M.S. degrees after graduating from Texas A&M University. He worked with the Defense Information Systems Agency, Arlington, Virginia and Bell Northern Research, Richardson, Texas and conducted research on adaptive and diversity antenna systems and radio wave propagation. His current work in software defined radio (SDR) includes leading projects related to the OSSIE open source effort. He chairs the Wireless Innovation Forum Educational Work Group, is a member of IEEE, ASEE, and Eta Kappa Nu, and is a Professional Engineer in Virginia.

Jeffrey H. Reed is the Willis G. Worcester Professor in the Bradley Department of Electrical and Computer Engineering and director of Wireless @ Virginia Tech. His area of expertise is in software radios, cognitive radios, wireless networks, and communications signal processing. He is an IEEE Fellow and the author of Software Radio: A Modern Approach to Radio Design (Prentice Hall, 2002) and An Introduction to Ultra Wideband Communication Systems (Prentice Hall, 2005).

Stephen H. Edwards, Associate Professor of Computer Science at Virginia Tech, has interests in component-based software, automated software testing, and educational uses of computers. As the PI on an NSF phase II CCLI project, he developed Web-CAT, the most widely used open-source automated grading tool for computer programming assignments, with nearly 10,000 users at over 30 institutions worldwide. He is also a member of his department's undergraduate program committee, and chair of the subcommittee on curriculum and courses. Dr. Edwards has a background in component-based systems and has collaborated on software-defined radio research since 2007.

Frank E. Kragh is an Assistant Professor of Electrical and Computer Engineering at the Naval Postgraduate School in Monterey, California. Dr. Kragh received his B.S. from Caltech in 1986, his M.S. from the University of Central Florida in 1990, and his Ph.D. from the Naval Postgraduate School in 1997. His chief research and teaching interests are digital communications, software defined radio, multiple-input multiple-out systems, and military communications systems.

"... the cloud will soon become a disruptive force in the mobile world, eventually becoming the dominant way in which mobile applications operate."

Sarah Perez
http://tinyurl.com/n5rsj7

Cloud computing is gaining acceptance as an efficient and cost-effective architecture to deploy many types of systems. More recently, mobile cloud computing has entered the scene, as an important means to deliver mobile apps and data. This article discusses trends that are driving the adoption of the mobile cloud, important components of mobile cloud infrastructure, and the role of open source.

Overview

Cloud computing provides a flexible, convenient and affordable way to remotely distribute processing and data. In the context of mobile devices, the ability to offload processing and data storage is highly desirable because mobile devices have limited processing, memory, network bandwidth and battery life. A mobile cloud infrastructure that remotely performs computing, manages data, and can back up wireless devices in the event of loss, power failure or network interruption, makes for a compelling use case.

Computing requirements of a mobile cloud differ significantly from those of a regular computer cloud. The biggest difference is found in the diversity of supported mobile devices. There are four billion mobile phones, representing a vast array of mobile operating systems and hardware platforms. While mobile standards exist on large numbers of phones, there are significant variations that make it practically impossible to provide rich, usable mobile apps that support more than a small fraction of devices.

This device fragmentation is poised to become more difficult to address before it improves as several new types of mobile devices are entering the market. These include e-book readers, tablet PCs, digital picture frames, wifi cameras, and wireless cars and appliances. These devices have increasing amounts of storage and connectivity, and are good candidates to connect to a mobile cloud. It is easy to imagine that people will want to access a wide variety of data and content on different mobile devices.

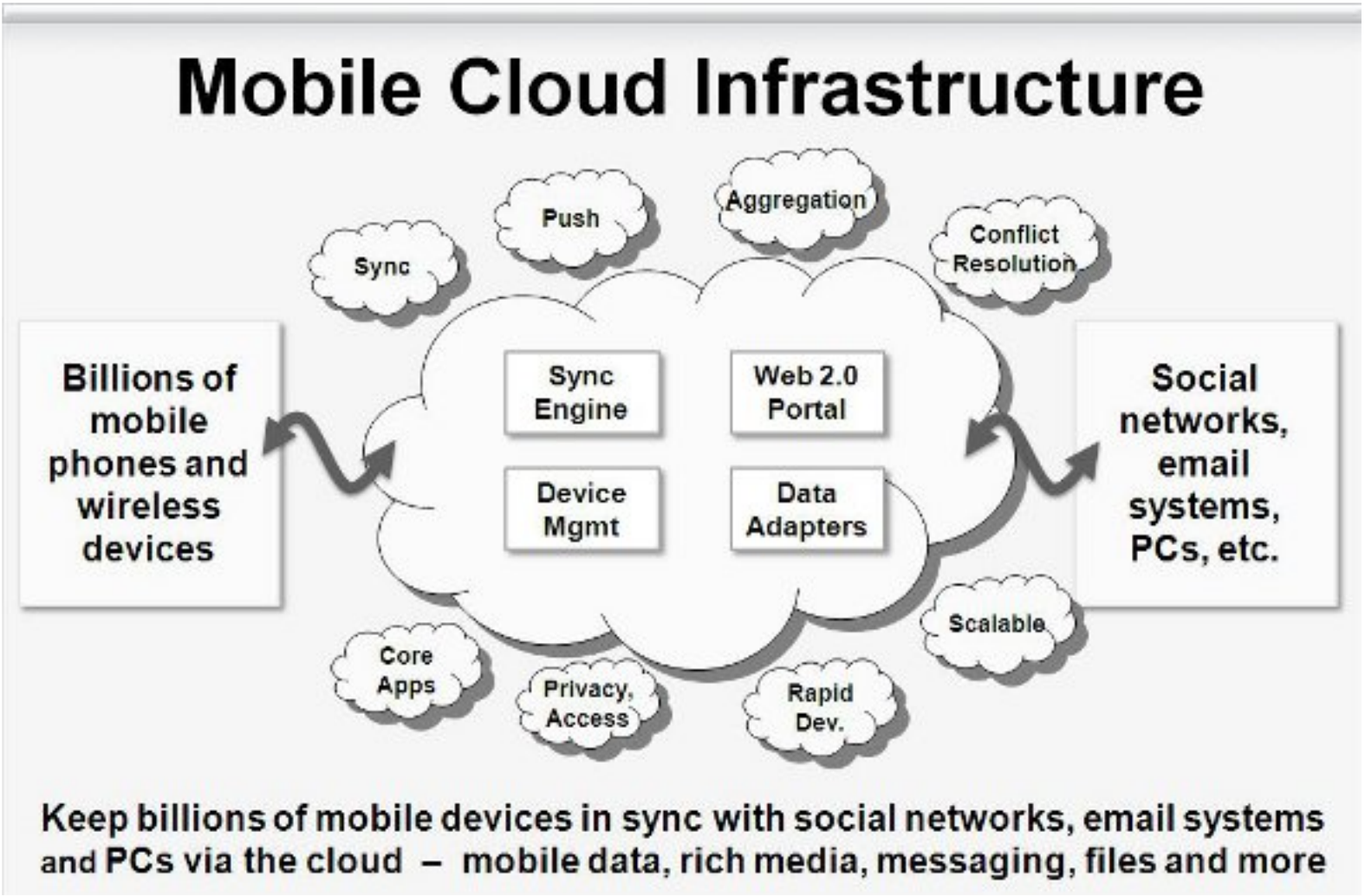
For example, people may want to access Gmail addresses on a wifi camera or e-book reader so they can email photos or an article. They may want to view a work or family calendar on a home refrigerator or vehicle screen, or they may want to post a photo from a camera phone to a corporate intranet or social network. A mobile cloud should provide the infrastructure to support these possibilities and more.

Mobile Cloud Components

There are several essential elements of mobile cloud infrastructure that make it distinct from a regular computer cloud, as depicted in Figure 1.

The primary purpose of a mobile cloud is to make it easy to sync mobile phones and devices with systems such as social networks, email systems, computers, and virtually any data store. The goal is to perform processing and to manage data in the cloud, to offload these functions from mobile devices. What follows is a description of the significant components and capabilities of the mobile cloud.

Figure 1: Mobile Cloud Infrastructure Elements



Sync engine: a mobile cloud should be able to sync a wide variety of data and content, between any source and device. Some people may question whether syncing is still needed in an age of broadband wireless networks. The answer is "Yes". Even with fast 4G networks, there will still be pockets of non-networked areas and times when devices are offline, and people will still want access to their data and content. Furthermore, for a good user experience, it is necessary for many apps to access local device data. For example, users do not want to wait while video buffers.

Web 2.0 portal: a second major element is a web 2.0 interface for user data and content. This provides a means to view, manage, edit and filter data and content that flows between devices and data sources. This applies to mobile data such as address books, calendars and email, as well as rich media such as photos and video. Consider the proliferation of contacts in systems such as email, social networks, VoIP and more. People may have hundreds or thousands of contacts in multiple places, yet they typically only want a small fraction of these on their phone. A web 2.0 portal should make it easy to set up groups of users or to indicate which contacts to include from which sources.

Another example is posting photos from a phone to multiple destinations, such as social networks, photo sharing sites or personal computers. There needs to be an easy way to allow people to specify how rich media should be managed. A web 2.0 portal that provides an intuitive

desktop-like user interface in a web browser, to access and manage mobile cloud data, is important.

Device management: small, portable and relatively inexpensive mobile devices are dropped, broken, lost, stolen and exchanged with greater frequency than other computing devices. This not only makes it more important to back them up, in case their data becomes lost, but it makes them more costly to support. An important aspect of a mobile cloud platform is the ability to remotely manage devices over the air, in terms of provisioning devices, performing diagnostics, updating software and settings, and remotely locking devices and erasing data for security reasons. These functions are typically found today with higher end smartphones such as BlackBerries and iPhones, but they are increasingly becoming expected with other types of portable devices.

Data adapters: to sync a wide range of data and content, there needs to be an easy and flexible way for mobile cloud apps to access diverse systems such as social networks, email systems, databases, customer resource management (CRM), and enterprise resource planning (ERP) applications and servers. Without this ability, it could take too long to perform even simple tasks. An important component of mobile cloud infrastructure is data adapters that provide the rapid ability to sync with common systems and to supplement this with the ability to interface with custom systems.

Beyond these core mobile cloud infrastructure components, there are several additional capabilities that are important in a mobile cloud platform, as illustrated by the smaller clouds in Figure 1.

Push notifications: when data or content is changed in one place, for example on a mobile device or online, it is important that the change automatically propagate everywhere it should, without the user initiating an update. This is the role of push notifications, which can be performed using a variety of methods, including TCP/IP, SMS and polling. Some networks and devices are only capable of supporting certain forms of push notification, so the form of push notification used needs to conform to the profile of the involved networks and devices.

Aggregation: many mobile cloud apps require aggregation, such as gathering data from multiple email systems, social networks and other systems. The mobile cloud platform should be able to intelligently source data from a variety of systems. Considerations include how often remote systems are accessed and which data is cached on the server versus stored locally or pointed to remotely.

Conflict resolution: when working with data from multiple sources, one of the most common yet perplexing challenges is reconciling differences among like data. A simple example stems from having someone's name in a mobile address book, while having a different version of their name in an email system or social network. When aggregating this information, it is easy to end up with multiple entries representing the same person.

A critical capability is detecting 'twins' by comparing attributes such as email addresses, phone numbers and other data, to determine whether these are the same person. There need to be configurable rules for determining which data should win a conflict. This may be viewed as a fairly arcane aspect of mobile cloud services, but maintaining the integrity of people's data is paramount, and a robust conflict resolution system is a must.

Core apps: many mobile cloud apps involve a common set of functions, such as syncing contact data, calendars, email,

files and photos. It is important for mobile cloud infrastructure to provide common capabilities so these functions can be performed without reinventing the wheel.

Privacy and access: as user data is stored in the cloud, it is critical that data is highly secure and backed up. At the same time, there needs to be a simple way for users to specify which data to share with other people and systems.

Rapid development: there needs to be a way to rapidly build mobile cloud apps that work on a variety of mobile phones and devices. Until recently, developers either needed to build native apps for each mobile platform, which was extremely expensive and labour-intensive, or build web apps, which worked on many phones but were unattractive and clunky. There are some new initiatives that purport to provide developers with the best of both worlds: the creation of one version of a mobile app that can be widely deployed, while exhibiting many of the characteristics of native apps such as a rich user interface, local data storage, and integration with other apps on the device. Examples include the newly announced Wholesale Applications Community (WAC, http://wholesaleappcommunity.com) initiative, technology from rhomobile (http://www.rhomobile.com), and a newly announced open source mobile web 2.0 framework from Funambol.

Scalability: an important aspect of mobile cloud infrastructure is the ability to support large numbers of users and, in some cases, millions or tens of millions of devices. This can be accomplished by using industry standard application servers and infrastructure, and approaches for load balancing and fault-tolerance.

The Role of Open Source

It would be seriously remiss to discuss the mobile cloud without mentioning the role of open source.

Two years ago, if you mentioned open source and mobile in the same sentence, most people thought you were referring to obscure projects for developers. Much has changed. Google's Android open source mobile operating system has been embraced by many top device manufacturers and is an attractive platform for developers due to its openness. On the server side, Funambol's open source mobile cloud sync server has been adopted by leading mobile companies (http://funambol.com/news/pressrelease_2009.10.28.php) and has been downloaded three million times. These projects illustrate that open source is transforming mobile and is attractive to both companies and developers.

In the case of Funambol (http://www.funambol.com), an open source mobile cloud platform, open source permeates all aspects of the solution. The vast majority of Funambol code is open source and there is a large, worldwide community of more than 50,000 developers who have contributed to the Funambol project. Their contributions generally fall into one of the following areas:

1. Device compatibility: the largest area of contribution pertains to enabling Funambol to support more mobile devices than comparable software. This stems from the community's development and testing of Funambol software on a large range of mobile devices and sharing this work.

2. Connectors: there are several dozen Funambol community projects for connecting the Funambol mobile cloud server with various email and groupware systems, social networks, CRM systems and other applications.

3. Server enhancements: some Funambol community developers have extended the core Funambol mobile cloud sync server to provide advanced functionality such as server-to-server synchronization.

4. Many other open source contributions: Funambol has received numerous additional contributions, in the form of software and documentation that has been translated into different languages, significant feedback about the software performance, and much more. Funambol's mobile cloud platform has reached a critical mass of functionality and usage due to open source.

Conclusion

Several trends are favouring the increased adoption of mobile cloud infrastructure, not the least being the growing diversity of mobile phones and devices. There are several major components of a mobile cloud platform, including a sync engine, web 2.0 portal, device management and data adapters. Important capabilities of mobile cloud infrastructure include sync as well as push notifications, aggregation, conflict management, core app functionality, privacy and access controls, rapid development and scalability. Open source has played a major role in the evolution of mobile cloud infrastructure by significantly increasing device compatibility, connectors with other systems and many other aspects.

Hal Steger is Vice President of Marketing at Funambol, inc., the leader provider of open source mobile cloud platforms for billions of devices. Hal has over twenty years of experience in the high tech industry. He holds an MBA (M.S.I.A.) from Carnegie-Mellon University and a B.S. from the University of Michigan with a concentration in computer science.

THE STATE OF FREE SOFTWARE IN MOBILE DEVICES

"We expect that, by 2012, around 62 percent of the whole smartphone market will be open source with Symbian, Android and other Linux flavours."

Roberta Cozza, Gartner analyst

I started using GNU/Linux and Free Software (http://gnu.org/philosophy/free-sw.html) in 1992. In those days, while everything I needed for a working computer was generally available in software freedom, there were many components and applications that simply did not exist. For highly technical users who did not need many peripherals, the Free Software community had reached a state of complete software freedom. Yet, in 1992, everyone agreed there was still much work to be done. Even today, we still strive for a desktop and server operating system, with all relevant applications, that grants complete software freedom.

Mobile telephone systems are not all that different from 1992-era GNU/Linux systems. The basics are currently available as Free, Libre, and Open Source Software (F/LOSS). If you need only the bare minimum of functionality, you can, by picking the right phone hardware, run an almost completely F/LOSS operating system and application set. Yet, we have so far to go. This article discusses the current penetration of F/LOSS in mobile devices and offers a path forward for free software advocates.

A Brief History

The mobile telephone market has never functioned like the traditional computer market.

Historically, the mobile user made arrangements with some network carrier through a long-term contract. That carrier "gave" the user a phone or discounted it as a loss-leader). Under that system, few people take their phone hardware choice all that seriously. Perhaps users pay a bit more for a slightly better phone, but generally they nearly always pick among the limited choices provided by the given carrier.

Research in Motion (http://rim.com) was the first to provide corporate-slave-oriented email-enabled devices. Today, most people using a "smart phone" are using one given to them by their employer to chain them to their office email 24/7.

Apple, excellent at manipulating users into paying more for a product merely because it is shiny, also convinced everyone that now a phone should be paid for separately, and contracts should go even longer. The "race to mediocrity" of the phone market has ended. Phones need real features to stand out. Phones, in fact, aren't phones anymore. They are small mobile computers that can also make phone calls.

Free Software in Nokia Devices

If these small computers had been introduced in 1992, I suppose I'd be left writing the "Mobile GNU Manifesto", calling for developers to start from scratch writing operating systems for these new computers, so that all users could have software freedom.

Fortunately, we have been given a head start. Unlike in 1992, not every company in the market today is completely against releasing Free Software. Specifically, two companies have seen some value in releasing (some parts of) phone operating systems as Free Software: Nokia and Google. However, the two companies have done this for radically different reasons.

Nokia likely benefited greatly from the traditional carrier system. Most of their phones were provided relatively cheaply with contracts. Their interest in software freedom was limited and perhaps even non-existent. Nokia sold new hardware every time a phone contract was renewed, and the carrier paid the difference between the loss-leader price and Nokia's wholesale cost. The software on the devices was simple and mostly internally developed. What incentive did Nokia have to release software in software freedom?

In parallel, Nokia had chased another market: the tablet PC. Not big enough to be a real computer, but too large to be a phone, these devices have been an idea looking for a user base. GNU/Linux remains the ideal system for these devices, and Nokia saw that. Nokia built the Debian-ish Maemo (http://maemo.org) system as a tablet system, with no phone. However, the market for these devices has been minute.

Few understand why Nokia took so long to use Maemo as a platform for a tablet-like telephone. But, a few months ago, they finally released one. The N900 (http://maemo.nokia.com/n900) is among only a few available phones that make any strides toward a fully free software phone platform. Yet, the list of proprietary components required for operation (http://wiki.maemo.org/Why_the_closed_packages) remains quite long.

The common joke is that you can't even charge the battery on your N900 without proprietary software.

While there are surely people inside Nokia who want more software freedom on their devices, Nokia is fundamentally a hardware company experimenting with software freedom in hopes that it will bolster hardware sales. Convincing Nokia to shorten that proprietary list will prove difficult, and the community based effort to replace that long list with Free Software (called Mer, http://wiki.maemo.org/Mer) faces many challenges. These challenges increased with the recent Maemo merger with Moblin to form MeeGo (http://meego.com).

Free Software in Google Devices

Fortunately, hardware companies are not the only entity interested in phone operating systems. Google, ever-focused on routing human eyes to its controlled advertising, realizes that even more eyes will be on mobile computing platforms in the future. With this goal in mind, Google released the Android/Linux system (http://www.android.com), now available on a variety of phones in varying degrees of software freedom.

Google's motives are completely different than Nokia's. Technically, Google has no hardware to sell. They do have a set of proprietary applications that yield the "Google online experience", and deliver Google's advertising. From Google's point of view, an easy-to-adopt, licensing-unencumbered platform will broaden their market.

Android/Linux is a nearly fully non-copylefted (http://en.wikipedia.org/wiki/Copyleft) phone operating system platform where Linux is the only GPL licensed component essential to Android's operation.

Google wants to see Android adopted broadly in both Free Software and mixed Free/proprietary deployments. Google's goals do not match that of the software freedom community, so in some cases, a given Android/Linux device will give the user more software freedom than the N900, but in many cases it will give much less.

The HTC Dream (http://www.htc.com/www/product/dream/overview.html) is the only Android/Linux device I know of where a careful examination of the necessary proprietary components have been analyzed. Obviously, the Google experience applications are proprietary. There also are about 20 hardware interface libraries that do not have source code available in a public repository (http://trac.osuosl.org/trac/replicant/wiki/HTCDreamProprietaryDrivers). However, when lined up against the N900 with Maemo, Android on the HTC Dream can be used as an operational mobile telephone and 3G Internet device using only three proprietary components: a proprietary GSM (http://en.wikipedia.org/wiki/Gsm) firmware, proprietary Wifi firmware, and two audio interface libraries. Further proprietary components are needed if you want a working accelerometer, camera, and video codecs as their hardware interface libraries are all proprietary.

Based on this analysis, it appears that the HTC Dream currently gives the most software freedom based on the Android/Linux platform. It is unlikely that Google wants anything besides their applications to be proprietary. While Google has been unresponsive when asked why these hardware interface libraries are proprietary, it is likely that HTC, the hardware maker with whom Google contracted, insisted that these components remain proprietary. While no detailed analysis of the Nexus One (http://google.com/phone) is available yet, it's likely similar to the HTC Dream.

Other Android/Linux devices are now available, such as those from Motorola. There appears to have been no detailed analysis done yet on the relative proprietary/freeness ratio of these Android deployments. One can surmise that since these devices are from traditionally proprietary hardware makers, it is unlikely that these platforms are freer than those available from Google, whose maximal interest in a freely available operating system is clear and in contrast to the traditional desires of hardware makers.

Mobile and the Free Software Community

Whether the software is from a hardware maker trying something new to sell their hardware, or an advertising salesman who wants some influence over an operating system choice to improve ad delivery, the software freedom community cannot assume that the stewards of these codebases have the interests of the user community at heart. Indeed, the interests between these disparate groups will only occasionally be aligned. Community-oriented forks, as has begun in the Maemo community with Mer, must also begin in the Android/Linux space too. We are slowly trying with the Replicant project (http://trac.osuosl.org/trac/replicant/wiki), founded by myself and my colleague Aaron Williamson.

A healthy community-oriented phone operating system project will ultimately be an essential component to software freedom on these devices. Consider the fate of the Mer project now that Nokia has announced the merger of Maemo with Moblin. Mer does seek to cherry-pick from various small device systems, but its focus was to create a freer Maemo that worked on more devices. Mer now must choose between following the Maemo in the merge with Moblin, or becoming a true fork.

Ideally, software freedom is a community-led effort, but there may not be enough community interest, time and commitment to shepherd a fork while Intel and Nokia push forward on a corporate-controlled codebase. Further, Moblin will likely push the MeeGo project toward more of a tablet-PC operating system than a smart phone.

A community-oriented Android/Linux fork has more hope. Google has little to lose by encouraging and even assisting with such forks as its goals include wider adoption of platforms that allow deployment of Google's proprietary applications. Operating system software-freedom-motivated efforts will be met with more support from Google than from Nokia and/or Intel.

Any operating system, even a mobile device one, needs many applications to be useful. Google experience applications for Android/Linux are merely the tip of the iceburg in the plethora of proprietary applications that will be available for MeeGo and Android/Linux platforms. For F/LOSS developers who don't have a talent for low-level device libraries and operating system software, these applications represent a straightforward contribution towards mobile software freedom. On this point, we can take a page from Free Software history. From the early 1990s onward, fully free GNU/Linux systems succeeded as viable desktop and server systems because disparate groups of developers focused simultaneously on both operating systems and application software. We need that simultaneous diversity of improvement to actually compete with the fully proprietary alternatives, and to ensure that the "mostly F/LOSS" systems of today are not the "barely F/LOSS" systems of tomorrow.

Other Systems To Consider

Careful readers have likely noticed that I have ignored Nokia's other release, the Symbian codebase (http://www.symbian.org). Every time I write or speak about the issues of software freedom in mobile devices, I'm chastised for leaving it out of the story. My answer is always simple: when a F/LOSS version of Symbian can be compiled from source code, using a F/LOSS compiler or software development kit (SDK, http://en.wikipedia.org/wiki/Sdk), and that binary can be installed onto an actual working mobile phone device, then I will believe that the Symbian source release has value beyond historical interest. We have to get honest as a community about the future of Symbian: it's a ten-year-old proprietary codebase designed for devices of that era that doesn't bootstrap with any compilers our community uses regularly. Unless there's a radical change to these facts, the code belongs in a museum, not running on my phone.

I must also mention the FreeRunner device and OpenMoko (http://wiki.openmoko.org/wiki/Main_Page). This was a noble experiment: a freely specified hardware platform running 100% F/LOSS. I used an OpenMoko FreeRunner myself, hoping that it would be the mobile phone our community could rally around. I do think the device and its software stack has a future as an experimental, hobbyist device. But, just as GNU/Linux needed to focus on x86 hardware to succeed, so must software freedom efforts in mobile systems focus on mass-market, widely used, and widely available hardware.

Jailbreaking and the Self-Installed System

When we decided to move our office as close to a software freedom phone platform as we could, we picked Android/Linux and the HTC Dream. We carefully considered the idea of permission to run one's own software on the

device. In the desktop and server system market, this is not a concern, but on mobile systems, it is a central question.

The holdover of those carrier-controlled agreements for phone acquisition is the demand that devices be locked down. Devices are locked down first to a single carrier's network, so that devices cannot (legally) be resold as phones ready for any network. Second, carriers believe that they must fear the FCC (http://en.wikipedia.org/wiki/Federal_Communications_Commission) if device operating systems can be reinstalled.

On the first point, the HTC Dream, while somewhat more expensive than T-Mobile branded G1s, permit the user to install any operating system on the phone, and extract no promises from the purchaser. Google has no interest in locking you to a single carrier, but only to a single Google experience application vendor. Offering a user "carrier freedom of choice", while tying those users tighter to Google applications, is probably a central part of their marketing plans.

The second point, fear of an FCC crack down when mobile users have software freedom, is beyond the scope of this article. However, what Atheros has done with their Wifi devices shows that software freedom and FCC compliance can co-exist. Furthermore, the central piece of FCC's concern, the GSM chipset and firmware, runs on a separate processor in modern mobile devices. This is a software freedom battle for another day, but it shows that the FCC can be pacified by keeping the GSM device a black box to the Free Software running on the primary processor of the device.

Conclusion

Seeking software freedom on mobile devices will remain a complicated endeavour for some time. Our community should utilize the F/LOSS releases from companies, but should not forget that, until viable community forks exist, software freedom on these devices exists at the whim of these companies. A traditional "get some volunteers together and write some code" approach can achieve great advancement toward community-oriented F/LOSS systems on mobile devices. Developers could initially focus on applications for the existing "mostly F/LOSS" platforms of MeeGo and Android/Linux. The challenging and more urgent work is to replace lower-level proprietary components on these systems with F/LOSS alternatives.

Bradley M. Kuhn is the Policy Analyst and Technology Director at the Software Freedom Law Center. He worked during the 1990s as a system administrator and software development consultant for Westinghouse, Lucent Technologies, and numerous small companies. In January 2000, he was hired by the Free Software Foundation (FSF). From 2001 until 2005, he served as FSF's Executive Director, where he led FSF's GPL enforcement efforts, launched the Associate Member program, and authored the Affero GPL. In 2005, he left FSF to join the founding team of SFLC. Kuhn holds a summa cum laude B.S. in Computer Science from Loyola College in Maryland, and an M.S. in Computer Science from the University of Cincinnati. His Master's thesis discussed methods for dynamic interoperability of Free Software languages. He is also a director and president of the Software Freedom Conservancy, and a member of the autonomo.us committee, which studies issues of software freedom as they relate to software as a service.

Online Guide to Open Access Journals Publishing

From the Introduction:

This guide focuses on Open Access scholarly journals publishing. By "Open Access journals" we refer to the publication of peer reviewed scientific manuscripts under the umbrella of a specific journal title. The Online Guide to Open Access Journals Publishing is a web-based, living document that allows users to navigate quickly to specific areas of interest. Each chapter contains links to additional resources on the same topic in the form of: other documents and websites, tools and templates that can be adapted for your own use, and examples and best practices from other editorial teams to illustrate how the information can be implemented.

http://www.doaj.org/bpguide/

Open Collaboration within Corporations Using Software Forges

From the Abstract:

Over the past 10 years, open source software has become an important cornerstone of the software industry. Commercial users have adopted it in standalone applications, and software vendors are embedding it in products. Surprisingly then, from a commercial perspective, open source software is developed differently from how corporations typically develop software. Research into how open source works has been growing steadily. One driver of such research is the desire to understand how commercial software development could benefit from open source best practices. Do some of these practices also work within corporations? If so, what are they, and how can we transfer them?

http://dirkriehle.com/2009/02/11/open-collaboration-within-corporations-using-software-forges

Governance Models

Copyright: OSS Watch

From the Introduction:

A governance model describes the roles that project participants can take on and the process for decision making within the project. In addition, it describes the ground rules for participation in the project and the processes for communicating and sharing within the project team and community. It is the governance model that prevents an open source project from descending into chaos. This document explains why a governance model is necessary, considers some of the challenges associated with adopting a governance model in open source projects, and looks at the key areas such a model needs to cover. It also describes how to encapsulate your governance model in a governance document.

http://www.oss-watch.ac.uk/resources/governanceModels.xml

The Open Source Way: Creating and Nurturing Communities of Contributors

Copyright: Red Hat

From the Abstract:

This guide is for helping people to understand how to and how not to engage with community over projects such as software, content, marketing, art, infrastructure, standards, and so forth. It contains knowledge distilled from years of Red Hat experience.

http://www.theopensourceway.org/book/

April 9

Atlantic Canadian Entrepreneurship Expo

Fredericton, NB

An unbeatable lineup of keynote speakers, a tradeshow, and a networking lunch make up the full-day event. Hundreds of entrepreneurs come together to learn, grow, and share their success. Business relationships are made, new skills are learned, and deals are signed.

http://www.atlanticexpo.ca/cities/fredericton

April 22

Atlantic Canadian Entrepreneurship Expo

Halifax, NS

An unbeatable lineup of keynote speakers, a tradeshow, and a networking lunch make up the full-day event. Hundreds of entrepreneurs come together to learn, grow, and share their success. Business relationships are made, new skills are learned, and deals are signed.

http://www.atlanticexpo.ca/cities/halifax

April 26-28

Canada International Conference on Education

Toronto, ON

The aim of CICE is to provide an opportunity for academicians and professionals from various educational fields with cross-disciplinary interests to bridge the knowledge gap, promote research esteem and the evolution of pedagogy.

http://www.ciceducation.org/


MASSIVE INNOVATION
Lead to Win
Exciting new businesses that are in Lead to Win now!
selectstart
CheckinWiz
media
ucreatemedia.com
catwalk networks
SCTi
EclipseSource
Castel RESEARCH
zeebumobile
Videotelephony Inc.
NOVABRAIN TECHNOLOGIES Inc.
data bridge
A NEXIA SERVICES COMPANY
EnTeraSec security
polynate
Accentway
EventuSix
the speed of success
Triple Through Sustainability
SMART ROTOR SYSTEMS
SiliconPly
cornerportal
MARKTEK
Informatique Clearview informatics
NewEraHealth
MarkOff Security

The goal of the Open Source Business Resource is to provide quality and insightful content regarding the issues relevant to the development and commercialization of open source assets. We believe the best way to achieve this goal is through the contributions and feedback from experts within the business and open source communities.

OSBR readers are looking for practical ideas they can apply within their own organizations. They also appreciate a thorough exploration of the issues and emerging trends surrounding the business of open source. If you are considering contributing an article, start by asking yourself:

1. Does my research or experience provide any new insights or perspectives?

2. Do I often find myself having to explain this topic when I meet people as they are unaware of its relevance?

3. Do I believe that I could have saved myself time, money, and frustration if someone had explained to me the issues surrounding this topic?

4. Am I constantly correcting misconceptions regarding this topic?

5. Am I considered to be an expert in this field? For example, do I present my research or experience at conferences?

If your answer is "yes" to any of these questions, your topic is probably of interest to OSBR readers.

When writing your article, keep the following points in mind:

1. Thoroughly examine the topic; don't leave the reader wishing for more.

2. Know your central theme and stick to it.

3. Demonstrate your depth of understanding for the topic, and that you have considered its benefits, possible outcomes, and applicability.

4. Write in third-person formal style.

These guidelines should assist in the process of translating your expertise into a focused article which adds to the knowledgable resources available through the OSBR.

Upcoming Editorial Themes

April 2010:	Cloud Services
May 2010:	Communications Enablement
June 2010:	Growing Business
July 2010:	Go To Market

Formatting Guidelines:

All contributions are to be submitted in .txt or .rtf format.

Indicate if your submission has been previously published elsewhere.

Do not send articles shorter than 1500 words or longer than 3000 words.

Begin with a thought-provoking quotation that matches the spirit of the article. Research the source of your quotation in order to provide proper attribution.

Include a 2-3 paragraph abstract that provides the key messages you will be presenting in the article.

Any quotations or references within the article text need attribution. The URL to an online reference is preferred; where no online reference exists, include the name of the person and the full title of the article or book containing the referenced text. If the reference is from a personal communication, ensure that you have permission to use the quote and include a comment to that effect.

Provide a 2-3 paragraph conclusion that summarizes the article's main points and leaves the reader with the most important messages.

If this is your first article, include a 75-150 word biography.

If there are any additional texts that would be of interest to readers, include their full title and location URL.

Include 5 keywords for the article's metadata to assist search engines in finding your article.

The Talent First Network program is funded in part by the Government of Ontario.

The Technology Innovation Management (TIM) program is a master's program for experienced engineers. It is offered by Carleton University's Department of Systems and Computer Engineering. The TIM program offers both a thesis based degree (M.A.Sc.) and a project based degree (M.Eng.). The M.Eng is offered real-time worldwide. To apply, please go to http://www.carleton.ca/tim/sub/apply.html.