Experiences on Product Development with Open Source Software

Ari Jaaksi
Nokia
P.O Box 779
33101 Tampere, Finland
ari.jaaksi@nokia.com

Abstract. This article discusses Nokia's experiences of using open source in commercial product development. It presents the development model used in the creation of mobile consumer devices and highlights the opportunities and challenges experienced. This article concludes that the main benefits come from the utilization of already available open source components, and from their quality and flexibility. It illustrates the challenges and solutions faced when mixing open and closed development models at Nokia.

1 Introduction

The Nokia 770 and N800 Internet Tablets are mobile consumer devices. They provide wireless internet access and enable internet use cases such as voice and video calls, web browsing, messaging, and media consumption in a pocketable mobile device. Nokia has built these products on Linux and other open source components in a close collaboration with open source communities. In addition, Nokia runs the www.maemo.org web site that supports community development on internet tablets.

Nokia uses open source extensively in the creation of the internet tablets. We favor components that are developed by active communities and used by many users. This ensures that the selected components are developed and maintained properly both now and also in the future. For this reason we prefer to use mainstream desktop components whenever possible. Desktop and PC related projects are typically more active and mature than the projects targeting embedded devices. Therefore, we actually run a Linux based desktop configuration on a mobile device.

2 Software Architecture

Figure 1 illustrates our software architecture [1]. We integrate unmodified open source components into our platform. We also sponsor the enhancements of many existing open source components to make them fit for our use. We then integrate

Please use the following format when citing this chapter:

Jaaksi, A., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 85–96.

these modified open source components into our platform. Some components we develop from scratch, many of which we then open source. Finally, we integrate closed components from various sources, such as from commercial software vendors.

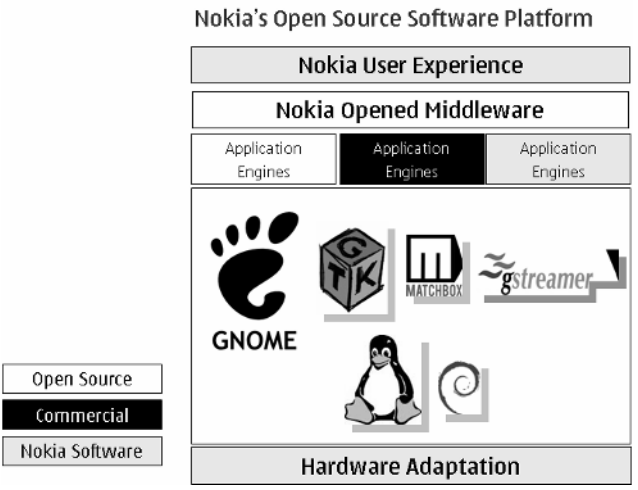

Figure 1: The Nokia open source software architecture

We have 428 source code packets in our platform. 25% of the packages are taken from open source projects without any modifications. Examples of such components include the gnuchess chess game engine, bzip2 data compressor, and id3lib for manipulating ID3v1 and ID3v2 tags in digital audio files.

About 50% of the packages originate form open source projects, but Nokia has made modifications to the components. In such cases, we actively push our modification upstream to the originating projects. The additional modification work is needed especially in the areas of UI and usability, power management, performance, and memory management. Our engineers work directly with communities participating development projects to ensure that our modifications are accepted upstream. In addition to making modifications ourselves, we also hire and ask developers within the communities to enhance components based on our needs. Examples of such components are the Linux kernel, D-BUS, GNOME-VFS, GTK+, GStreamer, and OBEX. We also reuse and improve entire subsystems and subsystem architectures, such as GNOME [2] and Debian [3]. Instead of separate components, we then reuse architectural blocks that already integrate several independent components.

Finally, some 25% of the packages are proprietary closed source components, either belonging to Nokia or licensed from commercial vendors such as Real Networks or Adobe. Some of the Nokia proprietary components that are kept closed are closely related to the hardware. Examples of such components are the boot loader and battery charging implementations. In addition, the majority of the user interface applications are also closed.

A European Union report by Ghosh [4] studied the software running on the Nokia 770. They concluded that the device runs 15 million lines of open source code, 200.000 of which where created by Nokia. This demonstrates that it is possible to use open source code and modify it to meet your own specific needs with minimal effort. In fact, Nokia manages to modify and use open source components for desktop environments in its mobile internet devices with less than 1.5% additional investment.

3 Community collaboration

We source our open source components directly from community projects. We do not use any embedded distros as the starting point of our architecture. Instead, we want to utilize mainstream desktop oriented open source components to get the maximum community benefits.

3.1 Selecting the core components

We analyze the technical suitability of all components and subsystems. All selected components must fulfill our functional requirements and meet our hardware specifications. The components also need to be of good quality and mature enough for consumer products.

We actively participate in the communities from which we source our components to ensure that the selected subsystems develop further over time. We also ensure that the goals of the development communities match our goals. This all happens through active community discussions, conferences, and workshops.

It is important that the open source components we use are licensed under proper licenses and have clear copyright and licensing information attached to them. We also choose to select components that do not lock us into one vendor through requirements such as mandatory copyright donations or dual licensing models. It is also important that our components be licensed under an open source license, such as LGPL, that allows us to integrate proprietary components into our platform as well.

For the key components and subsystems, we did not have too many options to choose from. For example, the only true graphical environment alternatives were Qt and Gtk+ [5][6]. We selected Gtk+ because it is developed by a vibrant multi-polar community with no single company dominance. It is therefore easy to contribute our changes to Gtk+ on the basis of general usefulness and technical merit only. Also, Gtk+ is licensed under LGPL, and that allows us to mix proprietary UI elements without a dual commercial license.

3.2 Creating software as a part of communities

Our strategy is to find a suitable community and then take part of the community work. We do not want to control the project or branch the work. Instead, working as

an integral part of a community provides us with access to code and engineers outside of our own development team.

As an example, we sponsored the development of the D-BUS [7] message bus system. We hired some key developers from the D-BUS community into our project but asked them to continue working within the project in open source. They then contributed code and participated in the development of D-BUS, and we performed a lot of testing that helped in reaching the needed product quality.

We open source new components and subsystems we have developed, such as our Hildon application framework. We have also opened the development of selected middleware components at the Maemo Sardine distro [8]. Open middleware development enables application developers to follow the latest changes in our code so they can test their applications against the latest changes, update them as a result of any API changes, and pilot the latest additions to our software. This open development allows anybody to participate in the development of the middleware code and see where it's heading. This is all available before a stable release of the software for the end-users. As an example, several parts of the code running on N800 Internet Tablet were already available before Nokia even announced the product in early January, 2007.

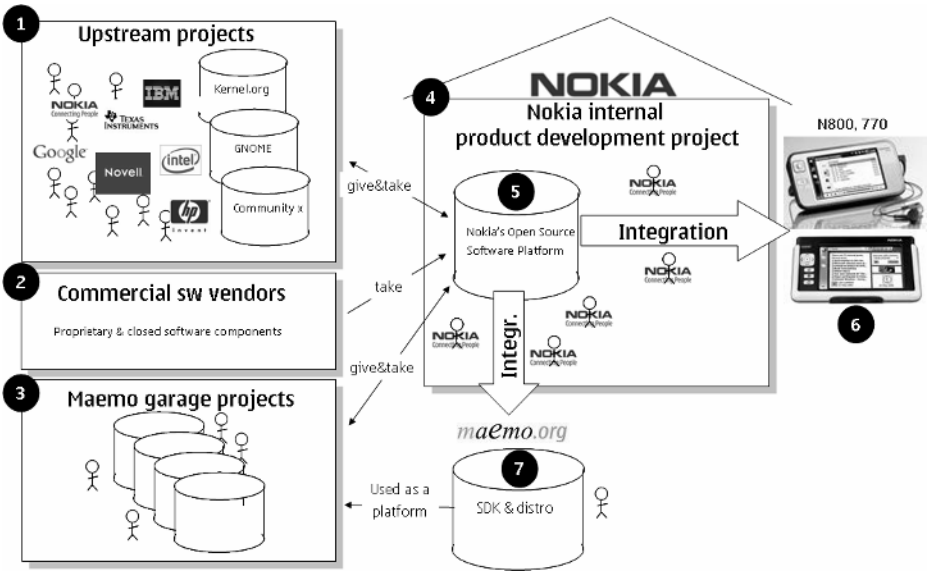

Figure 2: Working with communities

We work closely within communities to develop software, as illustrated in Figure 2. We collaborate with many individuals and companies in upstream projects (1) and Nokia engineers take part of the community work. We take selected components from those upstream projects (1), develop some code of our own (4), source components from commercial vendors (2) and create a Nokia internal distro called

Nokia's Open Source Software Platform (5). We then actively push our changes and modifications back upstream to minimize Nokia specific code.

We integrate the final software for our products within Nokia (4). We integrate both the product software for internet tablets (6), and the www.maemo.org tools and software for an external www.maemo.org software distro (7). While open source and communities help us in implementing software components and subsystems upstream (1)(3), we believe that the final product and product integration is better done within Nokia by Nokia (4). After all, we are responsible for meeting our quality, schedule, and monetary goals.

Finally, we offer www.maemo.org and Maemo garage for external developers (3). They use these facilities in their projects that develop software for Nokia's internet tablets. We are fortunate to have many volunteers and community people developing applications and submitting their work, such as documents, bug reports, and enhancements to www.maemo.org.

While Nokia provides the basic www.maemo.org infrastructure, the actual distro, and various development and community tools, it is the community members themselves who enhance them and provide support for each other. This greatly improves the developer experience on www.maemo.org. In the end of 2006, the www.maemo.org developer site hosted almost two hundred open source projects dedicated to the internet tablets, and had almost 60 000 unique visitors per month.

4 Benefits of open source

We have created two devices, provided software upgrades to these devices, and created an open source community around the www.maemo.org community site. Our development experiences include all the phases starting from initial requirements analysis to selling the devices, working together with many other companies and open source projects, and offering upgrade software for end users. We thus believe that we can draw some conclusions about developing consumer products with open source. The benefits are clear.

4.1 Efficiency

The biggest efficiency gains came from the utilization of already available components, such as the Linux kernel and the GTK+ toolkit. It was cheaper and more efficient for Nokia to build the internet tablets using the open source model than it would have been using a proprietary one. This conclusion can be drawn by studying other similar product development activities at Nokia.

In reality, developing an own operating system and middleware was never even an option for us. We needed to either use an existing commercial and closed operating system and middleware, or then use an existing open source operating system and middleware. We used the open approach in order to benefit from the cheaper or non-existent licensing costs, in order to have better strategical control,

and to have the ability to freely enhance the code according to product and market needs.

Productive software developers can enhance development efficiency significantly. With proprietary and closed software systems, we typically train and educate developers for a long period of time. It takes several projects for the developers to become productive with the closed and proprietary systems and technologies we use at Nokia. This is not the case with our open source based software platform, because we use widely known tools, components and architectures. The Linux operating system and Debian packages, for example, are commonly taught in universities and other companies. That makes new developers productive faster than with other software platforms used at Nokia.

4.2 Quality

The code that we obtain from open source projects is of better quality and has fewer errors than code we developed by ourselves. This is because open source code has already been used by others before we take it into use, and they have already fixed the most severe errors.

However, if we compare the open source code to the commercial components used in our platform, the quality difference is not that obvious. The commercial components have typically been used by others, too. That has improved their quality.

An additional benefit to open source is that the quality of the code and the skills of developers can be verified in advance. We can study the component code, build prototypes, and run performance tests freely with open source components. This helps us to select good quality components and subsystems. Also, when a developer or a subcontractor submits code to an open source project, the quality is easy to verify. This allows us to assess the quality of our developers and subcontractors before hiring them.

4.3 Flexibility

Open source provides flexibility when we need to fix problems or change functionality. We often request bug fixes or modifications for the commercial closed components on our platform. However, if the vendor of that particular component does not have the capacity or willingness to fix the problem on time, we can be left few options. Typically we cannot fix problems ourselves in these scenarios, because the companies from whom we license our closed components don't want us to access their source code. With open source components, however, we fixed bugs ourselves, hire somebody else to fix them, or work with the communities in order to obtain the modifications. With so many options available, we are able to fix the problems we have in most cases.

4.4 Software licensing

Software in-licensing requires a lot of negotiations between a licenser and a licensee. Based on our experiences, an average in-licensing process for a software component takes 6 – 12 months. Problems and delays in software licensing are one of the most common reasons for missing features or delayed projects.

In contrast, licensing with open source is simple. The licensor already has the license terms in place. She may offer some additional options, such as support or training, but the actual licensing terms are already established. In addition, all the source code is available for the licensee to study and evaluate. The licensee can also assess the community, companies, and available hackers supporting the technology in question prior to taking it into use. And, they are able to talk to others about the technology without worrying about trade secrets. Because of these factors, open source projects are never delayed because of complicated in-licensing negotiations.

Open source simplifies and accelerates software licensing, and reduces technology and quality risks. Instead of negotiation for months, the technical work can start immediately.

4.5 Future and roadmaps

The future direction and plans for open sourced components and subsystems are typically discussed openly, and are open for contributions. We can, therefore, monitor and influence the development of relevant technologies through the community work.

The choices are more limited with closed source commercial components. Companies developing closed source components typically decide themselves about the future of their technology. They may choose to reveal parts of they plans, and they may choose to take external input into account. But, unlike in the open source, you cannot participate yourself and contribute in an open fashion.

4.6 Open source and confidentiality

An open source approach requires openness and information sharing during development. However, you do not want to reveal the products to the public before the actual product announcement. There is thus a potential conflict between the open source openness and product launch secrecy.

Nevertheless, we worked intensively with communities already before we announced the Nokia 770 Internet Tablet. Also, we opened parts of the firmware development before launching the Nokia N800; we worked with several communities to develop code for it. Many community developers had very detailed information about our forthcoming products due to their involvement in this process. Despite this, however, we had no information leakage from developers prior to the commercial product announcements. Based on our experiences, therefore, it is possible to develop software openly, while maintaining product confidentiality.

5 Issues and challenges

The open source development model is different than the closed one. The open source model shares code and work with other people and companies. They have their own schedules and targets that may not coincide with ours. However, that doesn't necessarily matter as long as we work upstream on non-differentiating aspects of the software.

At the end of the day, however, we must ensure that we get our products done in time with proper quality. Thus, at some point we need to drive the project to the conclusion that benefits us the most. This typically requires a more closed and single company controlled way of working. This mixture of open and closed development, illustrated in Figure 2, is important to master and we already can draw some conclusions from our experiences.

5.1 Hacking vs. stabilizing

In the early stages of a product development project, we work closely with communities, individual hackers, and hacker companies. We develop code in a true hacking mode. Later, we freeze our requirements to get things focused and to get software ready for shipment.

We have an internal milestone when we all software functionality must be implemented. We predict the shipment date to synchronize marketing activities, and reserve a factory production line. At this milestone, System Testing can run all test cases. All features are implemented at this point, but the system is still unstable and buggy. From this point on, all effort is put into bug fixing and stabilization.

At this milestone, the whole development team switches modes, from hacking and new development, to integration and stabilizing. Hacking and new development happens around independent components within teams. Integration and stabilizing, on the other hand, happens around the entire software stack and between teams. This requires a shift from a component view to a system view of software development.

This is a radical change of mode. The open source culture is very much for trials, hacking, innovation and other creative aspects of software development. Meeting deadlines, not developing new features, and focusing on stability are not what many open source communities or developers naturally do. In addition to us, the Linux project, Debian, and others seem to have difficulties making a final good quality release on time [9], [10].

In recent projects we have made the move from the hacking to stabilizing more apparent and strict. We now make the change very explicit in our process, and enforce it even more than in some conventional product development projects. Accordingly, we managed to get N800 ready right on time with no delays. This proves that we have managed to improve our stabilization phase.

5.2 Architecture management

Open source requires us to manage our architecture not only form the conventional 4+1 point of view [11] but also from the legal and IPR point of view. Some components, such as players and codecs, are available only as closed source components. We need to, therefore, mix open source and proprietary code and manage different licensing rules within the product code.

We manage the legal and IPR status of each software component. It is not enough to manage the architecture in terms of the development time API compatibility or run time performance, but we also need to manage the mix of various open source and closed source licensing rules. This is an additional job that we need to do when developing products based on open source.

5.3 Community alignment versus backwards compatibility

Internet tablets provide a platform on top of which applications and services can be developed. It is important that the platform provides application compatibility over product generations.

A binary compatibility is an ultimate goal for such backwards compatibility. That would allow the same application to run on various platform and product generations without recompilation. This is typically achieved by selecting development APIs and components that remain unchanged over platform generations. If a company developing product generations also develops all the software, such API freezing can be achieved by not introducing new features and changes to the relevant code.

However, open source components develop constantly. They get new features, their architecture change, and bugs are fixed. This development happens and it cannot be stopped. Our current strategy is to stay close to the latest development. We want to avoid a big difference between the component developed in the community and the component version used by us. For example, we are eager to move to the latest Linux kernel version as soon as it is possible for two reasons. We want to get the latest development into use, and we want to minimize the delta we need to maintain our selves.

These two concepts, product backwards compatibility and staying current with open source development are somewhat contradictory. In several cases, it would have been easier for us to provide backwards compatibility simply by not changing the underlying code. However, communities move on. If we decided to use an old version of a component we would need to do all backporting and new development ourselves to that component. Communities would not help us for they are already working on new versions. That would eventually put us on a different development branch and increase the amount of code we need to develop and maintain ourselves.

We do not have a final answer to this contradiction. So far we have not managed to maintain true backwards compatibility as we had hoped. Going forward, this is one of the things we need to better understand and manage. To do so, we are implementing more strict architecture management and compatibility layers between of the open source originated platform components and applications.

5.4 Community participation in product integration

We work closely within communities in the upstream projects to develop various software components. This work happens in a community mode. Then, when we decide on the product features and integrate the final software, we do it ourselves in Nokia internal product development projects, illustrated in Figure 2. The closed way of integrating the software causes frustration among some external open source developers. They'd like to find a way to be part of the Nokia's product development, not only in developing components and technologies in upstream projects but actually deciding on features, and integrating the final product together with Nokia.

We launched the Sardine distro [8] partially to address this problem. We now allow external developers to participate more in the actual development and integration process within Nokia. This is this is one of those areas where we still need to collect experiences and learn. There may be room for more community collaboration even in the most crucial steps of the product development. But in all cases, we expect that the product companies, such as Nokia, must have the final control over the product features and quality. It cannot be given to communities.

5.5 Investing in community work

Using open source code effectively requires community participation. It is sometimes possible to use an open source component without further development. Such participation is almost free of charge. In many cases, though, we work with communities to enhance components and develop them further. Such participation requires extra resources.

When open sourcing our own code and patches, we must ensure that our developers can work with the open sourced components. We must continue support the code in open source so that the code will meet our future needs, too. Just releasing code with no plans to develop it further won't benefit us.

We have open sourced individual components and participated some development with no clear benefit for us. We have either been left alone to develop the component, or our needs have not been taken into account when developing a component further. In these cases the joint open source development didn't happen or it didn't benefit us. Therefore, we now observe individual projects and try to identify when the open source investment pays off and when it doesn't.

6 Summary

We have created two consumer devices, the Nokia 770 and the Nokia N800 Internet Tablets, utilizing open source software. In addition, we have made software updates to those devices and initiated community work around the www.maemo.org web site. Our experiences demonstrate that an open source technology and

development model is well suited for consumer devices. We have created products in a shorter time and with fewer resources with open source than we have managed to create using proprietary software alone. In essence, open source offers time and cost savings in a form of readily available components and subsystems, available developers, and an effective development model.

Open source doesn't make software development free or easy. It provides effective tools for product creation. Combining these new tools, such as community involvement, and utilization of open components, with more traditional software and product engineering practices is a good mix.

As a device manufacturer, we alone are responsible for the quality of the end product. We must therefore utilize all quality and software engineering mechanisms to achieve the needed quality. We cannot skip such development aspects such as specifications, integration, testing, and documentation, for example. In addition, open source introduces certain new requirements, such as community interaction and legal and IPR management. These hard requirements seem to contradict with open community work in certain cases. We have not managed to successfully solve all these conflicts. However, we are working on improving community participation in the stabilization process as well as allowing community members to participate firmware development. The results are not yet known, though.

Acknowledgements

Thanks to Bradley Mitchell for his assistance in writing this article.

References

[1] Jaaksi 2006: Building consumer products with open source http://www.linuxdevices.com/articles/AT7621761066.html

[2] http://www.gnome.org/

[3] http://www.debian.org/

[4] Ghosh, R A (ed.), 'Study on the:Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU, Final report, November 20, 2006, Merit

[5] http://www.trolltech.com/

[6] http://www.gtk.org/

[7] http://www.freedesktop.org/wiki/Software/dbus

[8] http://sardine.garage.maemo.org/

[9] Glance, David, G : Release criteria for the Linux kernel, 2004: http://www.firstmonday.org/issues/issue9_4/glance/index.html

[10] Brockmeier, Joe, The 2005 Debian Project Leader election, 2005 http://lwn.net/Articles/127031/

[11] Kruchten, P.B. The 4+1 View Model of Architecture. In IEEE Software, November, 1995: 42-50