OPEN SOURCE IN DEPENDABLE SYSTEMS: CURRENT AND FUTURE BUSINESS MODELS

Cyrille Comar and Franco Gasperoni
AdaCore/ACT Europe
8 rue de Milan, 75009 Paris, France
comar@act-europe.com
gasperoni@act-europe.com

Abstract This article begins by describing the legal foundations of business models in the software industry. It then introduces the Free Software (FS) and Open Source Software (OSS) movements, and surveys various business models found in the FS/OSS based on them. A special emphasis is placed on the applicability of these business models to industries that produce dependable systems. This article concludes by exploring a FS/OSS co-op model that could enable the development of software components for dependable systems.

Keywords: Business Model, Free Software, Open Source Software, Co-op, Safety-Critical Software, Integrated Modular Avionics.

1. Introduction

This section provides the legal background to understand business models commonly adopted in the computer software industry. It then explains the Free and Open Source Software movements and how they relate to dependable systems.

Legal Background

Copyright is the exclusive legal right to copy, distribute, modify, display, perform, rent or more generally exploit a work by its copyright holder. Copyright applies to software works. In the context of software, both source code and binaries are protected by copyright. Loading a copy of the software onto a computer is considered copying. Only the copyright holder of a software program is allowed to copy, modify, make derivative works, and distribute the software. Anybody else needs permission from the copyright holder. In the software industry this permission is called a software license.

Nothing requires the copyright holder of a software work to give a license, or to give the same license to all possible users. Copyright holders can, at their discretion, charge a fee in exchange for a copy of the software work and its attached license. In this case what the recipient is buying is not the software per se but a copy of it. Virtually all software products are sold this way.

The software license specifies the conditions under which the recipient can (and more often, cannot) copy, distribute or modify the copy of the software received.

Free Software and Open Source Software

The *Free Software* (FS) movement created by Richard Stallman in the early 80's aims to "preserve, protect and promote the freedom to use, study, copy, modify, and redistribute computer software, and to defend the rights of FS users" [1]. To protect the freedoms of FS, Richard Stallman created the GPL (General Public License) [2].

In this respect, FS is no different from proprietary software sold by companies like Microsoft. A FS program comes with a license (the GPL), just as does software from "closed-source" software vendors. The big difference between the GPL and closed-source software licenses is that the GPL is written to favor the users of FS, while the license of closed-source software vendors is written to allow as little as necessary, and to favor the closed-source software vendor. Specifically, the GPL allows:

- Copying with no restrictions of any kind;

Furthermore in the case of redistribution the GPL requires:

- Redistribution of original software with sources;
- Redistribution of derived works with full sources.

Note that the term "free" in Free Software is intended to connote "freedom" but is often misinterpreted as "no cost". As a result the term *Open Source Software* (OSS) was coined to address this problem. Given the rapid gain in popularity of the FS/OSS movement, unscrupulous software vendors have claimed their programs to be "Open Source" by simply adjoining the sources along with their binaries - while imposing draconian restrictions on what could be done with those sources. Such actions repudiate the notion of freedom that founded the FS/OSS movement.

To counter such abuse, the Open Source Initiative (OSI) was created to unambiguously define the term "Open Source Software" [3]. This non-profit corporation defines "OSS", and lists a number of approved OSS licenses that match the definition. There are many approved OSS licenses. In addition to

the GPL, examples include: the BSD License, the MIT License, the Apache Software License, the Mozilla Public License. There are over 50 approved OSS licenses enumerated on the OSI site [4].

FS/OSS and Dependable Systems

FS/OSS projects have blossomed due to the synergy created by openly sharing software and ideas. The FSF/UNESCO Free Software directory contains over 3,000 packages from visible projects such as GCC (the GNU Compiler Collection) to Gnome (the GNU desktop [5]). The SourceForge site alone hosts over 80,000 projects accessed by nearly 850,000 registered users [12].

As opposed to most existing FS/OSS, the software in a dependable system cannot be reduced only to a set of sources with build scripts and instructions. Evidence of dependability also resides in the availability of other artifacts, such as requirement documents, system models, design specs, proofs of correctness, test suites, verification and validation processes. In the most critical safety and/or security contexts, these kinds of evidence are formalised as certification materials to be provided according to the requirements of a recognized, often international, standard. The creation and evolution of the software component in a dependable system must be part of an auditable and repeatable process with stringent quality requirements.

These aspects of system development are not the primary concern of FS/OSS projects. This does not mean that FS/OSS programs are not dependable. It does mean that a record of quality assurance measures used to achieve dependability in a FS/OSS application is often not readily available. The focus of a FS/OSS project is typically to offer a useful, reliable software program to a given community.

Given that FS/OSS projects have shown that they can produce high-quality, widely available software programs for the benefit of all, it is natural to wonder how people building dependable systems can leverage, or even contribute, to the FS/OSS movement. In [6] the authors analyze this question in the area of safety-critical systems. The question they ask is: "What would be the benefit of using FS/OSS in safety-critical systems, and under what conditions can FS/OSS be used"?

In the remainder of this article we look at business models that have sprung up in the context of FS/OSS and analyze their relevance to the production of dependable software. We conclude by elaborating one of them and examining how it might be adapted to the benefit of the dependable software industry.

2. Business Models and the Computer Industry

The business models underlying most of the traditional computer industry rest, at least initially, on the notion of supplier scarcity. Put simply, there are three basic types of scarcities: product, expertise, and infrastructure.

An example of software product scarcity is the MS Windows operating system: you can purchase MS Windows only from Microsoft. An example of hardware product scarcity is the Pentium-compatible microprocessor: you can purchase Pentium chips from Intel, the originator of the Pentium design, or AMD. An example of expertise scarcity is the expertise required to manage a large IT (Information Technology) project: you can purchase this expertise from companies such as IBM Global Services, EDS, Cap Gemini, and a few others. An example of scarcity of infrastructure is the infrastructure developed by Dell in the PC industry. The PC industry has become a commodity industry, i.e. an industry with an abundance of suppliers. Thanks to its infrastructure, Dell has evolved from a garage-like operation to the PC industry leader.

In closed-software products, initial scarcities can be made into long-lasting ones thanks to the creation of technical entry barriers that give rise to monopolies or oligopolies. This phenomenon is well described in [7].

3. Business Models of FS/OSS

In contrast to traditional scarcity-based models as described in the previous section, the FS/OSS movement has created economies of abundance leading to a "community/network effect". In the "network effect", a community of software developers is created. This community, and the mind-share that is associated with it, is leveraged to sell products, expertise, infrastructure, advertising space, or some mixture of these.

The purpose of this section is to explore some business models involving FS/OSS. It will also examine how the three previously mentioned kinds of scarcities (product, expertise, infrastructure), along with the "network effect", are used to generate revenue. The list given here is by no means exhaustive.

Pure FS/OSS Product

As for any software product, there is no restriction on the price that can be charged for software licensed under a FS/OSS license. In this model the FS/OSS vendor limits itself to selling a standalone, self-contained set of FS/OSS programs with perhaps some minimal installation help.

Recipients of FS/OSS can buy one copy of the software and install/run it on as many machines as they like. This means that the conventional sell-per-unit business model does not work very well for FS/OSS since FS/OSS licenses favor an economy of software abundance. Furthermore, recipients of FS/OSS

programs have themselves the freedom to redistribute copies of the software product and to charge for it. This limits the amount that the initial vendor can charge to pretty much commodity prices, unless the FS/OSS product is both one-of-a-kind and sold in low volume to address specialized business needs.

Thus, this business model leverages solely on scarcity of infrastructure (assembling of proper FS/OSS packages and redistribution). Most commercial GNU/Linux distributions until recently were based on this pure FS/OSS product commodity price model. Because the infrastructure required for this kind of operation is easily available, this type of business model has been abandoned by a number of GNU/Linux distributors.

For developers of dependable systems, this business model is not particularly attractive or relevant. Development of dependable software is an expertise-intensive task usually employing sophisticated technology. As such, buying a CD-ROM at commodity prices with little or no associated service is not particularly interesting for two reasons. Either the FS/OSS product is a sophisticated toolset (in which case it is critical that high-quality service accompany the product, regardless of its FS/OSS status), or it is a component, such as a library or operating system to be embedded in the final dependable system (in which case getting a CD containing sources is nice, but not nearly enough to meet the customer's need for evidence).

Dual License Product

In this model, the FS/OSS work is distributed at no cost under the terms of the GPL, as well being sold under a different license to customers who do not want to be bound by the terms of the GPL. This model is available only to the copyright holders of the software, and leverages product scarcity. It is viable only when the software being sold is included, in whole or in part, in another application that is in turn sold by a customer under terms different from the GPL. If a customer were to include in his own work FS licensed under the GPL, then his application would be considered a derivative work of the FS, thereby forbidding redistribution unless the whole application were licensed under the GPL.

MySQL, a database, and Cygwin, a UNIX-like environment for Windows, are licensed this way. This is an interesting illustration of the fact that the copyright holder can distribute the same version of a software work with different licensing conditions.

With respect to dependable systems, this business model rests on vendor lock-in. It has the advantage, relative to the completely proprietary model, of offering the sources of the product at no extra cost. It may also leverage a larger community of users and perhaps additional expertise independent of the vendor. Apart from these, it offers no significant advantage over a propri-

etary software model. Its appropriateness to industries developing dependable software depends more on the relevance of the product and associated services than on any enhancement of value due to the business model.

The Proprietary FS/OSS Product

This business model is an oxymoron, or we could call it the have-your-cake-and-eat-it-too business model. In this model the software vendor assembles a set of FS/OSS and proprietary applications, and markets the bundle on the basis of perceived synergy among them. The vendor uses the conventional charge-per-unit model where the unit price can be higher because of the scarcity created by the proprietary components in the bundle. Some GNU/Linux distributors have adopted, often temporarily, this business model.

This model is so anchored in the conceptual milieu of proprietary software business that it offers no advantage to the customer over the conventional closed-software model.

FS/OSS Leveraged Hardware

With the advent of 64-bit chips such as the Itanium from Intel [8] a number of hardware vendors have announced 64-bit workstations running (sometimes customized versions of) Linux. An interesting example is the SGI Altix platform [9]. In this business model hardware vendors leverage the Linux "network effect" to offer high-performance servers at premium prices.

Bundling a FS/OSS operating system (OS) and related software components, along with services covering the complete platform, is an interesting offer to developers of dependable software: the entire system on which their application rests becomes open for inspection. The QA and other documents typically needed when developing dependable software are not available, but for certain types of dependable systems having access to the OS source code allows a dependable system vendor to create an in-house team to qualify and tailor the OS according to the dependable system requirements. The additional support provided by the hardware vendor for the platform (and typically bundled into its premium pricing) is a differentiating factor for customers developing dependable systems, compared to commodity hardware with little or no vendor support.

Infrastructure Provider

Companies such as VA Software [10] with its OSDN (Open Source Development Network, Inc.) [11] leverage scarcity of infrastructure to provide an OSS development website, SourceForge [12] with a large repository of FS/OSS projects for whom the basic service is free of charge, while advanced services come at a fee. OSDN delivers more than 222 million page views and

reaches 12 million unique visitors per month. On the strength of this network effect, some of its revenue is indirect from web advertising. Other online revenue comes from per-user annual subscriptions that give access to advanced services on the SourceForge.net web site, such as advanced search capabilities, priority technical support, project monitoring, etc. The annual cost per user is low (less that USD 50 in 2004).

VA Software also sells SourceForge Enterprise Edition [13] to manage and execute offshore and distributed team development. The SourceForge application itself does not appear to be FS/OSS. VA Software leverages the free service it provides to the FS/OSS communities to sell and promote its SourceForge product.

This business model is interesting for developers of dependable system that have a large and possibly distributed development teams. Leveraging the experience with collaborative development in the FS/OSS community and using the same infrastructure that was created to support it can be an attractive solution.

Pure Service

In this model a service company builds expertise around a number of FS/OSS works and helps its clients with installation, support, use, maintenance, upgrades and adjustments or customizations that may be needed. This business model is based on a relative abundance of software, but scarcity of expertise, to market service. An example is Alcove, a European company providing professional FS/OSS services [14]. Another interesting example in this domain is what IBM is doing with Linux with over 2,000 Linux-skilled IBM Global Services professionals [15, 16].

Offering service, and in this case expertise, is certainly interesting in the context of dependable systems development, if the consultants being engaged are experts and contributors to the FS/OSS being used in the construction of the dependable system. The difference between this business model and that of "regular" service companies is that in the FS/OSS case consultants can inspect and contribute to the development of the FS/OSS product. As such they are in a position to provide a deep level of know-how to the client developing a dependable system.

Leveraged Service

A leveraged service is an expertise-based service that facilitates the effective application of a FS/OSS product to specific business needs. This product/service combination is typically sold as a yearly subscription comprising the FS/OSS toolset with upgrades, high-quality toolset support and online consulting from toolset specialists. GNAT Pro, the FS development environment

for the Ada programming language used in mission- and safety-critical systems is an example [17].

We use the GNAT Pro offering to illustrate the value proposition of this model. GNAT Pro customers receive a high-quality FS product accompanied by service directly from Ada experts and the GNAT Pro developers. The GNAT Pro experts serve as partners to the customer's development team. They work closely with the customer team to help them make optimal use of GNAT Pro and to assist them with all aspects of their Ada software development. As part of that dialogue the GNAT Pro engineers regularly address issues such as code optimization, programming language semantics, multi-language systems, and code organization. In addition to the benefits stemming from a high-quality product, this level of service leads to higher productivity and reduced risk in projects using GNAT Pro.

This business model leverages the subtle combination of two scarcities: scarcity of the FS/OSS expertise and scarcity of the FS/OSS quality assurance (QA) infrastructure. In the case of GNAT Pro, the GNAT Pro development team has built a sophisticated QA infrastructure comprising over 30,000 real-life test cases (a total of over 6 million lines of Ada source code) and automated capabilities for execution and evaluation of the tests, supporting nightly integration, regression testing and analysis on multiple platforms.

This business model works for FS/OSS used in fairly to highly technical software projects requiring sophisticated tools, libraries, and other software components, along with a high degree of expertise, and of confidence in the reliablility of the products. The software deployed in a dependable system is often of this sort. As a result, this business model is well-suited for industries producing highly-dependable software. What's more, this business model is in the interest of the companies ultimately responsible for the software used in dependable systems because it aligns their interests with those of their tool vendors. A company developing software for a dependable system is interested in receiving a high-quality product with high quality service in order to develop its dependable application. The only way a FS/OSS company can secure recurring revenue with this business model is by ensuring that the customer renews his subscription. This will happen only if the customer is happy with the technology and the service he receives: unlike for closed-software, the FS/OSS product alone provides no vendor lock-in.

OSS Co-Ops

The Eclipse Foundation [18] and the Apache Software Foundation [19] have an interesting and unconventional business model.

Eclipse is an open, extensible IDE (integrated development environment) built on a mechanism for developing, integrating, and running modules called

plug-ins. Put it another way, Eclipse is a platform providing a common IDE infrastructure for tool providers to plug-in their tools.

The Apache Software Foundation provides support for the Apache community of open-source software projects such as the Apache HTTP server.

Both the Eclipse Foundation and the Apache Software Foundation are non-profit organizations. They are characterized by a collaborative, consensus-based organization and development process. They form a tightly-coupled community of developers and users governed by an open and pragmatic software license. We call these types of organizations OSS cooperatives or OSS co-ops.

A general-purpose co-op is an association formed and operated for the benefit of those using it. Its business purpose lies more in cost reduction than in revenue generation. The roots of the co-op concept lie in the work of Peter Kropotkin in 1902 [21]. In his book Kropotkin describes how in Siberia animals, instead of competing for resources, have to work together to stay alive. Throughout his book, Kropotkin stresses that cooperation is the main factor in evolution, rather than the competing forces posited by Darwin and his adherents.

It is precisely this drive for cooperation to solve a mutual problem, too hard or too costly to solve alone, that is the business case for OSS co-ops. The idea of industrial cooperation is not new. The Eurofighter consortium [22] is a good example of cooperative effort across aerospace companies and nations. What is novel, and what the FS/OSS movement has shown, is that this idea can be applied to the development of software.

In the next section we look into how this concept of OSS co-ops can be put to work in the development of software for dependable systems.

4. OSS Co-Ops and Dependable Systems

In this section we explore, through an example, how the OSS co-op business model could lead to a new model for the development of software components in dependable systems.

Example: Integrated Modular Avionics

Our example is set in the avionics industry - an industry where dependable software is crucial. Until recently, the avionics industry has relied on federated architectures where separate avionics subsystems (computers) each perform a dedicated function. With the constant increase in aircraft functionalities, computional requirements, and corresponding embedded software, the life-cycle cost of this federation of individual systems has led aerospace companies and program offices to seek alternative approaches. The main solution is a move towards generic computing platforms that can run multiple applications con-

currently, possibly at different levels of safety criticality. This approach, known as Integrated Modular Avionics (IMA), results in a reduced number of subsystems (computers) and reduced weight [23, 24]. One of the outcomes is the need for an operating system to support the IMA software architecture.

When an aeronautics company FlightEX wants to supply IMA-compatible avionics software, for instance a flight management system (FMS), they need to re-engineer their current FMS software to execute under an IMA-compatible operating system (OS). The re-engineering effort of their FMS is within the core competence of FlightEX since the FMS represents an important part of its avionics software know-how. On the other hand, creating and maintaining an IMA-compatible OS layer is not part of the usual activities of FlightEX. Since this is a requirement for new avionics programs, it must be done. To solve this dilemma FlightEX has adopted the following strategies:

- Follow the conventional "do-it-yourself" approach. Use internal resources to design, develop, provide certification evidence, and maintain the IMA-compatible OS.

- Purchase a commercial off-the-shelf (COTS) OS and adapt it to be IMA-compatible along with providing all the certification artifacts necessary for its deployment in avionics applications.

- Team up with commercial off-the-shelf (COTS) OS vendors to provide avionics guidance and funding for the development of IMA-compatible OSes.

Example Continued: What to Do for the OS?

Faced with the absence of IMA-compatible OSes it is tempting for FlightEX to develop its own OS, or internally adapt an existing one. On the positive side, the OS specifications can be customized to match the exact needs of FlightEX. Furthermore, the first steps for the development of such an OS are based on the company's detailed avionics know-how. On the negative side, internal resources are diverted from the core avionics focus of FlightEX, in addition to the necessity for long-term maintenance of the home-grown OS and the related expertise. Further, the adapted FMS, which is one of the company's core products, runs the risk of being less flexible to deploy in other IMA architecture implemented over a different OS choice.

Faced with this unpleasant situation a number of avionics companies have transferred the commercial rights of the IMA-compatible OS to a commercial vendor. The idea being that the vendor can sell the IMA-compatible OS to other avionics companies and, with the resulting revenue, ensure long-term maintenance and evolution of the OS. This approach is based on the insight

that the IMA-compatible OS alone will not be the key differentiating factor when competing for avionics projects.

Following a proprietary COTS-only approach has its own risks. In fact, a commercial offering can live for as long as it is profitable. This is an issue when the required expertise and related investments are high and the customers few. This, along with the viability of the COTS supplier, is a risk factor when software choices must stand for decade-long projects. In addition, there are potential geo-political concerns: European avionics companies may need to avoid COTS OSes restricted by US export regulations and vice-versa. See [6] for a complete discussion of the risks related to the use of COTS in avionics software.

5. Conclusion: Towards an OSS Co-Op

In addition to the home-grown or the COTS approaches described above, a cooperative approach may provide an interesting middle ground. Instead of going alone, or relying on a third party, aeronautics companies and others sharing similar OS needs, can cooperate towards a common OS. The objective would be to ensure long-term availability to all co-op members of the OS along with a reduction in their overall costs of deployment.

The OSS co-ops such as Apache and Eclipse offer an interesting model for sharing the development and maintenance effort as well as the ownership of the software. They also provide interesting ideas on how to manage software independence from any particular member of the co-op. In the case of the avionics industry, the co-op model needs to be enhanced to take into account the specificities of safety-critical software (creating and sharing certification material, putting in place a strict QA infrastructure for software evolution, etc.)

- Sharing the source code is insufficient. Other things need to be shared and jointly owned, in particular certification materials.

- A strict QA infrastructure needs to be put in place as part of the evolution process.

As in the Apache and Eclipse cases, an independent organization made of its co-op members can play the leading role in hosting and maintaining the overall infrastructure, as well as guaranteeing the quality and independence of the certification material that is shared by co-op members. The software would be available as OSS and made available publicly, to spread its use and allow third part vendors to adapt their toolset easily. To encourage others to become members of the co-op, certification artifacts along with support and expertise as detailed in the "leveraged service" business model would be provided to members of the co-op for their projects.

This software co-op business model leverages the economics of sharing which has made the success of the FS/OSS movement. A COTS OS vendor may even get it started by pitching its own OS, in order to gain a decisive competitive advantage over its competitors. A fantasy?

References

[1] http://www.fsf.org/

[2] http://www.gnu.org/copyleft/gpl.html

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

[4] http://www.opensource.org/licenses/

[5] http://www.gnu.org/

[6] Logiciel libre et sûreté de fonctionnement, by Philippe David and Hélène Waeselynck Editors, 2003, Lavoisier. In French.

[7] The Gorilla Game, by Geoffrey A. Moore, Paul J. Johnson, and Tom Kippola, 1998, Harper Business Publisher.

[8] http://www.intel.com/products/server/processors/server/itanium2/

[9] http://www.sgi.com/servers/altix/

[10] http://www.vasoftware.com/

[11] http://www.osdn.com/

[12] http://sourceforge.net/

[13] http://www.vasoftware.com/products/index.php

[14] http://www.alcove.com/

[15] http://www.ibm.com/linux/

[16] http://www-5.ibm.com/services/be/pdf/linux_services_opportunity.pdf

[17] http://www.gnat.info/

[18] http://www.eclipse.org/

[19] http://www. apache. org/

[20] http://www.eclipse.org/

[21] Mutual Aid: A Factor of Evolution, by Peter Kropotkin, 1902. Available from Porter Sargent Publisher (December 1976). Also at http://socserv.socsci.mcmaster.ca/ econ/ugcm/3113/kropotkin/mutaid.txt

[22] http://www.eurofighter.com/

[23] DO-255: Requirements Specification for Avionics Computer Resource, RTCA, 2000. Available at http://www.rtca.org/onlinecart/product.cfm?id=193.

[24] ARINC 653-1: Avionics Application Software Standard Interface, ARINC, 2003. Available at https://www.arinc.com/cf/store/catalog_detail.cfm?item_id=495.