International Free and Open Source Software Law Review

Volume 05} Issue 01} March 2013

ISSN 1877-6922

Available online at: http://www.ifosslr.org

Peer reviewers

The Editorial Committee wishes to thank the work of the many referees and peer reviewers whose professional expertise and dedication to high standards have made the publication of this issue possible.

Contact

All administrative, bibliographic and pre-publication enquiries should be directed to the Editorial Coordinators via email at:

admin@ifosslr.org

The Editorial Committee can be contacted via email at:

ed-com@ifosslr.org

Policies and bibliographic information

Copyright and licensing statement

IFOSS L. Rev. is committed to the improvement of understanding of legal issues in digital society. A licensing statement is therefore attached to each article, clearly outlining the particular terms which apply to the article. Most use Creative Commons licences with special exceptions for translations.

Graphic design

The Editorial Committee wishes to thank Tomasz Politański Design for its logo and associated graphic design work.

http://tomaszpolitanski.com

Publisher & sponsorship

IFOSS L. Rev. is published by its Editorial Committee, with financial and administrative assistance from NLnet Foundation and Mozilla Foundation. Please note that neither NLnet Foundation nor Mozilla Foundation accept correspondence on behalf of this publication. All correspondence should be directed to the Editorial Committee via email (see below).

Editorial policies

IFOSS L. Rev. accepts articles for publication from qualified personnel based on the criteria available to view on its web site (**http://www.ifosslr.org**). Submissions are welcome from all, and your business. Authors are strongly encouraged to read the style and content guidelines available on the web site. The review operates an anonymous peer review system for articles as appropriate, and expects all authors to meet the highest standards of scholarship and integrity.

Bibliographic information

The authors explicitly encourage libraries, archives and educational institutions to hold copies of IFOSS L. Rev. in their collections, in electronic and/or printed form. All users are advised that articles may occasionally be updated after publication. Linking back to original copies on the IFOSSL. Rev. web site, where authoritative versions are archived, is strongly recommended. Please contact the Editorial Coordinators for further information on best practices. It participates in the CrossRef system.

ISSN: 1877-6922

Publication schedule

IFOSS L. Rev. is published biannually. Submissions for publica-tion are welcome at any time, but publication deadlines exist for each issue. For the latest information on papers sought and deadlines for submission, please consult the IFOSS L. Rev. website or contact the Editorial Coordinators at (**admin@ifosslr.org**)

Editorial

Daniel M. German,[a]

(a) Associate Professor, Computer Science, University of Victoria, editor of this Review;

DOI: 10.5033/ifosslr.v5i1.83

Abstract

Editorial for Issue 1, Volume 5

Keywords

Editorial

In Oct 1985, a few months after publishing his seminal GNU Manifesto, Richard Stallman founded the non-profit Free Software Foundation. It is likely that he had realized that he needed the legal framework of an organization that could own and administer the assets that he and his collaborators were creating, and that would manage the financial aspects of the project.

The communities of contributors of large Free and Open Source Software (FOSS) projects have found themselves in a position similar to that of Richard Stallman. As he did, they have created non-profit foundations to help them achieve their goals. The GNOME Foundation, the Linux Foundation, the Mozilla Foundation, the Apache Foundation, the Document Foundation, the Open Street Foundation and many others have been established by their corresponding communities to help manage their projects. In their paper titled "The Rise and Evolution of the Open Source Software Foundation," Paula Hunter and Stephen Walli explore the reasons behind the creation of FOSS foundations from the legal, business, and technical point of view. They explain that, as projects grow in size and attract commercial interest, foundations are not only needed to manage the potentially conflicting interests of its participants, but to administer its assets and to help create a structure that supports and fosters the further development of its software.

Today, reuse is a very important aspect of software engineering. It is very rare to see a software product that has been developed completely from scratch. Instead, software is structured in layers, such that software systems "build" on the features of others. From a technical point of view, reuse is facilitated by well defined interfaces, typically known as Application Programming Interfaces or APIs. APIs become the protocol that defines how a software product (a library, an operating system, a programming language, a plugin, a web service, etc.) expects to interact with another one.

When the API that governs the interactions between two software systems is well-defined, either one of them can (at least in theory) be replaced by an alternative implementation that has the same API and equivalent functionality.

For this reason, it is important to know if a given API is copyright-able. If it is, then anybody wanting to implement a software system that implements such API would require a license from the API copyright owner. In this scenario, the API owner would be in the position to control the market of products that implement such API, potentially restricting competition. Perhaps more important is the question of whether APIs should be copyright-able at all. In a span of few months, two legal cases, one in Europe and one in the United States address this issue in a similar manner.

In Europe, *SAS Institute Inc. v. World Programming Ltd*, C-406/10[1] involved the SAS language. SAS is a programming language for data processing and statistical analysis. WPL created an implementation of this language without approval from SAS Institute, and without access to the source code of SAS. SAS Institute sued them for infringement of copyright.

In the United States, *Oracle America, Inc. v. Google, Inc.*[2] revolved around Java, the popular programming language, developed by Oracle America. Google had created, as part of its Android mobile platform, a partial implementation of the runtime library of Java – without the authorization of Oracle America, its copyright owner. Oracle America argued that Google had violated its copyrights, and sued.

Walter van Holst's article "Less may be more: copyleft, -right and the case law on APIs on both sides of the Atlantic" discusses both cases within the context of the licenses of the Free Software Foundation. In particular, he argues that if APIs are not copyright-able, then linking (whether dynamic or static) could be considered mere aggregation, and the General Public License could be interpreted to be equivalent to the Lesser General Public License (LGPL) and therefore weakened.

License proliferation is another issue that can potentially hurt software reuse in FOSS. One would expect that new FOSS licenses are created because their authors believe that current licenses do not satisfy their legal requirements. In the article "Copyleft: A Close Reading of the Lisp LGPL" Eli Greenbaum analyses the Lisp Lesser General Public License (LLGPL), a license derived from the LGPL version 2.1. Greenbaum describes the rational behind its creation, and argues that the LLGPL is redundant and that its drafters would have achieved the same goals using the LGPL instead.

About the author

Daniel M. German *is Associate Professor, Computer Science, University of Victoria. He completed his Ph.D. in computer science at the University of Waterloo. His work spans the areas of software evolution, software engineering of free and open source software and the impact of intellectual property in software engineering.*

1 See SAS Institute Inc. v. World Programming Ltd. Case C-406/10 Judgement of the Court (Grand Chamber) of 2 May 2012. Available at http://curia.europa.eu/juris/liste.jsf?num=C-406/10

2 See Oracle of America v. Google Inc. Case No. C 10-03561 WHA. Order RE Copyrightability of Certain Replicated Elements of Java Application Programming Interface (N.D. Cal. July 22, 2011).

Less may be more: copyleft, -right and the case law on APIs on both sides of the Atlantic

Walter H. van Holst

Senior IT-legal consultant at Mitopics, The Netherlands

(with thanks to the whole of the FTF-legal mailinglist for contributing information and cases that were essential for this article)

DOI: 10.5033/ifosslr.v5i1.72

Abstract
Like any relatively young area of law, copyright on software is surrounded by some legal uncertainty. Even more so in the context of copyleft open source licenses, since these licenses in some respects aim for goals that are the opposite of 'regular' software copyright law. This article provides an analysis of the reciprocal effect of the GPL-family of copyleft software licenses (the GPL, LGPL and the AGPL) from a mostly copyright perspective as well as an analysis of the extent to which the SAS/WPL case affects this family of copyleft software licenses. In this article the extent to which the GPL and AGPL reciprocity clauses have a wider effect than those of the LGPL is questioned, while both the SAS/WPL jurisprudence and the Oracle vs Google case seem to affirm the LGPL's “dynamic linking” criterium. The net result is that the GPL may not be able to be more copyleft than the LGPL.

Keywords
Law; information technology; Free and Open Source Software; case law; copyleft, copyright; reciprocity effect; exhaustion; derivation; compilation

Introduction

A recurring issue surrounding copyleft licenses is the question at which point the reciprocal effect (which has been called the "viral effect" of these licenses by some) ceases to exist. There has hardly been an issue of IFOSSLR that did not touch this particular issue. Since the most important family of copyleft licenses is the GPL family of licenses and the GPL (both version 2 and 3) states that the rightsholder's authority solely derives from copyright law, the boundaries of copyright on software are paramount in order to be able to answer this question. To some extent, the boundaries of software copyright have been addressed in recent case law, both in the European Union (SAS Institute vs WPL Ltd) and in the United States (Oracle vs Google). The subject of this article is the interplay between the aforementioned copyleft provisions in the GPL-family of free software licenses and these fairly recent developments in jurisprudence. The conclusion is that the net difference between the LGPL and the GPL may be a lot less than intended by their drafters.

In this article the issue at hand, the scope of the reciprocal effect of the GPL-family of licenses, is addressed through an analysis of the applicable rules as supplied by said family and copyright law on software with a focus on reciprocity in case of inclusion (and no other adaptation) of (L)GPL software in other software. Although the prism through which this is looked at is primarily the EU Software Directive (and more precisely the Dutch transposition of it into law as well as wider Dutch copyright jurisprudence), other jurisdictions, notably the US, will be taken into account by an analysis of the case law mentioned above.

Legal framework as provided by the GPL family

Roles of the GPL family of licenses

It is important to understand the GPL-family as dual-purpose licenses. They provide both an end-user license and a distribution license. The end-user license is relatively simple, the core of it is included in art. 2 GPL v3, which among other things says "This License explicitly affirms your unlimited permission to run the unmodified Program". The distribution license is where the pitfalls lie, but again from a relatively uncomplicated basis in section 4 GPL v3:

> *"You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program."*

And also a delineation of its scope in section 5 GPL v3 (see also GPL v2, section 2, final paragraph):

> *"A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate*

does not cause this License to apply to the other parts of the aggregate."

The complexity starts here, because the GPL speaks about not being "by their nature extensions of the covered work". In other words: as long as no derivation takes place. And then it becomes relevant what defines derivation, does the GPL family of licenses take precedence here or does copyright law?

Bare licenses based on copyright law

Section 0 of GPL v2 is rather explicit about its tie-in with copyright:

> *"Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does. "*

Somewhat more implicit, but with seemingly the same meaning is Section 0 of GPL v3; several core concepts are defined within that core concept by using copyright (or related rights such as semiconductor masks) as the explicit reference for their scope.

This is no surprise since the Free Software Foundation (FSF) has always claimed that the GPL-family of licenses is a set of so-called bare licenses,[1] meaning that they should be interpreted solely through the prism of copyright law and not contract law.[2] This distinction is mostly relevant in common law jurisdictions since in most civil law jurisdictions the exonerations of liability in the GPL-family of licenses are most likely to be treated as negative obligations of the licensee, which automatically makes the license a bilateral contract. The upside of civil law jurisdictions is that generally speaking the licensor will not be deprived from enforcement options based on copyright infringement by the mere fact that there is a contractual relationship with the licensor. Basically, copyright infringement overrides the safeguards that a liable party which is in breach of contract could otherwise rely on.

The net result of all this is that in order to find the scope of what constitutes a derivative work under the GPL-family of licenses we will have to focus on software copyright and as far as that does not provide answers, to copyright law in general. Neither the EU Software Directive nor art. 117 of the US Copyright Act of 1976 (USC) contain specific provisions about derivation of software. Also literature on this subject is relatively scarce, with the notable exception of Pamela Samuelson's impressive analysis of derivation under US copyright law.[3]

So we have to turn to 'classical' copyright on the subject of what constitutes a derivative work under copyright law. Article 2 sub 3 of the Berne Convention defines derivative works as:

> *"Translations, adaptations, arrangements of music and other alterations of a literary*

1 Moglen, E. (2001), Enforcing the GNU GPL, http://www.gnu.org/philosophy/enforcing-gpl.en.html
For a similar analysis of how the GPL works see also Stoltz, Mitchell L. (2005), The Penguin Paradox: How the scope of derivative works affects the effectiveness of the GNU GPL, in Boston University Law Review, Vol 85, nr. 5, December 2005, pp. 1440-1477.

2 See for example: Henley, Mark (2009) 'Jacobsen v Katzer and Kamind Associates – an English legal perspective', IFOSS L. Rev., 1(1), pp 41 – 44, and Rosen, Lawrence (2009) 'Bad facts make good law: the Jacobsen case and Open Source', IFOSS L. Rev., 1(1), pp 27 – 32.

3 Samuelson, P. (2012), The Quest for a Sound Conception of Copyright's Derivative Work Right (August 29, 2012). Georgetown Law Journal, Forthcoming; UC Berkeley Public Law Research Paper No. 2138479. Available at SSRN: http://ssrn.com/abstract=2138479 or http://dx.doi.org/10.2139/ssrn.2138479

> *or artistic work shall be protected as original works without prejudice to the copyright in the original work."*

Copyright laws in the various signatory countries of the Berne Convention tend to be variations of that theme, examples are art. 101 USC, art. 13 of the Dutch Auteurswet (Aw), art. 23 of the German Urhebegesetzbuch (UHG) and art. 21 of the UK Copyright, Designs and Patents Act 1988. The operative term in all these legislative terms is 'adaptation' in the sense of alterations made to the work.

This is not wholly reflected in section 5 of GPL v3 which starts with:

> *"You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4..."*

and then continues with a series of conditions, among them the famous reciprocity clause:

> *"c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it."*

To clarify this further, at the very bottom of the GPL v3 the following note can be found:

> *"The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>. "*

Section 2 of the GPL v2 and another note at the end of the GPL v2 contain very similar language.

It is therefore not unreasonable to conclude that the GPL family of licenses generally assumes that the notion of derivative work extends beyond actual transformations or adaptations to include the use of API calls to libraries. This is underlined further by the way the LGPL (both v2 and v3) treat this, although not entirely consistently. In the LGPL this is treated as a "Combined work" per the definitions of section 0 LGPL v3 and to which notably the rules of section 4 LGPL v3 apply. These state the requirement of a 'suitable linking mechanism', which is a fascinating read on its own and will be discussed shortly. The reasoning seems to be based on the idea that a creation of dependencies on L(GPL) software through library calls is use of the software beyond the permitted use and distribution of Sections 2 and 4 of the GPL, which gets us to the extent such library calls are indeed covered by copyright as protected acts.

Analysis and application to libraries

Linking mechanisms

Before getting into detail on the copyright aspects of library calls, a minimal explanation of library calls and linking mechanisms is in order. A lot of software, especially application software, relies on function libraries that are typically employed by calling the Application Programming Interface (API) of those libraries. To use a real-world analogy, it is not unlike using Legos as an underlying foundation for a top layer that is typically written by the author(s) of the software. There are typically two ways of providing the foundations. The first one is so-called static linking, which works like incorporating the building blocks permanently in the application. The other way is to give the operating system a bill of materials stating the building blocks needed (including version information) and to have them assembled at run-time. This is called dynamic linking. As a result multiple programmes can share common building blocks which save both storage and memory space. Again, this is a very minimal explanation, for a truly thorough overview of the amount of copying and remixing of software that goes on during the normal usage of a contemporary computer that is accessible to a lawyer, I refer to Determann.[4]

It can be argued however that dynamic linking is less of an adaptation than for example a collection of poems, or the incorporation of graphical materials, musical scores or photographs in a text, which usually are not considered as adaptations (which in certain jurisdictions would be treated as collective works). The actual act of setting up the necessary references to successfully make library 'calls' is done at run-time by the operating system, when loading the calling programme into memory, not when distributing the programme that is dependent on the libraries.

It should also be noted that similar mechanisms are employed in the case of contemporary multi-platform language frameworks such as Java, Dalvik and .NET which all use virtual machines as target platforms, but in practice often rely on Just-In-Time (JIT) compilers that ultimately function very similarly to that of traditional compilers with the difference being that they are invoked on the fly during startup of a programme written in a higher or intermediate level language. Even more dynamic are programmes written in so-called dynamic languages such as Python, PHP, JavaScript (ECMA script) and Ruby, but ultimately the lower level mechanisms are not dissimilar to those described above.

Transformation and derivation in case law

Neither with static nor with dynamic linking there is much, if any, transformation of the work at a technical level, although practically speaking it will be very difficult to separate the building blocks from a statically linked executable. When looking at the rare cases about derivative works these tend to concentrate on the edges of exhaustion of copyright (also known as the first sale doctrine). They also appear to be toss-ups between being qualified as derivative works (and therefore infringing) or as mere exhaustions of existing copies. An example in the US is **Lee v. A.R.T. Company**, 125 F.3d 580 (7th Cir. 1997).[5] One Deck the Walls store sold note cards and small lithographs created by Lee to A.R.T. Company, which mounted the works on ceramic tiles (covering the art with transparent epoxy resin in the process) and resold the tiles. Lee was of the opinion that this constituted an adaptation and therefore a derivative work, while A.R.T. Company claimed that this was a case of copyright exhaustion. The 7th Circuit Court of Appeals upheld the District Court's view that this was copyright exhaustion whereas in a similar case, **Mirage Editions, Inc. v. Albuquerque A.R.T. Company**, 856 F.2d 1341 (9th Cir. 1988)[6] the 9th Circuit Court of Appeals came to an opposite decision. In this case the 9th Circuit wrote "We conclude, though, that appellant has certainly recast or transformed the individual images by incorporating them into its tile-preparing process.", thereby referring to art. 101 USC which describes recasting

4 Determann, L. (2006), Dangerous Liaisons – Software Combinations As Derivative Works? Distribution, Installation And Execution Of Linked Programs Under Copyright Law, Commercial Licenses And The GPL, 21 Berkeley Technology Law Journal 1421 (2006)

5 A commentary on this decision can be found at http://www.law.cornell.edu/copyright/cases/125_F3d_580.htm

6 The decision can be found at https://bulk.resource.org/courts.gov/c/F2/856/856.F2d.1341.87-6465.html

or transforming as one of the ways a work can be derived.

Closer to the copyright in software is a series of video game console related cases: **Sega Enterprises v. Accolade**, 977 F.2d 1510 (9th Cir. 1992),[7] **Sony Computer Entertainment, Inc. v. Connectix Corp.**, 203 F.3d 596 (9th Cir. 2000)[8] and **Lewis Galoob Toys, Inc. v. Nintendo of America**, Inc. 964 F.2d 965 (9th Cir. 1992).[9] Surprisingly enough, in none of the three cases it was claimed that interfaces are copyright protected material. This despite the fact that in all three cases use or even reimplementation of APIs lied at the heart of the matter. Stolz rightfully notes in his analysis that “the cases strongly imply that any parts of a program that must necessarily be copied in order to create a compatible module are not protected by copyright, denying copyright holders one of their key tools for controlling unauthorized linking”.[10] Very striking is that in Sega the Ninth Circuit Court held that under the circumstances of the case the functional need to use some of Sega's code to use the functionality of its cartridge interface was grounds for a fair use defence, despite the literal copying and distribution of code from Sega involved. In Connectix the full reimplementation of Sony's PlayStation game console in software was not even the heart of the dispute; the core arguments were about to what extent intermediate versions of Connectix's product had been a derivative work of Sony's software as embedded in the PlayStation. And here the Ninth Circuit ruled that since the end result was free of Sony's code, there was only indirect derivation. From a pure derivative works perspective this jurisprudence is mostly tangentially relevant, but does not explicitly answer the question. Stolz also describes related cases[11] in which derivation was judged to exist, but he clearly thinks these cases do no longer provide much precedent after Connectix.

For case law from this side of the Atlantic we stay in the realm of repurposing popular art, since there is no jurisprudence equivalent to the aforementioned game console cases in the United States. For example in **Rien Poortvliet**[12] the Dutch High Court ruled that cutting up a calendar with authorised reproductions of the artist Rien Poortvliet and selling the pieces after having glued them to cardboard constituted an infringing derivation. One of the reasons the High Court found these infringing was that the author's partial transfer of copyright only had calendars within its scope and never was intended to encompass other markets than calendars. With the interesting consequence that a contractual limitation was deemed relevant for third parties that had no way of knowing about that contractual limitation. Equally similar to the US jurisprudence were the recent **Pictoright/Allposters**[13] cases in the Netherlands in which for reasons very similar to Mirage vs Alberquerque it was decided that the sale of art posters transferred on canvas surface constituted sale of derivative works, not exhaustion, and therefore infringement. Another case of creative reuse of existing artwork was the German **Flachmembranlautsprecher** case[14] in which electrostatic loudspeakers had been fitted with art posters on their surface. The Upper State Court of Hamburg followed a reasoning that the artwork still performed a very similar function on the electrostatic loudspeakers, namely wall decoration, as on the original medium (the posters) and that it therefore was not being used outside the economic scope for which it had been licensed to by the poster publisher.

Applying the foregoing jurisprudence, the inclusion of libraries in other code through the

7 Retrieved from: https://bulk.resource.org/courts.gov/c/F2/977/977.F2d.1510.92-15655.html
8 Retrieved from: https://bulk.resource.org/courts.gov/c/F3/203/203.F3d.596.99-15852.html
9 Retrieved from: https://bulk.resource.org/courts.gov/c/F2/964/964.F2d.965.91-16205.html
10 Stolz, p. 1458.
11 Worlds Of Wonder v. Veritel Learning Systems, 658 F.Supp. 351 (1986) and Micro Star v. FormGen Inc. 154 F.3d 1107 (9th Cir. 1998), which narrowed Galoob down considerably.
12 HR 19 januari 1979, NJ 1979, 412 m.nt. LWH; AMR 1979, p. 50 m.nt. JHS; AA 1980, p. 311 m.nt. Cohen Jehoram.
13 Rb Roermond, 22 september 2010, Pictoright v Art & Allposters, overturned by Hof Den Bosch, 3 januari 2012, HO 200.079.664, LJN: BV0773 which can be found at http://www.rechtspraak.nl/ljn.asp?ljn=BV0773
14 OLG Hamburg - Urteil vom 10.10.2001 (5 U 86/01) - DRsp Nr. 2003/6820

mechanism of static linking as described above may be qualified as derivative works on either side of the Atlantic, even though one could argue that the linked code has not been adapted otherwise. It should also be added that different jurisdictions already have differing outcomes when it comes to relatively simple cases such as repurposed art publications, so that this already is a very grey area in copyright law.

The analogy with “recasting” as was made in the Mirage decision becomes difficult to hold onto when applied to dynamic linking. By its very nature dynamic linking only takes place at runtime, so the “recasting” only takes place at the end-user's machine, not during distribution. The distribution itself may be accompanied with the dependent application, but not necessarily so. Extending the prism of “recasting” to dynamic linking of libraries would make a lot of the dependencies of applications on operating systems a reason to assume that such applications would be a derivative of the operating system. For example a great deal of non-kernel API calls tend to employ dynamic linking mechanisms. Typical examples are graphical user interface (GUI) elements and other standard components of contemporary operating systems. A stronger argument may be the economic reasoning taken by European courts in the cases quoted above because they do focus on the market as intended by the author, but this still assumes that the API itself is subject to copyright.

This was in essence one of the questions raised in both the **SAS/WPL**[15] (in the EU) and **Oracle/Google**[16] (in the USA) cases. The dust has not settled on either case yet and in the case of Oracle/Google an appeal has been filed, so especially regarding the situation in the USA this analysis is somewhat preliminary.

In SAS/WPL one of the main questions was whether a reimplementation of a programming language in a new piece of software would be an infringement of the copyright of the original piece. This is relevant in the context of library calls because the keywords and syntax of a programming language in themselves do constitute a (high level) API, but as Vezzoso[17] rightly points out, this decision does not expressly concern APIs. The European Court of Justice (ECJ) built further on its earlier **Bezpečnostní softwarová asociace**[18] decision and ruled that this matter falls outside the scope of the Software Directive (91/250 EC), but in such a way that it does not explicitly place APIs outside the scope of general copyright:

> *“Consequently, the answer to Questions 1 to 5 is that Article 1(2) of Directive 91/250 must be interpreted as meaning that neither the functionality of a computer program nor the programming language and the format of data files used in a computer program in order to exploit certain of its functions constitute a form of expression of that program and, **as such**, are not protected by copyright in computer programs for the purposes of that directive.” (emphasis mine)*

The strange reminiscent of the European Patent Convention, use of 'as such' implies that under certain (however unspecified) circumstances functionality or a programming language (which are a species of API) may be protected by copyright in computer programs for the purposes of Software Directive (91/250 EC). It also does not exclude the possibility that an API may be covered by general copyright law at all, but given the technical nature of APIs they by and large

15 SAS Institute Inc. vs World Programming Ltd, ECJ May 2nd, 2012, C-406/10, retrieved from http://curia.europa.eu/juris/document/document.jsf?text=&docid=122362&pageIndex=0&doclang=EN&mode=req&dir=&occ=first&part=1&cid=972439

16 US District Court for the Northern District of California, No. C 10-03561 WHA

17 Vezzoso, S. (2012), Copyright, Interfaces, and a Possible Atlantic Divide, in: JIPITEC no 3, p. 153, para. 1.

18 Bezpečnostní softwarová asociace, ECJ December 22nd, 2010, C-393/09, retrieved from http://curia.europa.eu/juris/document/document.jsf?text=&docid=83458&pageIndex=0&doclang=EN&mode=lst&dir=&occ=first&part=1&cid=713053

are unlikely to fall within the scope of general copyright law. This is crucial for the question where the reciprocal nature of the GPL ends since the GPL, as earlier mentioned, relies solely on copyright law.

For the purpose of the question where the reciprocal nature of the GPL ends when it comes to (dynamic) linking of libraries, the ECJ's implicit caveat about general copyright law is of lesser importance than for interoperability matters. Because even if an API falls inside the scope of copyright, it generally is no longer constrained by the intended use of articles 4 and 5 of the Software Directive (91/250 EC), unless the unspecified circumstances of the 'as is' are in play. This also means that the exceptions of classical copyright can be invoked and may overrule the GPL as far as the API of a (L)GPL-covered piece of software is concerned. It even opens the door for potential corner-case scenarios in which minor cases of static linking (so the inclusion of parts of a GPL-covered library into a calling program) may possibly fall outside the scope of the reciprocity clauses of the GPL-family of licenses. It also puts the AGPL's reciprocity clause which extends distribution to the provisioning of online services into a new light as far as its applicability to APIs for web-services is concerned.

Although admittedly a lower court, so not necessarily setting a precedent for the whole of the US yet, the US District Court of Northern California went a significant step further than the ECJ in Oracle vs Google when confronted with the question whether an API is covered by copyright. The court answered it with a rather resounding no:

> *“This order holds that, under the Copyright Act, no matter how creative or imaginative a Java method specification may be, the entire world is entitled to use the same method specification (inputs, outputs, parameters) so long as the line-by-line implementations are different.”*

The conclusion of all of this is that if the Java API falls outside the scope of copyright protection and if we can extend this reasoning to any library API, the particular use of a library API without the full inclusion of the library cannot possibly constitute the type of adaptation that is covered by art. 117 USC or equivalent laws in European jurisdictions. In the European context an API may possibly fall within the scope of copyright protection, although likely a very limited protection due to the highly technical constraints within which APIs typically are designed. Arguments based on the intended use of the GPL-covered library cannot hold up either because a) in the case of dynamically linked libraries that use is by the end-user, not the publisher of the library-dependent programme, and b) they are self-contradicting with both sections 2 and 5 GPL v3.

It should be noted that this analysis does not deviate from Stolz's earlier analysis of the GPL v2 based on earlier case law that was more implicit on the question of derivation in software.

Conclusion

In order to establish at which point the reciprocal nature of the GPL in case of inclusion (and no other adaptation) of (L)GPL software in other software should take place, I have assessed both the GPL and the LGPL in their role as distribution licenses. In order to establish the precedence of the GPL-family of licenses over copyright, I have established that the latter takes precedent since they are designed as bare licenses. This means that they cannot redefine what constitutes a derivative work and can only cover that what is governed by copyright law as far as the question when the GPL should be applied to computer programmes that are dependent on (L)GPL libraries is concerned. As a consequence, the question to what extent inclusion of a covered library constitutes the creation of a derivative work beyond "mere aggregation" becomes relevant. When analysing the typical mechanisms for inclusion, both static and dynamic linking, it must be concluded that the closest analogies to dynamic linking in jurisprudence are in a grey zone. Furthermore, these analogies are of limited use since the mechanisms of dynamic linking are common practice in most contemporary computer systems and are generally understood not to constitute derivation. When taking the most recent jurisprudence on software APIs into account, one can argue that the LGPL is not really the Lesser GPL, but that the GPL is based on a by now outdated understanding of software copyright and effectively becomes equal to the LGPL. In light of the fact the open source communities tend to think of the currently most popular sets of licenses as a continuum from permissive (Apache 2.0) to very far copyleft (AGPL 3.0), the conclusion that this continuum does not stretch much further in the copyleft spectrum than LGPL is not a happy one. It means that there is a serious disconnect between the expectations developers may have from their chosen license in the GPL family and the legal reality. The intent of these developers is not necessarily reflected in the effects of their chosen licenses, which is rather unfortunate.

About the author

Walter van Holst *is a senior legal consultant with Mitopics (http://www.mitopics.nl)*

Lisping Copyleft: A Close Reading of the Lisp LGPL

Eli Greenbaum [a]

(a) Attorney, Yigal Arnon & Co. Jerusalem

DOI: 10.5033/ifosslr.v5i1.75

Abstract:

The idioms of both the General Public License (the "**GPL**") and the Lesser General Public License (the "**LGPL**") seem to be grounded in the C programming language. This article analyses the Lisp Lesser General Public License (colloquially and here referred to as the "**LLGPL**"), a specific attempt to apply the LGPL to a language with a programming paradigm and method of building and distributing programs that traditionally differs substantially from the approach of C. In addition, this article attempts to understand whether the LLGPL actually succeeds in its stated goal of translating the LGPL to the Lisp context or whether the LLGPL changes the requirements and philosophical moorings of the LGPL.

Keywords:
Law; information technology; Free and Open Source Software; copyleft, copyright; derivation; compilation; Lisp; LGPL;

Introduction

The idioms of both the General Public License (the "**GPL**") and the Lesser General Public License (the "**LGPL**")[1] seem to be grounded in the C programming language. The licenses refer to "compiling", "linking" and "header files", features of the C programming languages which may not be present in other languages that are not traditionally compiled. Similarly, the licenses do not expressly include provisions relating to features of object-oriented programming languages.[2] Do the GNU licenses work as intended when applied in these other contexts? [3] This article analyses the Lisp Lesser General Public License (colloquially and here referred to as the "**LLGPL**"), a specific attempt to apply the LGPL to a language with a programming paradigm and method of

1 The LLGPL license is drafted as a preamble to version 2.1 of the LGPL. As such, in this article, unless states otherwise references to the GPL and LGPL are references to version 2.0 of the GPL and version 2.1 of the LGPL.

2 In contrast, version 3.0 of the LGPL does relate to features of object oriented languages. For example, the definition of "Application" in that license discusses the effect of defining a subclass of a class defined by the Library.

3 The Free Software Foundation has strongly asserted that the LGPL may be applied to all known programming languages. *See* David Turner, The LGPL and Java, *available at* http://www.gnu.org/licenses/lgpl-java.html (stating that "FSF's position has remained constant throughout: the LGPL works as intended with all known programming languages, including Java.").

building and distributing programs that traditionally differs from the approach of C.[4]

Lisp is one of the oldest programming languages still in use. Lisp was invented in 1958 by John McCarthy at the Massachusetts Institute of Technology. The language was first implemented when one of McCarthy's graduate students hand-compiled the Lisp *eval* function into machine code, and created the first Lisp interpreter. Following in this history, while implementations of Lisp can allow for the compilation and distribution of executables, Lisp was traditionally developed and distributed as an interpreted rather than a compiled language. Lisp was closely connected to research in the field of artificial intelligence, and the popularity of the language declined in the late 1980s together with interest in that field. Nevertheless, Lisp seems to have enjoyed somewhat of a resurgence in recent years, and currently there are several open source and commercial implementations of the language.

Open source programs are not frequently written in Lisp.[5] Nevertheless, certain features of Lisp have inspired a broader family of "dynamic languages" that can be considered to include popular languages such as PHP or Python. As with Lisp, for example, those languages are typically interpreted rather than compiled into executables. As such, programs written in those languages will also generally require the distribution of an interpreter together with the application. To the extent the LLGPL's claim that the GNU licenses are not appropriate for Lisp is justified, the suitability of the GNU licenses for these other languages will also be implicated.

This article presents a close reading of the LLGPL license. In analysing the license, this article attempts to understand whether, as the LLGPL claims, another document is necessary to apply the LGPL to the Lisp context. In addition, this article attempts to understand whether the LLGPL succeeds in its stated goal of translating the LGPL to the Lisp context or whether, in making the transition, the LLGPL moves away from the requirements and philosophical underpinnings of the LGPL. Before concluding, this article briefly discusses some issues raised by Lisp that were not expressly addressed by the LLGPL.[6]

History and Philosophy of the Lisp LGPL

The LLGPL was authored by Franz, Inc. ("**Franz**"), a leading commercial Lisp vendor based in California. Franz is the corporate developer of "Allegro Common Lisp", one of several commercial implementations of the "ANSI Common Lisp" standard.[7] The ANSI Common Lisp

4 In a somewhat ironic twist, the history of the GNU licenses began with Richard Stallman's distribution of Emacs, a text-editing program written in Lisp. Stallman initially distributed Emacs under the Emacs General Public License, out of which grew the first version of the General Public License. For an early history of the GNU licenses, see Chapter 2 of Glyn Moody, Rebel Code (2002).

5 According to Black Duck, Lisp is not one of the top fifteen languages used in open source projects. *See* http://www.blackducksoftware.com/osrc/data/projects/. C is the most popular language, used in 44.95% of releases of open source projects. For a not-up-to date list of some commercial software projects in Lisp, *see* http://www.pchristensen.com/blog/lisp-companies/.

6 Aside from the LLGPL, there are a number of other licenses that have been drafted to apply to specific programming languages. For example, PHP is distributed under a permissive license similar to the BSD. *See* http://www.php.net/license/index.php#code-lic. Python is also distributed under a permissive license. *See* http://docs.python.org/2/license.html. These licenses are generic and do not have any technical provisions that apply to features of specific languages. A number of other licenses contain provisions expressly adapted for particular programming languages. For example, the GNAT Modified General Public License is a version of the GPL which has been adapted for the "generic" feature of the Ada programming language. *See* http://libre.adacore.com/tools/gnat-gpl-edition/faq/. In addition, the Falcon Programming Language License is "specifically designed around the concept of an open source scripting language engine." See http://www.falconpl.org/index.ftd?page_id=licensing. An analysis of these latter two licenses is beyond the scope of this article.

7 Commercial implementations of Lisp also includes LispWorks. Open source implementations of Lisp include Steel Bank Common Lisp, which is licensed under BSD-style licenses and also includes code in the public domain (*See*

standard was developed in the early 1980s as an attempt to unify the several dialects of the language.[8] Franz initially began distributing Allegro Common Lisp in 1986, and authored the LLGPL in 2000. Franz has shown a commitment to the open source development of software, and has licensed a number of open source software projects under the terms of the LLGPL.[9] Unfortunately, there is a dearth of commentary regarding the interpretation of the LLGPL. As such, and as the provisions of the license are not always completely clear, the application of the LLGPL to software may not always be completely straightforward.

The LLGPL is a short document, consisting of five not-lengthy paragraphs, and by its terms is intended to be read as a "preamble" to the LGPL.[10] Generally, the LGPL permits proprietary applications to be combined and distributed with LGPL-licensed libraries, and does not require that the source code of the proprietary application be disclosed. This is in contrast to the stronger "copyleft" requirements of the GPL, which generally requires that all works "based on" a GPL-licensed work also be distributed under the same license terms. The LLGPL is intended to adapt the weaker copyleft provisions of the LGPL to the Lisp setting. In the words of the first paragraph of the LLGPL, the "LGPL uses terminology more appropriate for a program written in C than one written in Lisp" and, as such, some "clarifications" are necessary to apply the LGPL in the Lisp context.

The first paragraph of the LLGPL implies that the application of the LLGPL results in licensing terms that are not very different than the LGPL itself, even though they have been translated to the Lisp context.[11] Even so, several provisions of the LLGPL seem to belie this understanding of the license. For example, the LLGPL provides that a "Lisp application may include the same set of Lisp objects as does a Library, but this does not mean that the application is necessarily a 'work based on the Library' it contains."[12] In contrast, the LGPL expressly provides that a work containing portions of the Library should be considered a derivative work of the Library under copyright law, and a "work based on the Library" under the LGPL."[13] The clause in the LLGPL seems to contradict express provisions of the LGPL. Unfortunately, the LLGPL does not explain the motivation for making this fundamental change in the terms of the LGPL.

Other clauses of the LLGPL also seem to diverge from the provisions of the LGPL. For example, the LLGPL provides that "[i]t is permitted to add proprietary source code to the Library, but it must be done in a way such that the Library will still run without that proprietary code present."[14] This seems to restrict a user's ability to modify the licensed work. Interpreting this sentence in light of the LGPL is quite difficult, as Section 2 of the LGPL expressly provides a user of the Library with the right to modify and copy the Library, without any requirement to ensure that it can still run without those modifications.[15] Again, the LLGPL does not describe the reasons for

http://www.sbcl.org/history.html), GNU Common Lisp, available under the LGPL (*See* http://savannah.gnu.org/projects/gcl), and GNU Clisp, available under the GPL (See http://www.clisp.org/).

8 See the history of Lisp at http://www.dreamsongs.com/Files/Hopl2.pdf

9 See http://opensource.franz.com/

10 The concept of presenting the LLGPL as a preamble to the GNU license seems to be inspired by the structure of the LGPL itself, which begins with a preamble that explains the goals of the license.

11 On its website, Franz itself provides a somewhat more ambiguous description of the LLGPL, stating that the document is a "new license" which is intended to take the special features of dynamic programming languages into consideration. See http://opensource.franz.com/

12 See the last sentence of the second paragraph of the LLGPL.

13 For example, the definition of "Library" in the LGPL provides that a "work based on the Library" includes a "work containing the Library or a portion of it. Similarly, section 5 of the LGPL provides that "linking a 'work that uses the Library' with the Library creates … a derivative of the Library (because it contains portions of the Library)."

14 See first sentence of the fourth paragraph of the LLGPL.

15 Indeed, one of the core freedoms advocated by the Free Software Foundation is the freedom to modify software. See http://www.gnu.org/philosophy/free-sw.html. The LGPL also evidences a similar concern that a modified Library should be able to operate even without the application that it is linked with. *See* Section 2(d) of the LGPL. Even so, that concern is with regard to the use of the licensed work when distributed to third parties, but the literal reading of the

adding this requirement to the provisions of the LGPL.

These examples demonstrate that it is difficult to reconcile certain provisions of the LLGPL with the original GNU license. Indeed, this article shows that the LLGPL does clarify certain provisions of the LGPL in the Lisp setting, but also substantially modifies the provisions of the original license. Unfortunately, the LLGPL is frequently not explicit regarding whether a specific provision should be seen as a "translation" to the Lisp context or as an intentional change in the licensing terms of the LGPL. Furthermore, the LLGPL does not always explain its motivation for making certain clarifications or changes, and this can make it difficult to interpret and apply the license.

Definitions and Redefinitions

Several provisions of the LLGPL seem to be motivated by an attempt to clarify the provisions of the LGPL in the Lisp setting. For example, the second paragraph of the LLGPL changes several definitions of the LGPL, such as the definitions of "library", "function" and "data", making them more amenable to the Lisp context. The second paragraph reads in full:

> *A "Library" in Lisp is a collection of Lisp functions, data and foreign modules. The form of the Library can be Lisp source code (for processing by an interpreter) or object code (usually the result of compilation of source code or built with some other mechanisms). Foreign modules are object code in a form that can be linked into a Lisp executable. When we speak of functions we do so in the most general way to include, in addition, methods and unnamed functions. Lisp "data" is also a general term that includes the data structures resulting from defining Lisp classes. A Lisp application may include the same set of Lisp objects as does a Library, but this does not mean that the application is necessarily a "work based on the Library" it contains.*

These revised and generalised definitions are to some extent useful in clarifying LGPL terminology for Lisp. At the same time, however, the revisions appear to focus on certain aspects of the technical distinctiveness of Lisp which would not seem to materially affect the interpretation of the LGPL.

For example, the first sentence of the second paragraph provides that "[a] 'Library' in Lisp is a collection of Lisp functions, data and foreign modules."[16] The purpose of this definition seems to be the subtle modification of the definition of a "library" (not capitalised)[17] in the LGPL, which provides that a library means "a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to

LLGPL provides no suggestion that it should be read as anything but a restriction of a user's right to modify the program, regardless of whether the modified version is distributed to third parties. In addition, the LGPL only requires that users make a "good faith effort" to ensure the operation of the modified library. In contrast, the LLGPL's requirement is formulated as an absolute requirement.

16 A "foreign module", according to the LLGPL, is "object code in a form that can be linked into a Lisp executable". Briefly, Lisp data types often differ from data types in other languages. As such, Lisp requires a "foreign function interface" in order to link code written in a different language. *See generally* Peter Seibel, Practical Common Lisp, 467 (2005). The existence of foreign modules and the need for a "foreign function interface" is not unique to Lisp. For example, the "Java Native Interface" enables a Java Virtual Machine to invoke (and be invoked by) the native code of libraries and applications written in other languages. The Perl XS interface also allows Perl to use C libraries. The Python extension module API allows the calling of library functions and system calls.

17 The LGPL contains a definition of "library" (not capitalised) and "Library" (capitalised). The former provides a generic description of a software library, while the latter refers to the specific work licensed under the LGPL. The need for the former generic definition of a software library is not clear in the document and, indeed, version 3 of the LGPL omits this generic definition. In version 3 of the LGPL, a "Library" simply means a "covered work governed by this License."

form executables." It seems that the authors of the LLGPL did not believe that this definition was completely appropriate for the Lisp context. First, Lisp programs are traditionally interpreted rather than compiled and, as such, the LLGPL definition omits the provision that libraries are intended to be "linked … to form executables." Second, the LLGPL definition adds that a library may include a foreign module.[18]

It is unclear whether these changes to the definition of "library" are necessary for the application of the LGPL to Lisp applications. First, the LLGPL's change in the definition of "library" does not broaden the application of the license, since in any event the LGPL expressly provides that the license may be applied to "any software library or other program". In other words, the application of the LGPL is not restricted to works that meet the LGPL definition of "library". Indeed, the definition of "Library" (capitalised) in the LGPL refers generically to "any software library or work which has been distributed under these terms." Second, it is in any event doubtful that a court would interpret the LGPL's definition of "library" with a level of specificity that would exclude similar linguistic structures of Lisp. For example, it is unlikely that the word "function" in the LGPL would be interpreted to exclude a "foreign module", since both terms essentially refer to software modules that provide a level of functionality.

The second paragraph continues with several other clarifications of the LGPL terminology for the Lisp context. First, the document provides that "functions" should be understood to include "methods" and "unnamed functions". These two terms refer to syntax that is not part of the C programming language. In brief explanation (with more to come later in this article): First, in Lisp, a "method" is the specific implementation of an abstract operation, where the generalised operation is referred to as a "generic function";[19] second, Lisp (as well as other programming languages) offers the opportunity to create "unnamed functions", a way of creating functions without actually providing the function with a defined name.[20] At the same time, however, it is difficult to see why it is necessary to clarify these points in order to apply the LGPL to Lisp programs. It is difficult to conceive of a legitimate legal claim that the word "function" in the LGPL should not naturally be extended to constructions (such as methods and unnamed functions) that act as functions even though they differ in their syntactic expression.

The next clause of the LLGPL also attempts to interpret the LGPL in the Lisp context, and provides that "Lisp 'data' is also a general term that includes the data structures resulting from defining Lisp classes". This provision seems to be an attempt to apply the LGPL to the abstract data types (such as "classes") that form part of an object-oriented language. Indeed, it is possible to interpret the term "data" in the LGPL as referring to "information" rather than "data structures that contain information" in the abstract sense of the word.[21] Nevertheless, as with the word "function", it is not likely that the term "data" in the LGPL would be interpreted with a level of specificity that would exclude appropriate and similar structures in the Lisp context.

In sum, it does not seem that the changes made to the definition of "library" by the LLGPL are

18 It is possible to opine that the LLGPL broadens the defined term "function" in order to include Lisp "macros" within that defined term. This possibility is not expressly acknowledged by the text of the LLGPL and, as such, the effect of the LLGPL on Lisp "macros" remains unclear. Lisp "macros" are further discussed below in Section "**Of Macros**".

19 Other object oriented programming languages (such as Java) also provide for "methods". In Java, however, methods are typically incorporated into the definition of a class, while Lisp methods are defined outside of a class and rather as part of "generic functions." The implications of these syntactical distinctions are beyond the scope of this article. *See generally* SEIBEL, *supra* note 16, at 191.

20 In Lisp, "unnamed" functions are typically referred to as "lambda" functions. Lambda functions are useful, among other things, for creating functions that can use the local variables of the environment in which they were created. *See generally*, SEIBEL, *supra* note 16, at 62-63. "Unnamed" functions are also supported by other "dynamic" languages such as Ruby, Javascript, Perl and Python.

21 For example, Section 2(d) of the LGPL refers to a "table of data", which seems to imply that word "data" is used to mean "information". On the other hand, Section 5 of the LGPL refers to "data structure layouts".

necessary for the application of the GNU license to Lisp. Indeed, it seems that the changes made by the LLGPL are grounded in an appreciation of the technical distinctiveness of Lisp rather than an analysis of whether these differences should change the interpretation of the LGPL or the application of copyright law.

What is a Derivative Work in Lisp?

As shown above, the second paragraph of the LLGPL aims only to generalise certain terminology of the LGPL. The third paragraph, however, seems to supersede several core provisions and principles of the LGPL. Indeed, as shown below, the third paragraph is best interpreted as an abrupt re-alignment of the thrust of the LGPL. Unfortunately, the LLGPL does not clarify the motivation for these changes. As such, it is not clear whether the provisions of the third paragraph are dictated by the technical aspects of Lisp or by philosophical differences with the LGPL.

The GNU licenses are built on the copyright law concept of the "derivative work." In very general outline, a derivative work incorporates and builds on a pre-existing copyrighted work. Under copyright law, one may generally not reproduce or distribute "derivative works" of a copyrighted work without an appropriate license. The GNU licenses leveraged this idea into the "copyleft": a license to modify and distribute an original copyrighted work, on the condition that any derivative works of the original work be distributed pursuant to specified license terms.[22] This idea sets the boundaries of the requirements of the GNU licenses and, generally, the GNU licenses are not intended to impose restrictions on works that are not "derivative works".[23]

The third paragraph of the LLGPL, however, seems to take a rather different approach. This section will individually examine each sentence of the paragraph, showing that the ideas underlying these provisions differ from the motivating principles of the LGPL. The third paragraph states in full:

The Library consists of everything in the distribution file set before any modifications are made to the files. If any of the functions or classes in the Library are redefined in other files, then those redefinitions ARE considered a work based on the Library. If additional methods are added to generic functions in the Library, those additional methods are NOT considered a work based on the Library. If Library classes are subclassed, these subclasses are NOT considered a work based on the Library. If the Library is modified to explicitly call other functions that are neither part of Lisp itself nor an available add-on module to Lisp, then the functions called by the modified Library ARE considered a work based on the Library. The goal is to ensure that the Library will compile and run without getting undefined function errors.

The first example in the third LLGPL paragraph provides that "[i]f any of the functions or classes in the Library are redefined in other files, then those redefinitions ARE considered a work based on the Library." In other words, if a LLGPL-licensed work contains a defined and named function, a licensee of the work may redefine that function in a separate and different file to provide for a

22 As per the explanation of the Free Software Foundation: "*To copyleft a program, we first state that it is copyrighted; then we add distribution terms, which are a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code, or any program derived from it, but only if the distribution terms are unchanged*." See "Free Software Foundation, What is Copyleft?", available at http://www.gnu.org/copyleft/copyleft.html

23 For example, The Free Software Foundation has stated that it considers the phrase "works based on the Program" in the GPL to be similar though perhaps not identical to the definition of a derivative work under copyright law. See Opinion of the Denationalization of Terminology, Free Software Foundation, *available at* http://gplv3.fsf.org/denationalization-dd2.html. Whether the actual provisions of the GNU licenses respect this boundary, or try to impose restrictions on works that are not derivative works under copyright law, has been the subject of much commentary. See LAWRENCE ROSEN, OPEN SOURCE LICENSING 119-128 (2004).

different operation – however, such redefinition will be considered a "work based on the Library".[24] It is difficult to see how the act of redefining an existing function should in itself create a derivative work under copyright law. Of course, to the extent any redefinition incorporates material from the original definition, the redefinition could be seen as a derivative of the original. The LLGPL, however, considers any redefinition of the original function to be a "work based on the library", regardless of whether it incorporates material from the original definition. This provision is not based on the understanding of copyright law but, as will be shown below, on the rather different principle of ensuring the functionality of the original licensed work.

A good illustration of the philosophy of the LLGPL is provided by the last clause of the third paragraph. That provision states that "[i]f the Library is modified to explicitly call other functions that are neither part of Lisp itself nor an available add-on module to Lisp, then the functions called by the modified Library ARE considered a work based on the Library." As noted earlier, Lisp allows users to call "foreign modules" written in a different programming language.[25] This clause provides that such called foreign modules will be deemed a "work based on the library" – in other words, a derivative work of the library which may only be distributed under the terms of the LLGPL.[26] This provision is at odds with the LGPL in several ways. First, subject to certain restrictions, the LGPL typically allows third party modules to be linked to the licensed work, without requiring that such third party modules themselves be licensed under the LGPL.[27] This LLGPL provision, on the other hand, provides that certain third party modules, even though they do not incorporate code of the licensed work, must also be licensed under the terms of the LLGPL. Second, the LGPL generally only imposes restrictions on modules to the extent they are compiled or linked with the original library, but this LLGPL provision seems to impose restrictions on such linked modules regardless of whether they are actually linked or compiled with the library. Third, the LLGPL provision relaxes the restriction in respect of functions that are "part of Lisp itself" or "an available add-on module to Lisp". There is no equivalent in the LGPL to the relaxing of restrictions solely in respect of modules written in a particular programming language.

What is the LLGPL's motivation for providing different requirements than the LGPL? The last clause of the third paragraph sets forth the underlying philosophy of these provisions, stating that: "[t]he goal is to ensure that the Library will compile and run without getting undefined function errors." In other words, these provisions of the LLGPL are not based on the copyright principles of the LGPL. Rather, they are motivated by the goal of ensuring that a modified licensed work "continues to compile and run."[28] This philosophy is reflected in the provisions of the LLGPL discussed above. These provisions state that certain redefinitions and foreign modules are subject to the full restrictions of the LLGPL, even though they would not be considered "derivative works" under ordinary circumstances.

24 Redefining an existing system function is permitted under the Common Lisp standard, though not generally recommended because of the unintended consequences that such redefinitions can generate. For example, the Allegro Common Lisp 8.2 documentation states that "Lisp permits already-defined functions to be redefined dynamically. However, redefining system-defined functions … is almost always a bad idea." *See* http://www.franz.com/support/documentation/7.0/doc/packages.htm. For a discussion of some problems associated with the redefinition of functions in Lisp, see SEIBEL, *supra* note 16, at 274-75.

25 *See supra* note 15.

26 An interesting question not expressly addressed by the LLGPL is whether a derivative of an LLGPL-licensed work must be distributed under the terms of the LLGPL itself or may rather be distributed under the terms of the LGPL. The LLGPL is not clear on this point.

27 It should be noted that the Allegro Common Lisp 8.2 documentation states that foreign functions are "linked" to a running Lisp process. *See* http://www.franz.com/support/documentation/7.0/doc/foreign-functions.htm#ff-intro-1. This is distinct from other Lisp code which is actually loaded into the memory of a running Lisp image rather than linked.

28 The LLGPL is not clear as to why applying the LGPL obligations to redefinitions and foreign functions ensures that they will continue to "compile and run". It is possible that the LLGPL believes that requiring the source code of these elements to be distributed will allow the modified library to be debugged.

Moving back to earlier provisions of the LLGPL, the second and third clauses of the third paragraph are also inspired by the same goals of ensuring functionality. The third clause provides that "[i]f Library classes are subclassed, these subclasses are not considered a work based on the Library." The question of whether subclassing creates a derivative work has been raised in other contexts. For example, in the GPL FAQs, the Free Software Foundation takes the position that subclassing creates a derivative work, without offering an explanation for that position.[29] Other commentators have taken different positions.[30] The LLGPL provides that subclasses will not be considered derivative works.[31] Again, the reasoning of the LLGPL seems to be based on whether the subclass could possibly interfere with the functionality of the original library. As simply adding the subclass would not interfere with the functionality of the original defined class, the LLGPL takes the position that the subclass should not be considered a "work based on the library".

The second clause takes the same approach, stating that "[i]f additional methods are added to generic functions in the Library, those additional methods are NOT considered a work based on the Library." Again and briefly, in Common Lisp, "methods" are various implementations of an abstract definition of a generic function in a variety of circumstances. For example, a generic function may state that it operates to draw shapes, without actually providing an implementation of that functionality. The specific methods of that generic function, however, provide the functionality for actually drawing a variety of shapes.[32] The question of whether adding additional methods to a generic function creates a derivative work is an interesting question, and beyond the scope of this article. However, it should be noted that again the approach of the LLGPL is not to ask whether the addition of methods to a generic function creates a derivative work, but rather to ask whether the modifications will preserve the functionality of the original library. Methods may be added or removed to a generic function without affecting the functionality of the generic function itself. As such, according to the LLGPL, the additional methods are not considered a work based on the library.

This section has shown that with regard to the question of what constitutes a "work based on the library," the LLGPL takes a very different approach than the original LGPL license. While the restrictions of the original license were based on an understanding of a "derivative work" under copyright law, the obligations of the LLGPL seek to ensure the functionality of the licensed program. In implementing these goals, the LLGL provides for very different requirements and obligations than the original LGPL.

Distribution

Another distinctive feature of Lisp – albeit a feature that has since been adopted by other languages – is the fact that Lisp programs are traditionally constructed within a dynamic run-time environment. Lisp programs may be developed incrementally by composing or loading program statements into the run-time environment, and the run-time environment will interpret or compile

29 See http://www.gnu.org/licenses/gpl-faq.html#OOPLang (stating that "[s]ubclassing is creating a derivative work. Therefore, the terms of the GPL affect the whole program where you create a subclass of a GPL'ed class."). The FSF takes the same position in its article "The LGPL and Java", where it states that " [i]nheritance creates derivative works in the same way as traditional linking". See http://www.gnu.org/licenses/lgpl-java.html. Version 3 of the LGPL expressly addresses the question of subclassing, *see* infra note 31.

30 *See, e.g.*, Derivative Works, http://www.law.washington.edu/lta/swp/law/derivative.html (arguing against the position of the Free Software Foundation regarding subclasses).

31 Unfortunately, the LLGPL does not actually clarify whether subclasses will be subject to the obligations imposed by the LGPL with regard to work linked with the licensed work. In contrast, Version 3 of the LGPL clarifies that "defining a subclass of a class defined the Library is deemed a mode of using an interface provided by the Library." As such, under Version 3 of the LGPL, a work that defines a subclass is subject to the usual LGPL obligations in respect of works that link with the licensed library. As such, it must be distributed under terms that "do not restrict modification […] and reverse engineering for debugging such modifications."

32 This example is taken from Seibel, *supra* note 16, chapter 6.

such forms as they are entered.[33] The Lisp run-time environment also impacts how programs are distributed. Unlike the C programming language, Lisp implementations do not generally offer the possibility of compiling source code into executables. Rather, Lisp programs may be distributed as a Lisp run-time environment together with an "image file", which is a saved representation of the state of the Lisp program. Alternatively, Lisp programs may also be distributed as a run-time environment together with compiled FASL files.[34] In other words, the distribution of a Lisp program often requires the distribution of files together with a run-time environment that will execute those files.

The fourth paragraph of the LLGPL addresses this distinct process of building and distributing Lisp applications. According to the LLGPL, applying the LGPL to these aspects of Lisp requires fundamental changes in the requirements and obligations of the GNU license. The fourth paragraph begins by providing an interpretation of the linking provisions of the LGPL:

Section 5 of the LGPL distinguishes between the case of a library being dynamically linked at runtime and one being statically linked at build time. Section 5 of the LGPL states that the former results in an executable that is a "work that uses the Library." Section 5 of the LGPL states that the latter results in one that is a "derivative of the Library", which is therefore covered by the LGPL.

Unfortunately, these provisions paint an inaccurate picture of the LGPL's requirements. Section 5 of the LGPL – notwithstanding the interpretation presented in the sentences above – does not distinguish between works that are statically or dynamically linked to an LGPL-licensed library. Rather, Section 5 clarifies that certain independent works which use an LGPL-licensed library can under certain circumstances become derivative works of that library. According to Section 5, a work may become a derivative work of the library even though the source code of that work does not contain portions of the library: the act of linking or compiling the work with the library will cause portions of the library to be incorporated in the linked or compiled work, and this linked or compiled work will then be seen as a derivative work of the library. Again, in providing that works may become a derivative work of the library, the LGPL does not distinguish between statically or dynamically linked works.[35] Indeed, Section 6 of the LGPL expressly contemplates that works may be either statically or dynamically linked with the library, and provides obligations for a party distributing both kinds of linked works.[36]

The next sentences of the LLGPL apply the previous (incorrect) interpretation of the LGPL to the Lisp context:

> *Since Lisp only offers one choice, which is to link the Library into an executable at build time, we declare that, for the purpose applying*

33 Third party Lisp libraries may similarly be loaded into the run-time environment, either as source code or as compiled files. See SEIBEL, *supra* note 16, at 17, 475.

34 Compiled Lisp files are referred to a FASL files, which stands for "fast-load file". Loading compiled Lisp files into the run-time environment can result in a faster and more efficient program. FASL files can be implementation dependent and may not be compatible between different implementations of Lisp. SEIBEL, *supra* note 16, at 475, n. 8.

35 On the other hand, certain commentators have differentiated between static and dynamic linking in determining the effect of the licenses. *See generally* "Working Paper on the legal implications of certain forms of Software Interactions (a.k.a linking)", which is available online at http://www.ifosslr.org/public/LinkingDocument.odt .

36 Section 6(a) of the LGPL addresses a situation where the "work that uses the library" is distributed as an executable linked with the library, and requires that the source or object code of the "work that uses the library" also be provided along with the linked work. This situation is colloquially referred to as statically linking the work with the library. Section 6(b) of the LGPL addresses a situation where the "work that uses the library" is linked to the library through a "shared library mechanism", which uses a copy of the library at run-time. This situation is colloquially referred to as dynamically linking the work with the library. In other words, both static and dynamically linking are governed by Section 6 of the LGPL. The preamble of the LGPL expresses the same when it states that "[w]hen a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library."

> *the LGPL to the Library, an executable that results from linking a "work that uses the Library" with the Library is considered a "work that uses the Library" and is therefore NOT covered by the LGPL.*

Because of this declaration, section 6 of LGPL is not applicable to the Library.

These provisions raise several problems. First, the statement that Lisp offers only the possibility of linking "the Library into an executable at build time" is not representative of all implementations of Lisp. It is correct that Allegro Lisp offers commercial licensees the possibility of creating an executable application – essentially a directory that contains the Allegro Lisp run-time environment, an image file, and a "license file" which together allow execution of the program.[37] It is not correct, however, that this is the only available method for distributing a Lisp application. Other commercial and free implementations of Lisp offer other alternatives for distribution. Lispworks, for example, provides the possibility of delivering an application as a dynamic library.[38] Allegro Lisp itself also offers additional options for distributing Lisp programs. For example, Allegro users that do not wish to build an executable may also distribute Lisp source code or compiled FASL files, and these files can be used by a user that already has an Allegro Lisp run-time system.[39] In addition, compiled FASL files may sometimes be distributed separately as patches to an application already executing on a run-time environment.[40] Free Lisp implementations also offer the possibility of saving a memory image of a running Lisp system, which can then be loaded by another user of the free implementation.[41] In other words, Lisp implementations offer a wide variety of distribution methods that are not addressed by the LLGPL.

Aside from the question of how a Lisp library may be distributed, the fourth paragraph of the LLGPL raises questions regarding the objectives and ambitions of the license. Indeed, the effect of the fourth paragraph of the LLGPL is to almost eviscerate the obligations of the LGPL. If Lisp programs can by definition only be distributed as an executable (an assumption that, as shown above, is not completely accurate), and such executables are stipulated as not being subject to the obligations of Section 6 of the LGPL, then the copyleft obligations of the LGPL will by definition never apply to any Lisp program. The weaker copyright obligations of the LGPL generally require linked applications to be distributed "in a form that allows for modification and relinking of the library," or pursuant to terms that "allow modification of the work … and reverse engineering for debugging such modifications." Under the LLGPL, however, even these weak copyleft obligations would never apply.

In other words, under the LLGPL, the copyleft provisions of the LGPL are essentially replaced with a rather permissive license.[42] Indeed, the copyleft obligations of the LLGPL may be

37 See http://www.franz.com/support/documentation/8.2/doc/runtime.htm.

38 *See* http://www.lispworks.com/documentation/lw61/DV/html/delivery-42.htm#pgfId-865189. Corman Lisp provides similar functionality. *See* Corman Lisp Common Lisp Development Environment, *available at* http://www.cormanlisp.com/CormanLisp/CormanLisp.pdf, page 73

39 *Id.* (providing that "[n]ote that because your source files and compiled versions of those files can be distributed without restriction, the way to distribute an application to another licensed Allegro CL customer without worrying about license agreement restrictions is to distribute your source files (and/or compiled versions of your source files), along with a file which creates the application."

40 Currently, however, not all commercial licenses to Allegro Lisp offer the rights to distribute a run-time environment together with a compiler that can read FASL files. See Franz's description of various runtime environment options, infra note 35. See also the short discussion regarding non-free runtime environments, *infra* text accompanying notes 42 - 43.

41 See http://www.sbcl.org/manual/index.html#Saving-a-Core-Image and http://www.clisp.org/impnotes.html#image

42 The LLGPL also seems to do away with several other obligations of the LGPL. For example, Section 6 of the LGPL also requires the provision of notices that the library is included in the work and that the library is covered by the LGPL. Section 6 also requires the retention of copyright notices. By broadly providing that Section 6 of the LGPL is not applicable to Lisp programs, the LLGPL seems to eliminate these requirements. In addition, to the extent the LLGPL can be applied to programs that are covered by Version 3 of the LGPL, the LLGPL may also eliminate the

summarised in the last sentence of that license: "[h]owever, in connection with each distribution of this executable, you must also deliver, in accordance with the terms and conditions of the LGPL, the source code of Library (or your derivative thereof) that is incorporated into this executable." In other words, under the LLGPL a licensee's copyleft obligations are limited to delivering the source code of the library itself.[43]

It should be noted that the permissive nature of the LLGPL stands in contrast to the fact that commercial Lisp applications are often developed using non-free platforms. For example, despite the fact that a particular application may be available under a permissive license, a commercial license to Allegro Lisp may nevertheless be necessary to run or modify that application. In addition, libraries developed with one implementation of Lisp are often not portable to another implementation.[44] As such, even though a particular application may be licensed under open source terms, a commercial license to a specific Lisp run-time environment may also be required to use that application. As such, the development of free and open source software in Lisp may require attention to both the license applicable to a particular program as well as the platform for which the software is developed.[45]

Of Macros

This section discusses two distinctive features of Lisp which are not clearly addressed by the LLGPL. First, Lisp contains "macros" – methods of defining new syntactical structures in Lisp – a feature not available in C or most other languages. Second, unlike C, Lisp programs have traditionally been constructed within a run-time interactive environment that interprets Lisp expressions. Neither of these features is expressly addressed by the LLGPL, and both raise issues regarding the interpretation and application of the LGPL. This section provides a brief overview of these features and the concerns they may raise in an open source license.

Macros are a distinctive feature of Lisp. In brief, macros are program snippets which take in Lisp code as input, manipulate that code, and return different Lisp code that is executed at runtime in place of the original code. Through such manipulations, Lisp macros allow users to extend the syntax of the language and create new constructions that can clarify and shorten code. Lisp macros differ from functions. Functions take arguments, and these function arguments are evaluated when the functions are executed. In contrast, the arguments in macros are not evaluated when the macro in executed. Instead, the macro returns code containing the unevaluated arguments, and these arguments are evaluated when the returned, macro-manipulated code is executed. Lisp macros also differ from C macros: while a C macro is essentially a textual search-and-replace mechanism, a Lisp macro provides a more general mechanism for generating code that preserves the data structures of the original code.[46]

How should the obligations of the LGPL affect the use of macros? On the one hand, it is not clear why macros should be treated differently than functions. Why should a Lisp program that uses the

requirement to provide installation information as required by the "Tivo" clause of the LGPL.

43 One ambiguity in this final, limited obligation of the LLGPL is the requirement to disclose the source code of the library and "your derivative thereof". It is difficult to clearly define what the requirement to disclose derivative works of the library refers to, since the LLGPL previously provided that works linked to the library do not constitute derivative works of the library and are not covered by the LGPL. This last requirement to disclose derivative works could either be seen as conflicting with the prior provisions of the LLGPL, as a requirement to disclose modifications to the library files themselves, or as some other undefined intermediate copyleft obligation.

44 Seibel, *supra* note 16, at 465, 475 n.8.

45 The Free Software Foundation described a similar problem with Java before Sun relicensed its Java implementation under the GPL. *See* http://www.gnu.org/philosophy/java-trap.html.

46 Much more complete explanations of the use and functionality of Lisp macros can be found in Seibel, *supra* note 16, ch. 7-8; Paul Graham, On Lisp, ch.7-8 (1993)

macros of a third party library be any less of a derivative work of that library than a Lisp program that uses the functions of that third party library? While macros do not constitute functions in the technical Lisp sense of the word, they do provide "functionality" and perform "operations" as those words are commonly understood.[47] On the other hand, the code generated by the macro may bear little resemblance to the text of the macro itself.[48] As such, it may not always be possible to say that a program that calls a macro incorporates the textual code of the macro. Rather, it may sometimes be more correct to say that the program that calls a macro incorporates code generated by the macro – and the LGPL itself provides that the output of a program need not necessarily constitute a derivative work of that program.[49] To make matters more confusing, Lisp macros are not expanded at either compile-time or run-time, but rather at an intermediate stage called macro-expansion time. Would this complicate the application of Section 6(b) of the LGPL, which defines a "suitable shared library mechanism" as a mechanism that uses a copy of the library already present on the user's system at "run-time"?

Answering the previous questions requires the untangling of complex legal and technical threads, and it is not the aim of this article to present a detailed analysis of these questions. However, any license tailored for Lisp should take a position on these questions in order to provide for legal clarity. It is unfortunate that the LLGPL does not provide any express guidance on the effect of the LGPL on macros.

As discussed earlier, the use of runtime environments is another distinctive feature of Lisp, a feature that has been adopted by other languages. A Lisp program may be developed incrementally by composing or loading functions into the run-time environment. Third party Lisp libraries may similarly be loaded into the run-time environment, either as source code or as compiled files. The Lisp run-time environment also impacts how programs are distributed. Lisp programs are generally distributed as a Lisp run-time environment together with either compiled FASL files or an "image file". Depending on the specific implementation, it may not be possible to distribute a single executable file for a Lisp program.

How should the LGPL relate to two functions loaded into the same Lisp run-time environment? Would the two functions be considered linked in the LGPL sense of that word? On one hand, linking two code files in the standard sense involves both the creation of links between the two files and the copying of the linked file (whether at build-time to create an executable or at run-time into memory to execute the program) into a larger program structure. In contrast, Lisp libraries present in an environment are already loaded into memory and do not need to be copied when a library function is called.[50] As such, it may be possible to assert that a program which uses a Lisp library already loaded into a runtime environment should not be considered "linked" to that function, and should not be subject to any obligations of the LGPL. On the other hand, the libraries present in the Lisp environment seem to satisfy the literal LGPL definition of "shared library mechanism", which is defined by the LGPL as using "at run time a copy of the library already present on the user's computer system".[51] As such, perhaps a program that uses a function loaded

47 See Graham, *supra* note 44 at 82 ("Since macros can be called and return values, they tend to be associated with functions. Macro definitions sometimes resemble function definitions, and speaking informally, people call do, which is actually a macro, a "built-in function." But pushing the analogy too far can be a source of confusion."); *Id.* at 84 ("Indeed, a macro is really a Lisp function –one which happens to return expressions").

48 For examples of how much a macro text can differ from the expanded macro program, see Paul Graham, *supra* note 46 at 97-98.

49 The preamble of the LGPL states that "[t]he act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does."

50 See Gary D. Knott, Interpreting Lisp, available at http://www.civilized.com/files/lispbook.pdf for a fuller explanation of how a Lisp interpreter stores library functions in memory.

51 The Free Software Foundation takes the position that if an interpreter includes certain facilities and "the interpreter is

into a runtime environment should be subject to the same obligations as programs that use a "shared library mechanism".

As with the question of macros, a license made for Lisp should provide a ready answer to the questions raised by the run-time environment. Again, it is unfortunate that the LLGPL does not provide easily applied rules for these questions.

Conclusion

One of the central assumptions of the LLGPL is that the GNU licenses, having been drafted with attention to a specific programming language, may need to be clarified for other programming languages. Indeed, the LGPL makes reference to technical details, such as the header files, linking and compilation, which are not applicable to all programming languages or to all situations. Even so, this article has shown that the clarifications made by the LLGPL to the original GNU license are largely unnecessary, and that the LGPL would probably be interpreted in a similar fashion without the clarifications proposed by the LLGPL. This is not to say, of course, that the LGPL comprehensively and expressly addresses all issues – as discussed, it does not expressly address the issues raised by Lisp macros or the Lisp runtime environments.

Licenses are legal documents, and chances are that their definitive legal interpretation will be made by persons with legal training but only a limited technical background. As such, to some extent, it is comforting that the interpretation of the GNU license does not depend on the details of specific programming languages. The technical detail necessary to draft similar license provisions for each and every technical context might prove too jargon-filled for the average court to apply.[52] On the other hand, the fact that such technical detail is not expressly included in the license text will not discharge a court from its obligation to understand such detail in order to properly interpret the license. In applying any software license, a court will need to understand the technical background regardless of whether it is clearly expressed in the text of the license.

In drafting software licenses – especially copyleft licenses that often refer to technical detail – it may be useful to keep these principles in mind. A well drafted license should not include an amount of technological detail that overwhelms the non-technical reader. On the other hand, it should to the extent possible provide for rules that are easy to interpret and apply in specific technical contexts. Balancing these often competing objectives is not a simple task. Nevertheless, having clear and easy to apply license terms will only increase the attractiveness of using open source software.

linked statically with these libraries or if it is designed to link dynamically with these specific libraries" then interpreted programs can be considered derivative works of those facilities. See GPL FAQs. This statement does not answer the questions raised in this section. First, the statement only addresses libraries that are statically linked or if the interpreter is designed to link dynamically with specific libraries – but not the situation of a library loaded into the run-time environment.

52 See Rosen, *supra* note 23, at 123-24 (stating that Section 2(d) of the LGPL is "an impenetrable maze of technological babble. They should not be a general-purpose software license.")

About the author

Eli Greenbaum *is an attorney at Yigal Arnon & Co. in Jerusalem, Israel, specialising in intellectual property law and transactions.*

The Rise and Evolution of the Open Source Software Foundation

Paula Hunter,[a] *Stephen Walli,*[b]

(a) Executive Director, The Outercurve Foundation: (b) Technical Director, The Outercurve Foundation.

DOI: 10.5033/ifosslr.v5i1.64

Abstract
Free and open source software (FOSS) project communities continue to grow and thrive. When such projects reach a certain critical point in their growth, corporations express interest in participating. Corporations have more stringent and robust software intellectual property (IP) management needs, however, and projects are not always up to the task. Neutral non-profit FOSS foundations have proved to be a solution to these problems, providing for the IP management needs of corporations while offering additional business and technical services to the project communities to encourage further growth and adoption. This article reviews how such neutral non-profit organizations have grown to meet the evolving legal, business, and technical needs of FOSS communities and businesses.

Keywords
Law; information technology; Free and Open Source Software; foundations

The growth and global participation in open source software development, aided by inexpensive and pervasive Internet access, has created a community of collaborators on whom software developers and IT professionals depend as a vital element in the software development process. As software intellectual property (IP) practices have matured, free and open source software (FOSS) communities have kept pace.

FOSS licensing has evolved over the past thirty years from the more liberal academic do-as-you-will licenses and initial ideas of software freedom to reflect the advancement of the general software landscape and include more complex methods of keeping software free. For example, with U.S. law recognizing software patents and the consequential risk involved with this, FOSS licenses began to introduce patent related clauses. As corporations became more interested in contributing to and using FOSS-licensed software, FOSS licenses were written using more traditional license structures and language. One of the key tools in this maturation has been the evolution of the non-profit technology foundation as a software IP management mechanism, as well as a hub for communications and collaboration.

Many volunteer-led and community-centred FOSS-licensed projects reach a point in their technology growth and evolution where corporations want to participate as well. Corporations have very different needs with respect to IP management, provenance tracking, and governance, as they are concerned with managing exposure to their patent portfolios and want to minimize to the potential for litigation. FOSS foundations provide such structure, as a number of key FOSS projects illustrate.

The Apache Software Foundation (ASF) formed around the Apache project as a non-profit charitable organization in 1999[1], adding a new, more structured license (Apache License 2.0[2]). This step happened as IBM became interested in participating with the intent to embed the Apache http daemon software in its Websphere product line. Likewise, the Open Source Development Lab[3] (OSDL) formed to support the Linux project in 2000 as a non-profit trade association to better manage IP risk as the Linux operating system became the cornerstone of a number of product lines from vendors that traditionally competed in the UNIX systems space. This non-profit later merged with the Free Standards Group[4] – a non-profit trade organization responsible for specifying Linux programming interfaces – to form the Linux Foundation[5]. The Eclipse Foundation[6] formed around the Eclipse IDE project in 2004, and has been the caretaker of the rigorous Eclipse software IP management process and the evolution of their FOSS license..

Each foundation represents different values and objectives to its constituency. Yet, what foundations have in common are governance structures to provide IP management and committer indemnification, as well as support mechanisms for community and collaboration.

The Outercurve Foundation was recently established to take this well-defined model and apply it forward for FOSS projects in such a way as to give vendors the benefits of such non-profit FOSS foundations without the expense and risks of creating their own foundations. The Outercurve Foundation provides the IP management and business operations associated with FOSS foundations as a non-profit trade association. It is technology, forge, and FOSS license agnostic (as long as the license is approved by the Open Source Initiative).

Public Good or Membership Benefits?

Many of the original FOSS foundations (e.g. the ASF and the Linux Foundation) were incorporated in the United States. An early decision for any FOSS foundation is whether to establish itself as a non-profit trade association (501(c)6 under U.S. tax law) or a non-profit charitable organization contributing to the public good (501(c)3 under U.S. tax law).[7] The FOSS community at large is very focused on the distinctions between these two types of non-profit organizations.

There are two major factors often discussed when evaluating these options: financial implications and control of the organization, in terms of who benefits. Many FOSS projects like the ASF or the Free Software Foundation are looking for a means to distance individual developers from the

1 http://www.apache.org/history/timeline.html
2 http://www.apache.org/licenses/
3 Weinberg, Bill. “OSDL: The Center of Gravity for Linux”. Presentation to the Silicon Valley Users Group. May, 2005. http://www.svlug.org/prev/2005jun/OSDL_Overview_SVLUG.pdf
4 http://www.linfo.org/free_standards_group.html
5 Walli, Stephen R. “Repeating History: The OSDL and Free Standards Group Merge”. 25 January 2007. http://stephesblog.blogs.com/my_weblog/2007/01/jim_zemlin_repe.html
6 http://www.eclipse.org/org/
7 http://www.irs.gov/charities/charitable/article/0,,id=96099,00.html, http://www.irs.gov/charities/nonprofits/article/0,,id=96107,00.html

finances of the organization while encouraging donations to the entity. The charitable organization status allows the organization to accept funds, which are tax deductible, and can be used to cover the basic operating expenses of the organization and, in some cases, fund development or specific project work. In many cases, a strong governance structure has evolved with the growth of the project (e.g. the ASF[8]), and thus codifying it with a formal charitable non-profit structure is a logical step in its lifecycle. The notion of "public good" is also very complementary to the philosophies of some FOSS communities, and thus the charitable organization is often chosen for more than simple accounting purposes.

The trade organization designation is frequently chosen by a collective of vendors, i.e. software companies, that want to collaborate on a project, jointly fund the effort, and establish a structure that ensures balanced control. While the primary distinction here is that the members are the beneficiaries of the efforts of the organization, in most cases a broader community can participate in and enjoy the fruits of the labor. The Linux project is an excellent example of a FOSS project that has benefited from significant vendor investment through a foundation. The Linux Foundation (a 501(c)6) trade organization under U.S. tax law) balances the needs and interests of its members in a very large community through its member programs[9] and membership bylaws.[10] In most cases, the tax implications are not a major factor; governance structure and IP management are far more important.

The Value of Foundations

Regardless of whether a FOSS foundation is organized as a trade organization or a charity, non-profit FOSS foundations offer projects three distinct types of services. First, they provide participants with a legal framework for software IP management in which commercial companies can work with FOSS projects and contributors. Foundations also provide technical services, such as software repositories and issue tracking, code signing certifications and technical mentorship. Lastly, foundations provide business operations and governance support, such as financial and banking services, membership management, and communications and PR around projects.

Legal Framework for IP Managment

Ownership Neutrality

One of the key benefits of using a non-profit FOSS foundation for project IP ownership and management is that it creates a neutral place for collaboration. Many corporations are loath to participate in FOSS projects held by other corporations that may be competitors or partners. There is a concern that their intellectual investments will go to other benefactors and they will see a poor return on investment. A neutral foundation holding the IP ownership allows all corporate sponsors to participate on equal terms. No one corporation owns the project software so partners and competitors alike can feel they are getting the best return on their contribution investment without giving others a significant advantage.

Foundations own the open source project's IP and have no commercial interests in the software, i.e. the foundations sell no products or services based on the software. Software copyrights are

8 How the ASF Works: http://www.apache.org/foundation/how-it-works.html
9 http://www.linuxfoundation.org/programs
10 http://www.linuxfoundation.org/about/bylaws

assigned or licensed by contributors to the foundation in a variety of ways through membership agreements, assignment or license agreements, and sometimes the project open source license itself. Patents are often licensed to the foundation. Providing a neutral place for companies and individuals to co-operate often leads to growth in the community of contributors. Well-managed IP also leads to growth in project acceptance and adoption as other parties become more confident adopting a project with well understood software provenance. Indeed, all of the largest and most active FOSS projects are managed by FOSS foundations.[11] Whereas, the next most active projects – smaller by an order of magnitude than the leading projects – are managed and owned by single corporations.

That FOSS projects run by a single corporation tend to be smaller may be due to concern around the consequences of a change in IP management or ownership. When a single corporation controls a FOSS project, what happens to the IP if the corporation changes direction or gets acquired? MySQL may be one of the most successful FOSS projects, but subsequent acquisitions by Sun Microsystems and then Oracle have left the broader project community confused.[12] [13] MySQL AB had rigorous IP management practices, but this means all the IP is now owned by Oracle, and they are rightly using it to corporate benefit. Over the past couple of years a number of competing FOSS-licensed database solutions are growing and interest in MySQL is waning.[14] If the ownership of MySQL had been held neutrally, none of the participants and users would have felt disenfranchised to the point of beginning their own projects or forking the MySQL software base, and the community may have continued on as strong as during its early years.

Liability and Risk Management

Foundations can also serve as liability firewalls or shields. Many companies are uncomfortable with publishing, sharing, and collaborating on open source software if they are the only copyright owner. Having a neutral third party hold the copyright ownership reduces some of that liability risk. A foundation, as a legal entity, acts as a shield that generally protects its members against liability for the contracts, commitments, and possible negligence of the foundation itself. The foundation (legal entity) may also protect the members that were not participating in a given activity for liability from other members' actions in a given situation (e.g. the introduction of infringing software into foundation owned software).

All FOSS licenses disclaim liability. Many vendors develop products out of FOSS-licensed projects (e.g. Red Hat Advanced Server is the product developed using software from the FOSS-licensed Linux project). Vendors are comfortable having product performance liability discussions with customers that have paid for the product and embed liability clauses in their product licenses. Many vendors, however, still feel there is the perception of liability risk for FOSS-licensed projects they own from unpaid users. Assigning the copyright ownership of a FOSS-licensed project to a non-profit foundation is a clear message of "non-copyright ownership". The vendors may still control the project direction through participation in the project governance and by supporting the primary developers and committers on the project, but there is a perception that they have reduced the liability risk as they don't own the project's software copyright. The foundation ownership acts to divert claims away from the original owner.

Additionally, the use of a legally incorporated foundation may provide certain soft benefits by

11 Henrik Ingo studied open source projects size and vitality and published his results on his website: http://openlife.cc/blogs/2010/november/how-grow-your-open-source-project-10x-and-revenues-5x (Valid: 2-May-2012)

12 First Sun: http://blogs.the451group.com/opensource/2008/05/07/mysql-licensing-redux/

13 Then Oracle: http://www.infoworld.com/t/dbms/oracle-eliminate-budget-plans-in-mysql-license-hike-323

14 http://arstechnica.com/business/2011/09/oracle-may-fork-itself-with-recent-mysql-moves/

playing to the perception that the legal issues and operations have been more carefully vetted and discussed with respect to members, contributors, and committers.[15]

As well as acting as a legal shield for members and contributors, foundations can also act to protect individual participants in the FOSS project by indemnifying their actions. This indemnification takes a number of forms.

FOSS project committers are primary developers on the project who have full write access to the software repositories, i.e. they are in the position of "committing" changes to the software. Committers may be individual software developers, and/or employees of independent software vendors (ISVs) or large corporations. Foundations can serve an important role indemnifying their committers depending on other governance and membership structures in place such that individual committers are not held personally liable for the software, regardless of the liability clauses embedded in FOSS licenses. This is certainly the case with the Outercurve Foundation and the ASF.[16]

Foundations typically explicitly indemnify their board directors and members as well in their governance policies. This is the case with the ASF[17], the Eclipse Foundation,[18] and the Outercurve Foundation,[19]

Code Provenance

Foundations may provide governance processes to track code provenance. A variety of legal opinions and practices discuss whether software contributions should be copyright assigned to a foundation, or copyright licensed into a software project's collaborative community or neither. Some believe in assignment of all rights under copyright (e.g. the Free Software Foundation[20]), while others believe a license of rights under copyright is sufficient (e.g. the ASF[21]). Still others feel all necessary rights are embodied in their open source license and membership agreements (e.g. the Eclipse Foundation[22]).

Each position and practice is defensible. Having all the rights of copyright ownership would allow the single owner to directly handle any litigation involving the software. It would also give the single owner the ability to unilaterally change the licensing terms for the software. This is what causes many developers concern if they are required to assign their copyright to a single entity when they contribute to a FOSS-licensed project, as it requires the single entity to be in a strong position of trust. Non-profit neutral foundations act in that capacity far better than for-profit corporations.

The opposite position – where everything is licensed and the community of licensees collectively

15 Personal correspondence with Andrew Updegrove of Gesmer, Updegrove LLP, Boston, MA, USA.
16 The Outercurve Foundation announced governance changes 1 November, 2010: http://www.prnewswire.com/news-releases/outercurve-foundation-changes-bylaws-and-governance-106520588.html. The president of the Apache Software Foundation, Jim Jagielski, confirmed similar support for Apache project committers in personal communication 18 June, 2012.
17 The Apache Software Foundation Bylaws: http://apache.org/foundation/bylaws.html
18 The Eclipse Foundation Bylaws: http://www.eclipse.org/org/documents/Eclipse%20BYLAWS %202011_08_15%20Final.pdf
19 The Outercurve Foundation Bylaws: http://www.outercurve.org/About
20 Eben Moglen answers the question as to why the Free Software Foundation expects copyright assignment for contributions: http://www.gnu.org/licenses/why-assign.html (Valid: 2-May-2012)
21 The Apache Software Foundation has contributors license their contributions to the foundation: http://www.apache.org/licenses/icla.txt (Valid: 2-May-2012)
22 The Eclipse Foundation doesn't use contribution license agreements, relying instead on the Eclipse Public License and membership agreements. http://www.eclipse.org/legal/eplfaq.php#RECRIGHTS (Valid: 2-May-2012)

owns the software – makes it difficult to act on such items as changing the licensing terms. Some view this as a re-enforcement of the community and the community's values.

The acceptability of contribution assignment and license agreement practices within communities of developers can be easily seen when you compare non-profit neutral foundations and for-profit corporations. Developers raise concerns that code assignments for FOSS contributions to corporations are at risk if the corporation chooses to close the project back behind its ownership wall or relicense them for its own corporate gain.[23] Again, MySQL stands as an excellent example here. MySQL AB required copyright assignment of all contributions to the for-profit company and made a considerable percentage of its profits selling closed licenses to its otherwise GPL-licensed software. This sole for-profit ownership caused a lot of concern through the two subsequent acquisitions by Sun Microsystems and then Oracle Corp. and ultimately led to the forking of the code base and new FOSS-licensed projects to replace the MySQL database.[24] Assignments and licenses to legally neutral non-profits remove such concerns.

It is important to note that regardless of the legal structures in place, software development practices to track the software contribution flow are also a critical and necessary part of the process of provenance tracking. Version control systems, issue tracking, and email archives all contribute to ensuring the software contributions themselves can be tracked, as well as the contributors assignment or license agreement. The ASF, Eclipse Foundation, Linux Foundation, and Outercurve Foundation all ensure such practices are in place for the projects they manage.

In any approach to assignment and contribution licensing practices, well managed IP with clear provenance tracking processes encourages adoption of FOSS projects by other organizations and grows the community of users and possible future contributions.

The License of a FOSS Project and License Curation

The license a FOSS project uses is often seen as more than a legal agreement for licensing the software. The license defines the project community's values for how they want to collaborate together and share the results of their work. Whether a project community believes all participants and contributors must license contributions and derivatives under the same license is wired into the choice the early community makes about the software. How the community wants to talk about patents that may relate to the software is embedded in the license, from the lack of discussion in such licenses as the BSD license to the various discussions of patents and patent retaliation embedded in licenses such as Apache License 2.0, the Eclipse Public License, and GPLv3.

Key current FOSS licenses evolved within foundations. As the Apache project evolved into the ASF, the simple BSD-like Apache 1.0 license evolved into the Apache License 2.0, which was a much more traditional license with respect to structure, and began to deal with discussions of patents. Likewise, the evolution of the Eclipse project's licensing has evolved with the Eclipse Foundation's governance over time,[25] from a project begun and anchored by IBM and the newly created IBM Public License, to the Common Public License as the Eclipse Foundation was created,[26] and most recently the Eclipse Public License as the Eclipse Foundation became the steward of the license.[27] The evolution of the GPL has been tightly bound to the Free Software

23 http://blogs.computerworlduk.com/simon-says/2010/08/on-contributor-agreements/index.htm
24 Again: http://arstechnica.com/business/2011/09/oracle-may-fork-itself-with-recent-mysql-moves/
25 The IBM Developer works FAQ on the Common Public License is informative on IBM's public statements about license evolution: http://www.ibm.com/developerworks/opensource/library/os-cplfaq/index.html
26 The Eclipse Foundation is formed 2 February, 2004 (http://www.eclipse.org/org/press-release/feb2004foundationpr.php)
27 IBM made the Eclipse Foundation custodian of the Eclipse Public License, 25 February, 2009:

Foundation and its evolving articulation of software freedom over the years, most recently culminating in the GPLv3 which attempts to set the concept of software freedom against the background of the modern Internet, the growth of the World Wide Web, and explicit clauses regarding patents.[28]

While FOSS licenses can be very foundation-centric based, nothing requires this to be so. For example, the Outercurve Foundation only requires that a project under its management use a FOSS license approved by the Open Source Initiative (OSI).[29] The Outercurve Foundation is not tied to a particular project and thus has the freedom to be agnostic as to a project's choice of license. Tying the choice to licenses the OSI has recognized as conforming to their Open Source Definition is a reasonable decision in light of the Outercurve Foundation's mission to support the growth of free and open source software projects and communities, and the OSI mission to be advocate for the benefits of FOSS.

Technical Services

The Forge and the Communications Channel

Every successful software project, regardless of how it is licensed, is supported by a software construction discipline that involves proper version control, configuration management, and build scripting, test automation, and issue tracking. These are the tools that enable consistent software delivery. Most of these tools are provided by modern internet-based forge sites (e.g. SourceForge, Codeplex, Github). As these are essential tools to supporting complex collaborative development, some FOSS foundations provide the tools as well, most notably the ASF and the Eclipse Foundation. The Linux kernel team has evolved their own infrastructure for handling this level of software construction discipline, but the Linux Foundation ensures the hosting. The Outercurve Foundation does not provide such toolsets, remaining forge "agnostic" and ensuring that projects are using the tools appropriately during the project proposal vetting.

Collaborative development requires strong communications channels as well. Developer email lists, IRC channels, forums, and wiki software all provide the basis of such communications. Foundations again can provide the infrastructure to support these channels.

Mentorship and Incubation

Software construction discipline is part of a project's culture. So too is the way a project makes decisions and communicates those decisions. Having a strong culture of sound software development practices allows a project to scale properly. New projects often come to a FOSS foundation in a stage of growth where they may not have instituted good practices, and to be accepted into the foundation the project needs to be educated in how the foundation's projects comport themselves.

The ASF and Eclispe Foundation run incubation phases for their new projects. New projects are assigned mentors and not allowed to graduate out of the incubation stage until they have demonstrated they adhere to the foundation cultural norms. The Outercurve Foundation did not grow up around a specific FOSS project with a history of specific practices in how to scale in a

http://www.eclipse.org/legal/cpl-v10.html

28 A history of the General Public License: http://en.wikipedia.org/wiki/GNU_General_Public_License

29 Outercurve Project Proposal requirements: http://www.outercurve.org/About/ProjectProposal

disciplined manner. Instead, projects are assigned a mentor to ensure the project is scaling in a disciplined way with appropriate practices.

General Management and Operations

Support Across the Project Lifecycle

As well as supporting a set of IP management mechanisms, foundations provide a set of business operational services to meet the needs of their managed projects at different stages of the project lifecycle. Historically, foundations created around existing projects had already evolved a set of software management practices essential for the software community to scale out to many developers, users, and releases of software. These collected software management practices became known as the Apache Way[30] at the ASF, and are likewise referred to as the Eclipse Way[31] at the Eclipse Foundation. Each of these foundations also hosts the original forge sites that support the software development processes, where the forge is the collection of software development tools (e.g. version control software and repositories, issue tracking) necessary for the development process.

While the foundations supporting the Apache and Eclipse projects each started around a single project, they have expanded to support new projects, in much the same way that the Outercurve Foundation was created to welcome FOSS-licensed projects that had reached a point in their evolution to need a foundation to support the next growth. Each of these foundations has developed mentoring processes to support new projects. The ASF[32] and Eclipse Foundation[33] each bring new projects through an incubation process to teach their respective development processes and IP practices and ensure over a period of a year or two that the project and the foundation are a match for one another. The Outercurve Foundation[34] instead directly matches a mentor to the project to ensure that the project leadership gets the best grounding in open source community collaborative and development techniques that meet its needs.

Different projects have different needs depending upon where they are in their life cycle and the experience that may already exist within the projects participants. Some projects, for example the Outercurve Foundation’s CoApp project, come into foundations in the concept phase, without a single line of code written. Other projects, such as the Outercurve Foundation supported Chemistry Add-on for Word, are mature projects with many downloads and users. These two projects have vastly different needs, from IP management practices to governance, operations, and marketing support, as well as technical mentorship and expertise to help organize and support collaborative development of software projects.

CoApp chose to license all software into the project (similar to the ASF projects), has run contests to encourage usage, and was started by a developer with a lot of experience in running an open source development community. In addition to using the Outercurve Foundation to manage the contest, most recently the CoApp project used the Foundation to pay a student to work on a summer work proposal similar to the Google Summer of Code programme run by Google. The Chemistry Add-on for Word, on the other hand, began with an assignment of software from

30 The Apache software management process is described: http://www.apache.org/foundation/how-it-works.html
31 The history of the Eclipse practices or Eclipse Way is described: http://wiki.eclipse.org/images/5/54/Eclipse-way.pdf
32 The Apache Software Foundation incubation process: http://incubator.apache.org/
33 The Eclipse Foundation incubation process: http://www.eclipse.org/eclipse/incubator/
34 The Outercurve Foundation mentorship program: http://www.outercurve.org/Blogs/EntryId/43/Outercurve-launches-new-Mentorship-Program-for-its-Projects

Cambridge University and Microsoft Corp. It has not taken advantage of the business operations of the Foundation, but has required more mentorship as it evolved, because there was not a lot of experience with open source community development practices. In addition, the project has also had to survive a transition in project leadership.

Each of these two projects uses the services provided by the Outercurve Foundation in different ways to match their needs.

Funding: Members, Dues, and Donations

FOSS foundations as organizations rely on the donations or dues of members and a volunteer workforce to get much of the work done, regardless of their non-profit organization.

The ASF is an excellent example of volunteer-led membership. The ASF is organized as a charitable non-profit organization and as such accepts donations, but it is volunteers that provide the majority of work in delivering against its mission, thus keeping operating costs relatively low. Donations cover the costs of items like the systems infrastructure used to support the forge.

When vendors invest in a non-profit trade organization, their expectations as members are different than what they would expect from a tax-deductible donation to a non-profit charitable organization. In addition to formal governance and operational support, members expect a staff to help drive programs and marketing. This staff can be comprised of full-time employees, employees assigned from member companies, and staff from firms that provide association management services (AMC). The Outercurve Foundation employs a hybrid model, with several full time staff members, while leveraging an AMC to provide financial, operational, administrative, and program management functions. This model allows the foundation to be nimble and scale as its project portfolio grows. Regardless of the staffing model, membership driven FOSS trade organizations are more expensive non-profits to operate than volunteer led charitable organizations working for the public good.

Conclusions

Developers have shared software since they began writing it, and the Internet has accelerated this process of shared collaboration. That said, collaborative software development needs more than the bandwidth of the Information Superhighway. To grow and thrive, projects need formal governance and legal structures that allow corporations to share the development work and contribute to the growth of the software and its community.

Collaboratively-developed software shared under liberal open source licenses continues to provide an enormous productivity boost for developers, speeds time-to-market for corporations, and delivers value to users. Open source foundations are a crucial part of the FOSS ecosystem. Foundations provide a simple, elegant mechanism through which corporate organizations can contribute to FOSS communities and develop their own projects by providing a neutral space for collaboration while mitigating legal risk; they also provide a safe haven for individual developers and projects of all sizes.

Perhaps most importantly, foundations support and enable community growth. An open source software project is only as good as its committers. Committers provide leadership and direction to their projects. Committers create the software but are also responsible for the discipline and quality of the software. Foundations provide the structure, governance and IP management to make it simpler for project communities to grow and flourish beyond their initial developers and users, as corporations become interested in participating. Foundations encourage communities to grow by providing an entity to hold the software property, ensuring no one person or entity throttles project growth by tightly holding the IP. Joining an established foundation also saves companies, and projects, the costs of starting a foundation from scratch, which is an expensive ordeal requiring a lot of expertise to avoid costly mistakes.

About the authors

Paula Hunter *brings a compelling combination of industry insight, executive-level business savvy and experience working with not-for-profits to the position of Executive Director of the Outercurve Foundation. Previously Hunter served as Director of Operations for SEMPO, the Search Engine Marketing Professional Organization, a non-profit professional association working to increase awareness and promote the value of Search Engine Marketing worldwide. Prior to SEMPO, Hunter was director of worldwide marketing and business development for the Open Source Development Labs, where she was instrumental in driving membership growth of industry advocacy group and lead initiatives to increase industry awareness and engage large enterprise IT organizations with OSDL programs. Previously, Hunter was general manager of UnitedLinux, a joint venture formed to create a unified Linux offering. She began her career at Digital Equipment Corporation, where she managed marketing programs for DEC's UNIX Workstation and PC product lines. Hunter received a BS in Computer Information Systems from Bentley University.*

Stephen Walli *is the Technical Director of the Outercurve Foundation. Walli has worked in the IT industry since 1980 as both customer and vendor. He was most recently a consultant on software business development and open source strategy. His customers included Microsoft, the Eclipse Foundation, the Linux Foundation. He's an adviser to Ohloh (acquired by BlackDuck), Bitrock, Continuent, and eBox. He organized the agenda, speakers and sponsors for the inaugural Beijing Open Source Software Forum as part of the 2007 Software Innovation Summit in Beijing. Stephen was VP Open Source Development Strategy at Optaros, a business manager at Microsoft on open source, and VP R+D and founder at Softway Systems, a venture-backed company that developed a UNIX portability environment for NT using free and open source software in combination with Microsoft-licensed Windows software, before being acquired by Microsoft. He was a long time* *participant and officer at the IEEE and ISO POSIX standards groups, representing both USENIX and EurOpen (E.U.U.G.) and a regular speaker and writer on open systems standards since 1991*

FOSS in the Italian public administration: fundamental law principles

Simone Aliprandi,[a] *Carlo Piana,*[b]

(a) ph.d. in Information Society, lawyer at Array and founder of Copyleft-Italia.it Project; (b) lawyer at Array and General Counsel (external) Free Software Foundation Europe.

DOI: 10.5033/ifosslr.v5i1.84

Abstract
We take a first reading of the recent modification to the fundamental law that governs the digital aspects of the Public Administration in Italy. These modifications require Public Administrations to prefer internally made solutions and FOSS solutions over proprietary ones, mandate an increased degree of interoperability and strengthen the push for open data.

Keywords
Italian law; information technology; Free and Open Source Software; public administration; e-government; public sector information; reuse of software programs; open by default;

The *Codice dell'amministrazione digitale* ("Digital Public Administration Act", also known with the acronym "CAD") is the most important Italian law about e-government. It includes provisions that govern the use of information technology as a privileged communication channel between Italian citizens and all the public administration system.

CAD's Article 68 establishes the core rules for all aspects related to openness in the Italian public sector: free and open source software[1] ("FOSS", par. 1 and 2), open formats and open data (par. 3). During 2012, these paragraphs underwent some important changes which created an unprecedented opening, *inter alia*, in favour of a preference for FOSS in the Public Administration.

Free and open source software, as well as in-house made or ad-hoc developed solutions or reused software, takes precedence by law (first reform)

The part of Art. 68 dealing with software procurement rules in the Public Administration was initially modified by Law 134/2012, approved by the Italian Parliament on August 7, 2012.

1 The law indeed uses both naming convention: "free/libre" (*libero*) and "open source" (*codice aperto*)

Here is an English translation of Par. 1 of Art. 68, resulting from this first reform. The first part remained unchanged and read:

> *Public administrations must acquire computer programs or parts thereof as a result of a comparative assessment of technical and economic aspects among the following solutions available on the market:*
>
> *a) develop a solution internally*
> *b) reuse a solution developed internally*
> *c) obtain a free and open source license*
> *d) obtain a proprietary license of use*
> *e) a combination of the above*

After this paragraph, that makes FOSS an overriding choice by law, the following language was initially added :

> *Only when the comparative assessment of technical and economic aspects demonstrates the impossibility to adopt open source solutions or any other software solution already developed (at a lower price) within the public administration system, the acquisition (by license) of proprietary software products is allowed. The assessment referred to in this paragraph shall be made according to the procedures and the criteria defined by the Agenzia per l'Italia Digitale, which, at the instance of interested parties, also provides opinions about their compliance.*

Second reform: enter the cloud option, some refinement

On December 17, 2012 a new law (commonly known as "Italian Digital Agenda Reform") was approved by the Italian Parliament: a broad-spectrum legislative package about digital innovation for all the Italian Public Administration information systems. It adds a further amendment to Article 68 of CAD.

With this amendment, Italian Public Administrations can choose between 6 options (and not 5 as it was in the previous version): cloud computing solutions are expressly included in the type of solutions that can be evaluated in the procurement process.

What is interesting is that the rest of Article 68 is quite different and more detailed. The principles governing the comparative analysis that every Public Administration is required to perform before choosing one of these 6 options is now set out in Paragraphs 1-bis and 1-ter.

Here is a complete version of the current wording of Par. 1 of Art. 68 CAD:

> *1) In accordance with the principles of economy and efficiency, return on investment, reuse and technological neutrality, public administrations must procure computer programs or parts thereof as a result of a comparative assessment of technical and economic aspects between the following solutions available on the market:*
>
> *a) develop a solution internally;*
> *b) reuse a solution developed internally or by another public administration;*
> *c) adopt a free/open source solution;*
> *d) use a cloud computing service;*
> *e) obtain a proprietary license of use;*

f) a combination of the above.

1-bis) For this purpose, before procuring, the public administration (in accordance with the procedures set out in the Legislative Decree 12 April 2006, n. 163) makes a comparative assessment of the available solutions, based on the following criteria:

a) total cost of the program or solution (such as acquisition price, implementation, maintenance and support);
b) level of use of data formats, open interfaces and open standards which are capable of ensuring the interoperability and technical cooperation between the various information systems within the public administration;
c) the supplier's guarantees on security levels, on compliance with the rules on personal data protection, on service levels [,] *taking into account the type of software obtained.*

1-ter) In the event that the comparative assessment of technical and economic aspects, in accordance with these criteria of paragraph 1-bis, demonstrates the impossibility to adopt an already available solution, or a free/open source solution, as well as to meet the requirements, the procurement of paid-for proprietary software products is allowed. The assessment referred to in this subparagraph [more correctly: "the above subparagraph"] *shall be made according to the procedures and the criteria set out by the Agenzia per l'Italia Digitale, which, when requested by interested parties, also expresses opinions about the compliance with them.*[2]

Some comments about the criteria and the role of the Agenzia per l'Italia Digitale

It is apparent how the criteria established to evaluate the "value for money" of the different solutions are now more detailed and encompass a larger spectrum of factors, in comparison with the former version of the law, which was more blunt and mainly referred to the "price" factor. However, the law is far from clear as to how the different factors must weigh in the evaluation, if they are all equal, if any can be completely ignored.

Here enters the *Agenzia per l'Italia Digitale*[3] (literally: Agency for a Digital Italy), which is in charge of defining practical rules for such evaluation. The *Agenzia* has a really difficult task, as the law is not technically well drafted. Besides the poor definition of the criteria and their scope, there is uncertainty as to what is the mandate of the *Agenzia.* The latter is in charge of defining the criteria "as per this subparagraph", where the criteria for such evaluation are actually defined in the earlier paragraph. But this is easily resolved. It is our opinion that the principles and criteria for the evaluation remain the same, whereas paragraph 1-ter adds a further and special requirement for the adoption of proprietary software (and arguably cloud services). Such requirement being that the evaluation must show that the inadequacy of available solutions under the first two categories (development of an ad hoc solution –which is then available for reuse to other PAs– or reuse of an existing one already developed for the PA; and free/open source) reaches an "impossibility level" Finally, it is also uncertain which metrics can be used, if a given model shall be preferred and so on.

2 An updatetd and verified text of the CAD is available at http://www.digitpa.gov.it/amministrazione-digitale/CAD-testo-vigente. Last accessed on 2013-03-19.

3 Carlo Piana is also a member of the consulting commitee appointed by the *Agenzia* to advise in the process of defining the evaluation criteria, called for at http://www.eupl.it/opensource/lagenzia-per-litalia-digitale-emana-una-call-per-la-formazione-di-un-tavolo-di-lavoro-volto-a-definire-i-criteri-di-valutazione-ex-art-68-cad.html.

One thing seems very clear, the procurement of proprietary solutions (or of cloud services for that matter) is an *extrema ratio*, available only if previous solutions fail. The evaluation between *ex ante* equally viable solution shall happen only between the preferred ones, otherwise the entire paragraph would lack any conceivable purpose and its words would be read against their very meaning. The only latitude that the Agency can arguably take is to define when "impossible" is impossible, in other words, to establish when no viable solutions exist and therefore the proprietary solution is by far the obvious winner.

This is a great achievement. FOSS solutions are to be preferred, and to a great extent. It is noteworthy that even when a software solution is made internally by the PA, it must be made available for reuse (i.e., offered at no licensing costs and accompanied by the complete source code to all other PA requesting it) to all other PAs.[4] One of the simplest form of reuse is to share it under a public FOSS license.

Interoperability as a mandatory goal

Paragraph 2 of Art. 68 has not been touched by the two recent reforms presented above (its last modification dates back to 2010). However, its content is relevant and also noteworthy. It establishes interoperability as a basic principle to achieve true openness in the public sector.

> *2) In the preparation or acquisition of computer programs, public administrations, whenever possible, must adopt solutions which are: modular; based on functional systems disclosed as stated by Article 70; able to ensure the interoperability and technical cooperation; able to allow the representation of data and documents in multiple formats, including at least one open-ended (unless there are justifiable and exceptional needs).*
>
> *2 bis) The public administrations shall promptly notify the Agenzia per l'Italia digitale the adoption of any computer applications and technological and organizational practices they adopted, providing all relevant information for the full of the solutions and the obtained results, in order to favour the reuse and the wider dissemination of best practices.*

Although this is clearly a provision that does not favour any licensing or business model, it is apparent that it creates an environment where FOSS licensing has a certain edge, at least in principle, because of the possibility to peruse the permissions that are embedded in it even without the cooperation of the copyright holders.

A new "open format" definition and the "open by default" principle in PSI

Another part of Article 68 which was involved in the second reform discussed above is Paragraph 3. This part of the Article provides a definition of two relevant aspects that contribute to define a healthy ecosystem for FOSS.

4 See Art. 69 CAD, which provides the basic principles for the so-called "reuse of software programs" (within public administrations). Here is an English translation of par. 1: "Public administrations owning computer programs made on specific demand by the public sector have a duty to give them in source code form, with the complete documentation, at no charge, to other public administrations that require them and want to adapt them to their needs, unless justified reasons."

The first definition is about **open formats**:

> *an open format is a data format which is public, documented exhaustively and neutral with respect to technological tools for the use of data*

The second definition relates to **open data**.

> *open data are data that:*
>
> *1) are available under the terms of a license permitting their use by anyone, even for commercial purposes, in disaggregated format;*
>
> *2) are accessible through the information and communication technologies, including public and private telecommunication networks, in open formats; are suitable for automatic processing by computer programs and equipped with relative metadata;*
>
> *3) are available for free through the information and communication technologies, including public and private computer networks, or are available to the marginal costs incurred for their reproduction and dissemination.*[5]

But this is not the entire story. The Italian lawmaker decided to introduce an "**open by default**" principle for all the public sector information. This choice, that sounds quite revolutionary for the Italian legal order, has been made operational by modification of Article 52 (entitled "Electronic access and re-use of public administrations' data"), where we now find the following paragraph:

> *Data and documents, which the public administrations own and publish without the express adoption of a proprietary license (as defined in Article 2, paragraph 1 of Legislative Decree 24 January 2006, n. 36), are released as open data in accordance with the definition provided in Article 68, paragraph 3.*

This provision is particularly important as it paves the way to open data by the Public Administration to an unprecedented level. Although it does not actually mandate the open data principle, and by all means it does not *per se* mandate the publication of data in general, it requires an actual decision when desiring to restrict the use of data that are published.

This also marks an additional U-turn in the field. Before this legislation, Italian PAs were facing a constant threat from a restrictive reading of the liability rules of public officers. Said reading was that if the PA could have been in the position to obtain benefits from the release of data (even to other Pas!) for a monetary compensation and failed to do so, the public officer making this decision could be asked to restore the loss suffered by the PA. Now, with the enactment of the opposite principle, the decision is clearly authorized –nay, defaulted to– by law, and it becomes clear that the widest release open data is a goal of the Public.

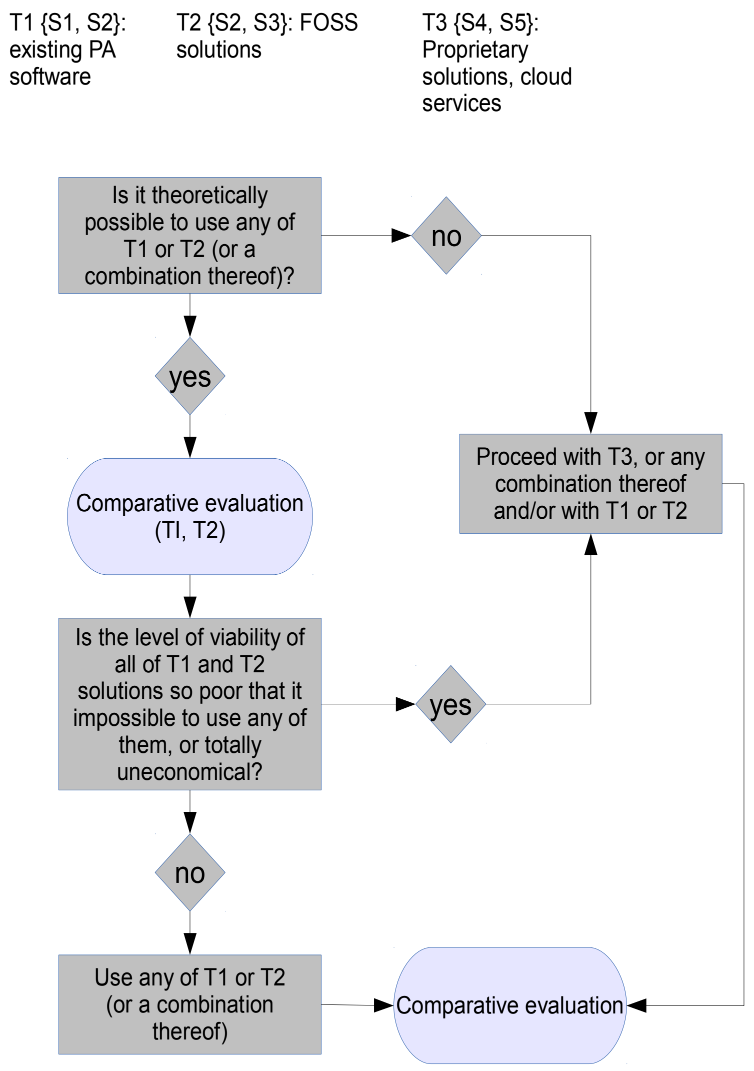
Conclusions and perspectives

5 The Agenzia per l'Italia digitale shall establish, with deliberation, exceptional cases, identified according to objective, transparent and verifiable, in which they are made available at higher rates to marginal costs. In any case, the Agency, in the treatment of exceptional cases identified, will follow the guidance provided by Directive 2003/98/EC of the European Parliament and of the Council of 17 November 2003 on the re-use of public sector information, implemented by legislative Decree 24 January 2006, n. 36.

To our knowledge, Italian law is the farthest-reaching law to date favouring the use of FOSS in the Public Administration and the general openness of their IT systems to create a public commons created by public money. The decision was made in a dire situation of the national economy and inspired by practical reasons (spending review) rather than idealistic ones. It seems however a new direction that can hardly be changed. Only it can be made less compelling by a slack implementation, if not outright non compliance.

Vigilance is therefore required.

A simple flowchart of the process for evaluating available software solutions


T1 {S1, S2}: existing PA software
T2 {S2, S3}: FOSS solutions
T3 {S4, S5}: Proprietary solutions, cloud services
Is it theoretically possible to use any of T1 or T2 (or a combination thereof)?
no
yes
Comparative evaluation (TI, T2)
Proceed with T3, or any combination thereof and/or with T1 or T2
Is the level of viability of all of T1 and T2 solutions so poor that it impossible to use any of them, or totally uneconomical?
yes
no
Use any of T1 or T2 (or a combination thereof)
Comparative evaluation

About the authors

Simone Aliprandi *is an Italian lawyer and independent researcher who is constantly engaged in writing, teaching and consulting in the field of copyright and ICT law. He has an additional degree in Public Administration Science and he holds a Ph.D. in Information Society at the Bicocca University of Milan. He founded and still coordinates the Copyleft-Italia.it project and has published several books devoted to openculture and copyleft. He also collaborates as a legal consultant with Array (http://www.arraylaw.eu).*

Carlo Piana *is the General Counsel (external) of the Free Software Foundation Europe and an Editor of this Review. An Italian lawyer in private practice, he advises a number of Public Administrations on open data and reuse of software through FOSS licensing. He is the founder of Array (http://www.arraylaw.eu), a group of IT lawyers focussed on FOSS and digital liberties.*

Volume 05 } Issue 01 } March 2013

Thanks to our Sponsors

IFOSS L. Rev. is published by its Editorial Committee. The Committee gratefully acknowledges the sponsorship of its many supporters and donors, including those who have permitted the upstart of the review with their contribution :

nlnet FOUNDATION mozilla FOUNDATION

Available online at: http://www.ifosslr.org