Packaging Open Source

Mark Webbink[a]

(a) Visiting Professor of Law and Executive Director of the Center for Patent Innovations at New York Law School.
With thanks to Daniel Hopkin for his invaluable assistance in the preparation of this article

DOI: http://dx.doi.org/10.5033/ifosslr.v1i2.26

Abstract
Now with the increasing popularity of virtual computing environments we are observing the packaging of operating systems, including Linux, with applications to create software appliances, essentially applications that carry their operating system with them. With this rapidly expanding market opportunity, traditional proprietary software vendors are increasingly interested in the "rules of the road" for open source licensing and, in particular, for packaging Linux with an application into a software appliance. This paper is intended to provide background information for such application developers interested in creating software appliances utilizing open source components while ensuring both their open source license obligations as well as protection of their own copyrights and patents.

Keywords
Law; information technology; Free and Open Source Software

Info
This item is part of the **Articles** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement.
This article has been independently peer-reviewed.

Running proprietary (non-open source) software on an open source operating system, such as Linux, has become commonplace. Major proprietary software vendors such as IBM, Oracle, and Adobe have adapted their proprietary applications to run on top of a Linux operating system without concern for open source licensing issues. In addition, Linux vendors frequently ship proprietary applications with their Linux distributions, although the packaging practices and open source license compliance may vary from one to the next. Major device manufacturers, including Sony, Philips, Cisco, and Nokia, have utilized a Linux operating system in both their open (modifiable) and closed (non-modifiable) devices for some time, although their open source license compliance has also frequently required greater effort. Even with open source versions of JAVA we are seeing both open source and proprietary files combined in the same class libraries.

Now with the increasing popularity of virtual computing environments we are observing the packaging of operating systems, including Linux, with applications to create software appliances, essentially applications that carry their operating system with them. With this rapidly expanding market opportunity, traditional proprietary software vendors are increasingly interested in the "rules of the road" for open source licensing and, in particular, for packaging Linux with an application into a software appliance. This paper is intended to provide background information

for such application developers interested in creating software appliances utilizing open source components while ensuring both their open source license obligations as well as protection of their own copyrights and patents.

The paper will first review the copyright, patent, and key license provisions that arise in an open source context. The paper will address the more commonplace packaging of open source applications within mainstream Linux distributions and the license compliance issues that arise. Next, the paper will examine alternative packaging models, including embedded devices, JAVA class libraries, and software appliances. Finally, the paper will suggest some best practices for software application developers to maximize their value and to minimize their risks.

Copyrights and Patents in the Software Context

Under U.S. law any software will be automatically covered by copyright at the time it is developed unless the author expressly disclaims copyright protection or the code, for a variety of exceptional reasons, does not rise to the level of copyright protection. For our purposes, we will assume that most code is subject to copyright protection. Such copyright protection extends to both the source code and binary versions of the software, the latter being viewed much in the same light as a translation of a literary work.

There are three principal rights under copyright law that are held by the copyright holder in software: the right to copy the software, the right to modify the software, and the right to distribute the software. Under traditional proprietary models, the copyright holder elects not to share these rights with the party receiving a license in the software. Thus, under most proprietary software licensing models the end user has no right to copy the software (other than the statutory right to make a back-up copy of the software under 17 U.S. Code § 117). In addition, proprietary vendors commonly include restrictions on the decompiling or reverse engineering of the software (such restrictions being of limited scope with respect to such actions taken purely for the purpose of interoperability), thus barring the modification of the software. Finally, proprietary licenses will generally limit the redistribution of the software to the absolute transfer of the single copy covered by the license. As a consequence, packaging issues in the purely proprietary software context tend to be matters of negotiated contracts, not merely end user licenses.

By contrast, under open source licenses[1] copyright holders share these rights to copy, modify, and redistribute, sometimes without restriction and sometimes with limited restrictions as to obligations arising upon redistribution. As a consequence, the licensee under an open source license will enjoy rights to modify, combine, copy, and redistribute software that they would not typically enjoy under a proprietary license. In this context the open source licensee must then be aware of treatment under copyright law of modified or combined software code.

U.S. copyright law addresses such modifications and combinations as, respectively, derivative works and compilations, which includes collective works. The formal definitions for these terms may be found in 17 U.S. Code § 101:

> A “derivative work” is a work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other

1 When referring to open source licenses in this paper, the author limits the reference to those licenses that have been certified as complying with the Open Source Definition as promulgated by the Open Source Initiative. That list of licenses may be found at http://www.opensource.org. The one exception to this convention is the inclusion of the GNU General Public License, version 3, as an open source license.

> form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications which, as a whole, represent an original work of authorship, is a "derivative work".
>
> A "collective work" is a work, such as a periodical issue, anthology, or encyclopedia, in which a number of contributions, constituting separate and independent works in themselves, are assembled into a collective whole.
>
> A "compilation" is a work formed by the collection and assembling of preexisting materials or of data that are selected, coordinated, or arranged in such a way that the resulting work as a whole constitutes an original work of authorship. The term "compilation" includes collective works.

It is important to understand the distinctions between these three forms of works. The first distinction is between compilations and collective works. Note that all collective works are considered a form of compilation, but not all compilations are collective works. That said, the primary distinction between the two is that compilations may consist entirely of non-copyrightable materials, where as a collective work will contain only copyrightable material. This can be better visualized in non-software terms. If I have factual information, say the names of all of the people who live in a city and their addresses, that information is not inherently protectible by copyright, i.e, there is nothing unique in the expression of that information – it is what it is. However, if I organize that same information in a unique way, say by first names, that particular organization may be protectible as a compilation even though the data within the compilation is not inherently protectible by copyright. Compilations may also exist in a combination of copyrightable material and non-copyrightable material, e.g., a book containing court opinions (which are public information and in the public domain) which are organized and annotated to provide more useful information to persons wanting to understand the decision.

By contrast, collective works consist of separate and independent copyrightable materials that have been organized into a single work. The typical non-software examples are periodicals, anthologies, or encyclopedias. Although collective works are considered a subset of compilations, they actually have as much, if not more, in common with derivative works than they do with non-collective work compilations in that both derivative and collective works are based upon pre-existing copyrightable works.

The key question is then where to draw the line between a work which is merely a collective work and a work that constitutes a derivative work. The treatise Nimmer on Copyright[2] identifies the following distinction:

> A derivative work consists of a contribution of original material to a pre-existing work so as to recast, transform or adapt the pre-existing work. . . . A collective work will qualify for copyright by reason of the original effort expended in the process of compilation, even if no new matter is added.

In either case, the contributions required to produce a new work, whether a derivative or collective, must be more than minimal and meeting the standards for copyright protection.[3] So what is the distinction between derivative works and collective works when it comes to software. The distinction largely arises in the number of underlying works involved. By definition a collective work consists of more than one underlying work while a derivative work consists of a

2 Nimmer on Copyright, Lexis-Nexis 2010, release no. 72, §3.03[A]

3 For a more thorough discussion of what constitutes sufficient originality to qualify for copyright protection, see Originality Requirements Under U.S. and E.U. Copyright Law, © 2007 Software Freedom Law Center found here: http://www.softwarefreedom.org/resources/

single underlying work. But as with all such matters, there is a point at which this simple distinction no longer holds. When one underlying work is materially modified and so combined with a second work such that the combination of the two effectively takes on a singular identity in terms of use and perception, the combination has moved more closely to the definition of a derivative work. At the same time, a collective work may include one or more underlying works which are, themselves, derivative works of pre-existing works.

This distinction between derivative and collective works may more clearly be drawn in software when examining how two independent works relate to each other. Where two independent works are capable of sharing information, passing such information back and forth through published interfaces or by a temporary connection, such as a pipe, and not by a modification of one of the works by the other work, those works when combined in a single package would constitute a collective work but neither would likely be considered a derivative of the other. On the other hand, where one such application, when compiled at the same time as a second application, modifies the second application in such a way as to cause that second application to act in a unique manner with the first application, the combination would likely constitute a derivative work of the first program even though the two works, in their source code form, are separate and independent.

Finally, the copyright holder in the compilation holds a copyright only in the compiled work, not necessarily in any of its component parts (although it would be somewhat atypical for the copyright holder in a collective work to not have produced some new, separately copyrightable work, in producing the collective work). Thus, the copyright holder in a compilation or collective work must have permission from the copyright holder of each component part to include it in the compilation or collective work.

As we will see, some open source licenses address only derivative works and others address both derivative works and compilations/collective works. And, of course, whether a new work, which utilizes only some but not all of a pre-existing work, is a derivative work is a matter of statutory interpretation and case law. For our purposes in this paper, we will consider all works that incorporate some or all of the code of a pre-existing work to be derivative works.

At the same time, software code developed in or imported into the U.S. may be subject to patent protection under U.S. law. Such patents coverage is not defined by specific source or object code but, rather, by the claims set forth in the patent. Whether specific source or object code infringes a given patent depends on whether the structure and operation of that source code (i.e., the methods it employs) reads on the claims of the patent.

Such so-called “software patents” do not distinguish between code licensing models. As a consequence, both open source and proprietary software licensing models need to be concerned with software patents. The manner in which such licenses address the subject of software patents varies considerably. In this paper we will only concern ourselves with the manner in which various open source licenses address software patents.

Open Source Licenses – Derivative Works, Collective Works and Patents

The Open Source Initiative lists 60 licenses that meet the OSI definition of open source. Although large in number, the differences among many of these licenses are relatively small. In fact, some are virtually identical but for their names, e.g., the Common Public License and the IBM Public License. All of these licenses cover the source code, and all permit the licensee to copy, modify,

and redistribute the code.

The simplest group of licenses are those that permit the exercise of the rights to copy, modify, and redistribute without limitation. These include the widely-used Berkeley Software Distribution (BSD) license, MIT license, and Apache Software License. Source code, and corresponding binaries, of software licensed under these licenses may be readily incorporated with other open source code or even into proprietary code without concern over license compatibility. That is the upside. The downside is that, in their brevity, these licenses do not address patents in any manner. One solution to this problem, as advanced by Intel, has been to combine the BSD with an express patent license, found here: http://infiniband.sourceforge.net/duallicense.htm.

Of the open source licenses that incorporate a restriction on redistribution, the most widely used are the GNU General Public License (both version 2 and version 3), the GNU Lesser General Public License (version 2), the Apache License version 2, Artistic License version 2, Common Public License, and Mozilla Public License version 1.1. Let us consider how each of these licenses addresses issues of derivative works, collective works, and patents. For our purposes we will consider any mere compilation that does not rise to the level of a collective work to be a non-issue.

GNU General Public License, version 2

("GPLv2" - http://www.opensource.org/licenses/gpl-license.php)

As with other open source licenses, GPLv2 permits licensees to make modifications of the work and to redistribute the work, either in its original form or as modified (GPLv2 refers to such derivative works as "works based on the Program") so long as the distributed work continues to be covered by this license (section 2 of the license). That aspect of GPLv2 is fairly well understood. But GPLv2 does not only address derivative works; it also addresses compilations in the form of collective works (which it refers to as "mere aggregations"). For compilations, or mere aggregations, GPLv2 does not apply or place limitations on the licenses pertinent to non-GPLv2 works included in the compilation. The same is not true for collective works. With respect to collective works, GPLv2 states in Section 2:

> These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
>
> Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

This language has been commonly misunderstood to mean that all works included in such a collective work are re-licensed under GPLv2, thus overriding their original license. That is not the case for two reasons: (1) it is not what the language says; and (2) the copyright holder of GPLv2 code has no legal right to impose a license change on the holders of the copyrights in the other code included in the collective work. What the GPLv2 licensor can, and does, say is, if you are going to include my work in a collective work, then the collective work must also be licensed under GPLv2. What that means is all works included in the collective work must either be under

GPLv2 or under a license that is compatible with GPLv2.[4] This concept of compatible licensing is manifested in the Section 2 phrase "whose permissions for other licensees extend to the entire whole." It doesn't say that GPLv2 is applied to each component, only that the permissions granted under GPLv2 (those being the permissions to copy, modify and redistribute in source code form) apply to each component.

Finally, GPLv2 does not include an express patent license grant. Rather, in Section 6 the GPLv2 makes clear that no other restrictions can be imposed on recipients, which would include any restriction arising from a patent held by the distributing party. In section 7 the GPLv2 makes clear that, if conditions are imposed on the distributing party that would interfere with the rights granted under the license, the distributing party is not to redistribute the software. These provisions have been construed as granting an implied license from a GPLv2 distributing party under any patent claims of that distributing party that read on GPLv2 code they distribute and preventing such a distributing party from entering into any form of license agreement with respect to patent rights that would not extend to all downstream recipients. GPLv2 does not prevent distributing parties from entering into other forms of agreements related to patents as evidenced by the Microsoft-Novell arrangement announced in the fall of 2006.

GNU General Public License, version 3

("GPLv3" - http://www.fsf.org/licensing/licenses/gpl.html)

While in most instances GPLv3 operates in the same manner as GPLv2, there are some important distinctions. As with GPLv2, GPLv3 addresses both derivative works and collective works with the following language from Section 0:

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.
A "covered work" means either the unmodified Program or a work based on the Program.

The language from Section 5(c) makes clear that all such derivative and collective works must be licensed under GPLv3:

> 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.

A key difference regarding collective works between this language and that of GPLv2 is that GPLv3 now states that it will apply to all parts of the collective work. Gone is the reference in GPLv2 to "permissions" which allowed GPLv2-compatible licenses to govern other parts. This is an important distinction that should not be lost.

GPLv3 deals with compilations in much the same manner as GPLv2 with the following provision in Section 5:

> 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

4 See http://www.fsf.org/licensing/licenses/ for a list of GPL-compatible licenses

> 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.

This language now provides a clear statement that distinguishes a compilation from a collective work and makes clear that works licensed under GPLv3 can be included in such compilations so long as nothing in the license for the compilation interferes with the operation of the GPLv3 license with respect to the code licensed under GPLv3.

Unlike GPLv2, GPLv3 contains an express patent license in Section 11. That license only applies if the licensor makes a modification to GPLv3 licensed code and then redistributes the code, at which point the license applies not only to the licensor's modification but to the entire modified work, whether merely a derivative work or a collective work. There continue to be restrictions in GPLv3 on patents licensed from third-parties that apply to the GPLv3-licensed work, and any party looking to redistribute a modified GPLv3-licensed work would be well advised to be familiar with these provisions.

GNU Lesser General Public License, version 2.1

("LGPLv2" - http://www.opensource.org/licenses/lgpl-license.php)

The LGPLv2 mirrors the GPLv2 in its application to derivative works, collective works, and compilations. The difference arises in Sections 5 and 6 of LGPLv2 where it permits the use of LGPLv2-licensed code (typically, libraries) with code licensed under a different license, including a proprietary license. The one limitation imposed on such combinations is that the licensor of the entire work (including both the non-LGPLv2 code and LGPLv2 code) must provide the licensee with the source code for the LGPLv2-licensed code and permit the licensee to make modifications in that code AND not prohibit the licensee from reverse engineering the non-LGPLv2 code included in the entire work solely for purpose of debugging the modifications to the LGPLv2 licensed code. This provision does not grant the licensee the right to copy, otherwise modify, or redistribute the non-LGPLv2 code included in the entire work.

LGPLv2 contains provisions in Sections 10 and 11 pertinent to patents that correspond with Sections 6 and 7 of GPLv2.

Apache License, version 2

(http://www.opensource.org/licenses/artistic-license-2.0.php)

The Apache License does not follow the same formula for derivative and collective works, but it still operates in much the same manner as the GPL. The Apache License uses the following definition:

> "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

Note two things about this definition. A modification that does not rise to the level of an original work of authorship, i.e., it is not protectible by copyright, does not create a derivative work. Thus

the author of such a de minimis change would have no independent copyright in the work. Second, no collective work is created if a separate work merely links or binds by name to the interfaces of the Apache-licensed work or derivative thereof. Using the terms linking and binding in this context causes some confusion because in GPL semantics these terms are used to imply two pieces of code that have been combined in such a manner as to create a single work unless the licensor provides an exception.[5] Hence, the standard approach for the Apache License in this context is somewhat equivalent to the GPL+exception. By contrast with the GPL, the Apache License makes no specific reference to compilations likely on the premise that no such reference is necessary.

The Apache License contains an express patent license grant in Section 3. Unlike GPLv3, the patent license only extends to those patents held by a contributor of code and that apply to that contributor's contribution, not to the entire work.

Artistic License, version 2

(http://www.opensource.org/licenses/artistic-license-2.0.php)

The Artistic License deals separately with derivative works, collective works, and compilations. In addition, the Artistic License differentiates derivative works into (a) those works for which the modifications were explicitly requested by the copyright holder for the licensed work, referred to as the Standard Version, and (b) those works containing modifications not explicitly requested by the copyright holder for the licensed work, referred to as Modified Version. This approach parses copyright law in a rather unusual way in that it permits modifications, but the manner in which you document and redistribute the derivative work varies depending on whether it is a Standard Version or a Modified Version, with the redistribution obligations being far more burdensome for Modified Versions.

The Artistic License addresses compilations in Section 7:

> You may aggregate the Package (either the Standard Version or Modified Version) with other packages and Distribute the resulting aggregation provided that you do not charge a licensing fee for the Package. Distributor Fees are permitted, and licensing fees for other components in the aggregation are permitted. The terms of this license apply to the use and Distribution of the Standard or Modified Versions as included in the aggregation.

This is pretty straightforward and should not be a cause for confusion or concern.

The Artistic License addresses collective works in Section 8:

> You are permitted to link Modified and Standard Versions with other works, to embed the Package in a larger work of your own, or to build stand-alone binary or bytecode versions of applications that include the Package, and Distribute the result without restriction, provided the result does not expose a direct interface to the Package.

This is one of the more challenging turns of a phrase you will run into in an open source license. For example, in saying you can redistribute the result "without restriction," does it mean you can't place any limitations on what the licensee does with the collective work or does it mean that none of the restrictions in this license apply. The former would be quite onerous, and the latter could be stated more clearly. The same for the last phrase of this paragraph when it says "provided the

5 See http://www.fsf.org/licensing/licenses/gpl-faq.html#LinkingWithGPL

result does not expose a direct interface to the Package." What does that phrase mean? Does it mean that you can embed the Package and use the Package within the context of your larger work so long as some using your larger work is unaware that the Package is included and can't use it independently. That would appear to be the most logical interpretation, but it could certainly have been stated more clearly.

Finally, on the subject of patents, the Artistic License included a patent grant, but only from the Copyright Holder. Specifically, Section 13 states:

> This license includes the non-exclusive, worldwide, free-of-charge patent license to make, have made, use, offer to sell, sell, import and otherwise transfer the Package with respect to any patent claims licensable by the Copyright Holder that are necessarily infringed by the Package.

To fully understand this section, you need to refer back to the license's definition of Copyright Holder:

> "Copyright Holder" means the individual(s) or organization(s) named in the copyright notice for the entire Package.

Because the patent grant is not limited to the contributions of a particular copyright holder, a subsequent modified version could, in fact, implicate the patents of another, earlier copyright holder in a manner that copyright holder did not intend. Let's take a hypothetical. "A" is the copyright holder of a Standard Version, and that Standard Version reads on claims 1 and 2 of a patent also held by A. However, the Standard Version does not read on claim 3 of that same patent. "B" comes along and modifies the Standard Version, creating a Modified Version, in a manner that now reads on claim 3 of A. Under this language, A's Claim 3 would be licensed with B's Modified Version because A would necessarily be named in the copyright notice (remember, the party creating a derivative work only holds the copyright in their modifications, not in the original work that was modified). A further complicating factor in this language is the statement that the licensed patent claims would include "any claims licensable by the Copyright Holder," not solely the claims owned by that Copyright Holder. What if those "licensable" patents are royalty-bearing or have use restrictions? This requirement would have made more sense had it been limited to claims "licensable by the Copyright Holder without restrictions or royalties."

It is worth noting that the Artistic License, version 2 does not appear to be widely used at this time, and perhaps for good reason. I have included it here to indicate another variation in approach and to demonstrate the need for careful drafting.

Common Public License, version 1

("CPL" - http://www.opensource.org/licenses/cpl1.0.php)

Eclipse Public License, version 1

("EPL" - http://www.opensource.org/licenses/eclipse-1.0.php)

The CPL and EPL are treated as the same license since the text of the licenses are identical in all but name. The CPL takes yet another approach to derivative works, somewhat defining them in circular fashion. The CPL defines a Contribution to include a change or addition to an existing program licensed under the CPL, but excludes from this definition contributions "which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program." So, in permitting the creation of derivative works, the CPL makes clear that compilations are permitted, but it does not really

address the concept of collective works at all. In theory, then, a person could modify a CPL-licensed work specifically for the purpose of including it in a larger work, and so long as they made the source code for their modified, CPL-licensed work available, they could license the larger work under any terms they choose, including proprietary license terms. Of course, a key question in this context is what constitutes a "separate module." This approach is further supported by the language of Section 3 which states:

> A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:
>
> a) it complies with the terms and conditions of this Agreement; and
>
> b) its license agreement:
>
> > i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
> >
> > ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
> >
> > iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
> >
> > iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

This approach gives a licensee wishing to redistribute the ability to incorporate the CPL-licensed code into larger works, including binary-only works, and to license those larger works under different license terms so long as the license terms of the CPL continue apply to the incorporated CPL-licensed work.

The CPL contains an express patent license grant in Section 2.b that is purely contribution based. This provides certainty to a contributor as to the patent license obligations the contributor is assuming. In this regard the provision is quite similar to that of the Apache License, version 2.

Mozilla Public License, version 1.1

("MPL" - http://www.opensource.org/licenses/mozilla1.1.php)

The final license considered is the MPL. It is important because it is the license used for the popular FireFox browser and Thunderbird e-mail client. One thing unique about the MPL is the breadth of its scope when defining modifications. Like the CPL and EPL, the MPL defines modifications as being any change to the code whatsoever. However, both the CPL and EPL then narrow their scope by stating that the definition of "Contributions" does not include that which would not constitute a derivative work. The MPL contains no such limitation and, thus, claims to apply to changes that, in and of themselves, may not constitute copyrightable material. This would appear to introduce a rather unique, and perhaps undesirable, aspect to the MPL in that any attempt to assert the license with respect to a non-copyrightable change could only be enforced under contract law, not copyright law.

That anomaly aside, the MPL is also unique in its approach to both compilations and collective works, treating them as one and the same. The MPL does so through the grant language contained in sections 2.1(a) (covering initial contributions):

> under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work

and 2.2(a) (covering subsequent contributions):

> under intellectual property rights (other than patent or trademark) Licensable by Contributor, to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code and/or as part of a Larger Work

and reaffirmed with the language in section 3.7:

> You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

As a consequence, the MPL provides broad rights to combine MPL covered code with any other code, including a larger work (either a compilation or collective work) so long as you make the source code of the MPL licensed code available.

The MPL follows the contribution approach to its express patent license grant language, i.e., a contributor is only providing a patent license to that contributor's contribution, including any necessary patent license pertaining to the combination of that contribution with the existing work. There is one last catch to the MPL, and it arises in the context of the express patent license. According to the language of sections 2.1(b) and 2.2(b), the patent license only pertains to the code in its source code form. The pertinent language appears in the following:

> 1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.
>
> 1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:
>
> > A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.
> >
> > B. Any new file that contains any part of the Original Code or previous Modifications.
>
> 2.1 (b) under Patents Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Code (or portions thereof).

> 2.2 (b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Modifications made by that Contributor (or portions thereof); and 2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

Section 3.8 of the MPL permits executable versions of the work to be distributed under other license terms, so long as the source code associated with the MPL-licensed portion of those executables is made available under the MPL. This is a potential trap for the unwary in that an executable version of the MPL-licensed source does not necessarily carry with it the patent claims licensed with the source code, and the licensor of the executable is not obligated to provide a patent license covering the executable.

Applying License Terms to Software Distributions

Having examined the way a variety of open source licenses treat the issues of derivative works, collective works, compilations and patents, we now turn to the mechanisms that may be utilized to effect these distributions. In this context we will focus on two principal means of packaging software: media packaging and electronic packaging. Given that both forms of packaging technically exist only in electronic form, a couple of definitions are helpful. Media packaging is defined as providing content on the same physical media or by means of the same media channel. Electronic packaging is defined as the organization of content into distinct electronic packages.

As a first premise, I assert that the proper measure of the relationship of one or more software applications is not the packaging but the actual interrelationship of the packages. In other words, regardless of mode of delivery, the issue is whether the works are derivative, collective, or merely aggregated (compilations) when they are resident on the computer that will run them.

Consistent with that first premise, let's examine various forms of physical packaging. At least three forms of such media packaging come to mind: fixed media, movable media, and electronic transmission, such as FTP. No one would logically argue that the mere fact two software applications reside on the same fixed media, such as a computer hard drive, creates any form of special relationship between those applications under copyright law. If that premise is true, which I believe it to be, then the fact the media is movable, such as a flash drive, or the content is delivered via electronic transmission, such as via FTP, would fall into that same category. So the mere presence of two software applications in the same media packaging should, in and of itself, never be a concern. This second premise is true under all of the licenses discussed above.

The second form of packaging, electronic packaging, may, at first blush appear to differ. Electronic packaging of software can occur in any of the following examples: an electronic package management system, such as RPM; a JAVA class library, or a software appliance. Because of the distinct differences of each of these electronic packages, it is worth examining them individually.

A package management system does not itself define the interrelationship of the software applications that may be included in any one package, other than to contain information about the included software. An example of a popular open source package manager is RPM. RPM consists of a software package file format and a free software tool which installs, updates, uninstalls, verifies and queries software packaged in this format. Each RPM package contains a package label holding the following information: the name of the software; the version of the software; the

package release; and the architecture on which the package is intended to run, e.g., X86. As a consequence, an RPM package does not, by itself, indicate a relationship among the code contained therein and any other code. In this manner, an RPM carries many of the same characteristics as a media package and not unique characteristics that define the interrelationship of the packaged software.

This same approach is true for a JAVA class library as packaged in a JAR file. Such files are mere aggregation tools; they again do not define the interrelationship of the individual programs or functions that may be included therein. In fact, there is no particular reason that everything in a single JAR file be licensed under the same license. It is entirely possible for a JAR to contain code licensed under the GPL, the LGPL, or any of a number of different open source licenses. The mere fact that these program or functions have been aggregated into a single JAR to make them readily available in a JAVA application context does not impute a relationship among the varying contents of a JAR. Such a relationship may or may not exist, and it is only when such an interrelationship exists that one need to go to the question of the nature of that relationship, i.e., derivative work or collective work. From this viewpoint, JAR files are mere compilations.

A software appliance is a software application packaged with just enough operating system components (abbreviated as JeOS) to allow it to run on hardware or a virtual machine. A JeOs is a customized operating system designed to fit the needs of a particular software application. However, the mere fact that the operating system has been customized to minimize its size and to work with a specific software application does not change the characteristic of the operating system as an independent work. This is especially true with respect to Linux, and it is worthwhile to digress for a second in looking at the standard Linux construct.

Linux, or perhaps more appropriately, GNU/Linux, consists of a kernel and various utilities and applications built to run on that kernel. The Linux kernel itself is of a modular construct, with groups of files organized into these modules. The internal Linux kernel modules pass information back and forth among themselves using internal symbols. In GNU/Linux the kernel is licensed under GPLv2. The utilities and applications, which exist in what is commonly referred to as "user space," are licensed under a variety of open source licenses, including most of the licenses mentioned above.

The Linux kernel provides a number of defined application interfaces that the user applications are permitted to use, and so long as a user space application does not seek to export an internal symbol, i.e., one of the symbols passed among the internal Linux modules, it is construed to be an independent work and neither a derivative work of the Linux kernel or a collective work as combined with the Linux kernel. Hence, a user space application may be under almost any license, including a proprietary license, as long as it behaves in the specified manner. Both non-commercial and commercial Linux distributions combine a Linux kernel with a wide variety of user space applications to provide a robust, general purpose operating system that is reasonably easy to install, manage and update. However, the mere fact that the kernel and user space applications are packaged in the same media does not change their characteristics as independent works.

Applying this construct of a typical Linux operating system to a software appliance that utilizes a Linux kernel-based JeOS, it is consistent to view the software application married with the JeOS and the JeOS as independent works in the same manner as the Linux kernel and user space applications. The mere fact that they have been placed on the same media or in the same electronic package does not change that relationship. Of course, this assumes that the application has been developed independently from the JeOS, and the more development distance that can be

placed between the two, the stronger the argument for independent works.

If a software application developer decides to build its own JeOS utilizing Linux and elects to export internal kernel symbols for use by the application, the software application and the JeOS may be construed as either a derivative work or a collective work, and the GPLv2, the license applicable to the Linux kernel, would be the governing license.

Lessons Learned and Lessons Applied

From the foregoing analysis we see that mere packaging of code rarely impacts the licenses applicable to the various components of the code. Rather, one needs to look at the interrelationship between software applications, or software applications and the operating system, to determine whether the two are independent works merely compiled into the same package or whether they could be construed as either derivative works or a collective work. Consequently, software application developers looking to utilize one of these packaging techniques can adopt practices that decrease the likelihood of a finding of a derivative or collective work and increase the likelihood of the combination being construed as a mere compilation or aggregation, regardless of packaging. Specifically, software application developers should consider the following practices:

- Maintain distinct development trees for the software application and any operating system on which it is to run, including a JeOS.
- If possible, utilize an operating system, including JeOS, that has been developed by an independent third party. This increases the likelihood that the two works will be properly construed as independent works.
- Do not export or seek to export Linux kernel internal symbols for use by your application. If such an export is necessary, make sure the driver or interface effecting that export is available under the GPLv2.
- Ensure that your software application in the form distributed is equally capable of running on other forms of operating systems. For example, if the application is being distributed as a part of a software appliance utilizing a JeOS, it is helpful if that same application code is capable of running on any standard version of the Linux operating system. This strengthens the argument of independence.

Conclusion

Open source licenses differ in their treatment of derivative and collective works, although almost all take the same benign approach to compilations or mere aggregations. It is not the manner in which an application is packaged with other applications or an operating system kernel that determines whether one of the two works is a derivative of the other or whether the combination is a collective work, it is the actual manner in which the two interoperate and are dependent on each other. Following a few simple practices will increase the likelihood that your software application will always be considered an independent work regardless of the packaging mechanism that places it in proximity to an open source operating system like Linux.

About the author

Mark Webbink *is a Visiting Professor of Law and Executive Director of the Center for Patent Innovations at New York Law School. Webbink is also a Senior Lecturing Fellow at Duke Law School and has served as an Adjunct Professor at NCCU Law School. From 2000 to 2007 Webbink served in various capacities with Red Hat, Inc., including General Counsel, Deputy General Counsel for Intellectual Property, Senior Vice President and Secretary. Webbink presently serves on the board of directors of the Software Freedom Law Center. Webbink has written and spoken extensively on the subjects of open source software, software patents, and patent reform. Webbink received his B.A. Degree from Purdue University in 1972, his Masters in Public Administration from the University of North Carolina – Chapel Hill in 1974, and his J.D., magna cum laude, from North Carolina Central University School of Law in 1994. Webbink maintains a website on open source and intellectual property law at www.walkingwithelephants.com.*