License Integration Patterns:
Addressing License Mismatches in Component-Based Development

Daniel M. German
Department of Computer Science
University of Victoria, Canada
dmg@uvic.ca

Ahmed E. Hassan
School of Computing
Queen's University, Canada
ahmed@cs.queensu.ca

Abstract

In this paper we address the problem of combining software components with different and possibly incompatible legal licenses to create a software application that does not violate any of these licenses while potentially having its own. We call this problem the license mismatch *problem. The rapid growth and availability of Open Source Software (OSS) components with varying licenses, and the existence of more than 70 OSS licenses increases the complexity of this problem. Based on a study of 124 OSS software packages, we developed a model which describes the interconnection of components in these packages from a legal point of view. We used our model to document integration patterns that are commonly used to solve the license mismatch problem in practice when creating both proprietary and OSS applications. Software engineers with little legal expertise could use these documented patterns to understand and address the legal issues involved in reusing components with different and possibly conflicting licenses.*

1. Introduction

Most large software applications are not built from scratch; they are built by combining several components such as reused code snippets, self-contained binary libraries, or other applications. Component-based development (e.g., [19]) has been a catalyst for the creation of many successful projects.

Over the last decade, various research efforts have focused on the technical aspects of supporting and improving component-based software development processes. For example, Garlan *et al*. discuss the challenges of component development due to architecture and interface mismatches [8]. However, little attention has been directed toward the legal complexities surrounding component-based software development. Builders of component-based applications must combine components with different licenses to create a new software application, i.e., derivative work, with its own licensing terms.

With the widespread use of open source components, practitioners are likely to pick open source components when building their next large component-based application. In contrast to commercial components, open source components have a large number of licenses. At last count there are 70 approved open source licenses. Each license has its own set of permissions and restrictions. Combining components of differing and possibly conflicting licenses is the next big challenge for component-based development. We call this challenge the *license-mismatch problem*.

The Bugzilla software application [31] is a great example to highlight the sheer complexity of this problem in modern component-based development. In its most common instance, Bugzilla makes use of 82 packages. These packages use 10 different licenses including original 4-clauses BSD, new 3-clauses BSD, Artistic v1, GNU General Public License (GPL) v1, GNU GPL v2, GNU Lesser General Public License v2.1, MIT, Apache v2, and IBM Public License v1.0. Many of these licenses conflict with each other. For example, the GPL licenses insist that all code linked to them must be GPL-licensed as well; one would expect that the final product, i.e., Bugzilla, would be licensed under the GPL. However, Bugzilla is licensed under the Mozilla Public License 1.1. To combine all these conflicting licenses, the developers of Bugzilla had to adapt and modify their technical solutions and architecture to ensure that Bugzilla complies with the other ten licenses.

Several models have been proposed in the past to model the selection of components (such as [3]). However, these models do not address how the license of a component affects the requirements, the architecture and the potential uses of a component-based application. Others have warned about the difficulties of including open source software in commercial software [1, 16, 25, 29]. IBM's *Ariadne* appears to be the only tool that fully incorporates the management of intellectual property in software development [4].

License compliance is rapidly becoming an important and critical challenge for many software organizations worldwide. Companies like Hewlett-Packard (http://www.hp.com), Black Duck Software (http://blackducksoftware.com) and Palimida (http://palamida.com) have created infrastructures and toolsets to help software organizations tackle the license-mismatch problem. For example, Koders.com by Black Duck Software is a source code search engine that permits developers to limit their code search to specific licenses. The FOSSology Project by HP provides the infrastructure to automatically detect licenses in software packages to aid in identifying possible license mismatches [10].

In contrast to technical challenges, the license mismatch problem is a more complex challenge for which software engineers have limited training and knowledge. Undergraduate's exposure to legal issues is confined to a few lectures in a single course as per the ACM Software Engineering Curriculum guidelines.

The main contributions of this paper are twofold. First, the development of a model to describe licenses, and the implications of licenses on the reuse of components. This model is the first step toward creating frameworks which could automatically verify legal compliance. Second, we highlight and document the efforts of the open source community in addressing the license-mismatch problem. Through a detailed study of 124 OSS packages, we identified and documented patterns that are commonly used to integrate components with different licenses. By documenting these patterns, we aim to 1) demonstrate the effect of legal issues on the architecture of modern software applications; 2) define common vocabulary for discussing and analyzing the effects of licenses on software; 3) provide a set of patterns for practitioners to learn best practices, and for academics to improve research and education matters associated with software licensing issues.

This paper is organized as follows. Section 2 gives a brief overview of the legal protections available for software. Section 3 presents our model to describe licenses and the legal consequences of combining components of different licenses. Section 4 presents our system of patterns. Section 5 concludes the paper and outlines avenues for future work.

2. Legal protections for software

From a legal point of view, software, or more specifically, a "'computer program' is a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result."[37]. Computer programs are usually protected using one or more legal alternatives: a) *trademarks*—which protect the software name, logos, and any specific mark associated with the software; b) *trade secret*—the source code of the program is kept secret, only binary or obfuscated versions of the program are distributed, c) *patents*—software related inventions are patentable, giving the owner of the patent a monopoly on its exploitation though not all countries permit software patents, d) *copyright*—gives its owner certain exclusive rights such as making copies of the software. See [2, 11, 14, 24] for comprehensive discussions on how these protections are applied to software. This paper focuses on the use of copyright to protect software.

2.1. Exclusive rights and licenses

The copyright owner of a software system has various exclusive rights over it [37]: 1) to make copies of it; 2) to prepare derivative works based on it; 3) to distribute copies for sale, rent, lease or lending; 4) to perform the work in public; and 5) to display the work in public. A copyright owner can exploit these exclusive rights for a fixed period of time. An owner can explicitly forfeit the copyright of a work. A work with no copyright owner is said to be in the public domain.

A license is a legal mechanism used by the copyright or patent owner (the licensor) to grant permission to others (the licensees) to use and exploit her intellectual property in ways that would otherwise be forbidden by copyright or patent law [15, 17, 28]. For example, an integrator who wants to modify and include a component as part of a larger software application and sell the application will require the rights to create a derivative work of the component, make copies of it, distribute it and sell it. The integrator has these rights if a) she owns the intellectual property of the component, b) the intellectual property of the component is in the public domain, or c) she has a license for the component which grants her these rights.

2.2. Collective and derivative works

The concepts of collective and derivative works are fundamental to understanding the legal issues involved in creating a component-based software application. The United States Copyright Act defines a collective work as "a work [...] in which a number of contributions, constituting separate and independent works in themselves are assembled into a collective whole" and a derivative work (also known as derived work) as "a work based upon one or more preexisting works [...] in which a work may be recast, transformed, or adapted" [37].

An integrator of a component-based software application must determine if this application is a collective or a derivative work of any given component. If the application is a derivative work of one or more components, then the integrator must have a grant to the right to create a derivative work from each of these components. The license of the component-based application will be subject to the restrictions imposed by these grants. Unfortunately there is no simple method to determine if a work is a derivative or a collective work; the final decision is made by a judge in a court of law (see [26] for a detailed discussion).

Collective and derivative works are entitled to their own copyright protections. The author of a collective or a derivative work owns the copyright only to the material contributed by her, as distinguishable from the previous works. The author of a new collective or derivative work is subject to the copyrights of the components that are part of the work [38].

2.3. Open source licenses

Open source licenses create a legal framework which permits the collaboration of different individuals and organizations in the creation of software: "Open source licensing has become a widely used method of creative collaboration that serves to advance the arts and sciences in a manner and at a pace that few could have imagined just a few decades ago" [13].

The Open Source Initiative (OSI) (`http://www.opensource.org/`) defines and promotes the "Open Source Definition" (OSD). The OSD defines open source as software that is distributed under a license that satisfies 11 specific criteria for an open source license [32]. These criteria include, for example: the source code should be available; the license should allow modifications and derived works, and their further distribution; and the license must not discriminate against fields of endeavor nor persons or groups.

The OSI is also responsible for officially approving licenses as open source. Some widely used licenses are considered open source but have never been approved, such as the *original BSD* license (also known as 4-clauses BSD), and the GPL_1 license (we use a subscript after the license name to indicate a particular version). As of August 2008, there exists 72 OSI-approved open-source licenses. 26 cannot be reused by anybody else but their author because they include the name of their author inside the license itself and are not in the public domain, hence they cannot be modified by a new author; e.g. the Apple Public License, and the PHP License. Four licenses have been voluntarily retired. Some open source licenses are recent versions of older licenses (e.g., the GPL_3 is a newer license than the GPL_1 and GPL_2) but from a legal point of view each is different and independent from the earlier versions.

3. A model to describe the licensing requirements of components

From a legal point of view, we define a component-based software application (S) as a work composed of one or more software components (C_i) functioning together. Our definition of the term component is very lax: a component is any software product (proprietary or open source), including any "glue" that might be required to integrate or adapt one or more components. This definition is similar to that in [3], with the addition of components that are first modified and then reused. Each component (C_i) has its own copyright owner (who can be the end user or the integrator putting S together) and its own license ($\mathbb{L}(C_i)$). Similarly, S can have its own license ($\mathbb{L}(S)$).

A component C can be reused in two primary forms:

White-box: Using one or more files of C, either in their original or modified form. Usually these files are distributed as part of S.

Black-box: Using C without any modifications to it. C can be distributed separately or along with S.

White-box reuse is likely to create a derivative work of the modified component. Even copying a small fraction—less than 100 LOCs—of a component C might result in S being considered as a derivative work [18]. The integrator or user must acquire the right to modify and potentially redistribute all the files of C in S.

Determining whether S is derived or collective work for a black-box reuse is a more complex task which depends on the nature of the use and interconnection of each C_i with the rest of S and other reused components in S. A component C can be reused in one of the following ways, which we call **interconnection types**:

Linking: Calling functions or methods in C using dynamic or static linking.

Fork: Stand alone execution via a fork or exec system calls. C is executed in a separate space from S. The communication between the rest of S and C might be done via pipes, sockets or files.

Subclass: Other parts of S inherit one or more classes of C.

IPC: C is built as a service or server. Other parts of S use C via a well-defined process intercommunication protocol, such as CORBA and COM.

Plugin: S extends the functionality of C using C's plugin-architecture.

A software system S can be modelled as a directed dependency graph with each component (C_i) as a node. An edge between two nodes indicates that the two components are interconnected through one of the aforementioned interconnection types. This graph is not acyclic, as two components might require each other to function. All the nodes (components) in the graph should be available for all the features of S to properly work, and the user of S should have the right to use them.

3.1. Modeling open source licenses

As we have seen in section 2, an open software license provides its licensee with a grant to one or more of the exclusive rights owned by the copyright owner of that component. Because an open source license is unilateral, each grant is granted provided a set of conditions are satisfied; if one of such conditions is violated, then the grant is not given[28]. by the US Court of Appeals for the Federal Circuit in Jacobsen v. Katzer [13].

A license (L) is, therefore, a set of grants. The conditions for each grant to right r (G_r) can be represented as a set of m conjuncts. All conjuncts should be satisfied for the licensor to receive such grant:

$$G_r(L) = p_1 \wedge \cdots \wedge p_m$$

For example, one of the several grants of the *original BSD* allows the licensee to distribute derivative works in binary form as long as the following three conjuncts are satisfied: 1) "Redistribution [..] must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution." 2) "All advertising materials mentioning features or use of this software must display the following acknowledgements: This product includes software developed by the <OWNER> and its contributors." 3) "Neither the name of the <OWNER> nor the names of its contributors must be used to endorse or promote products derived from this software without specific prior permission."

Intellectual property lawyers, who are familiar with the copyright laws in the applicable jurisdiction, must interpret which grants are present in a license, and their corresponding conjuncts.

3.2. Modeling the licensing of component-based applications

The integrator and the user of a component-based software application S expect to have some rights to it, such as the right to execute it, to create other derivative works from it, the right to license it, and to sell copies of it. The needed rights vary between applications, while the available rights depend on the components of S and their interconnection types.

Let us assume that S is composed of n components $\{C_1...C_n\}$. U denotes the user of S and I denotes the integrator of S. $\Delta(C)$ denotes the set of all derivative works of a C. S is a derivative work of C iff $S \in \Delta(C)$. S might be a derivative work of zero or more of its components C_i.

U and I need to obtain a license to each C_i that grants them the rights needed to use C_i in S. For example, if I wants to sell binary copies of S, and $S \in \Delta(C)$ then the license to C, which we denote as $\mathbb{L}(C)$, should grant the right to distribute derivative works of C in binary form. This grant is likely to impose conditions that should be satisfied by I.

The specific grant required by I to be able to integrate C in S depends on two factors:

1. Whether $S \in \Delta(C)$, which depends on the interconnection type of C with the rest of S; and
2. The rights required by I and U to use, and potentially distribute C.

For example, let us assume that I wants to distribute copies of S under a proprietary license P; one of S's components, C is licensed to I under the terms of the GPL_2, i.e., $\mathbb{L}(C) = GPL_2$. The position of the Free Software Foundation (FSF), creator of the GPL_2, is that if $\mathbb{L}(C) = GPL_2$ and C is interconnected to the rest of S via *Black-box Linking*, then $S \in \Delta(C)$ and I acquires a grant to create and redistribute copies of the derivative work of C from $\mathbb{L}(C)$. One of the conjuncts of this grant under the GPL_2 is that $\mathbb{L}(S) = GPL_2$, i.e., the license of the derivative work of C should be under the GPL_2. Since $\mathbb{L}(S) = P$, this conjunct is not satisfiable, hence under these circumstances C cannot be used in P. On the other hand, if C is interconnected to the rest of S via *Black-box Forking* then according to the interpretation of the FSF $S \notin \Delta(C)$, and under the GPL_2 I acquires a grant to make and redistribute copies of C as part of S. This grant does not have a conjunct which imposes conditions on the license of S. Therefore, it is possible for I to distribute copies of S under a proprietary license. However, the grant has other conjuncts which impose conditions on the availability of the source code of C and the ease of identifying C as a separate executable program.

3.3. Reusing components made from components

Frequently a component C—used by S—is created from other components as a derivative or collective work of these components. As previously discussed, I requires a license from the licensor of C. To be valid this license should satisfy all the conditions of each of the sub-components $c_1..c_m$ of C; if the conditions of one of them (say c_i) is not satisfied (either by mistake or on purpose) then the creator of C might not have a license to C, and I does not have a license to use C as part of S. Under this scenario I might be liable for copyright or patent infringement, even if I was never aware of how and why such license was not honoured inside one of the components C_i of S. For this reason I must be aware of all components and sub-components included explicitly, or implicitly in S and the licensing requirements of these components.

3.4. Compatibility of licenses

As mentioned before, each of the rights needed for S will require a grant to one or more specific rights from each of the $C_1...C_n$. Each grant imposes a set of conditions, which are modeled as conjuncts. If the union of all these conjuncts is not satisfiable, then I cannot acquire the desired rights and might not be able to create S or license it to anybody. We say that, for a given grant g, $\mathbb{L}(C)$ is not compatible with $\mathbb{L}(S)$ if at least one of the conjuncts of g from $\mathbb{L}(C)$ is not satisfiable under $\mathbb{L}(S)$. We denote this relationship as $\succ_g$ (compatible) and $\not\succ_g$ (not compatible): if under grant g, $\mathbb{L}(A)$ is compatible with $\mathbb{L}(B)$ then $A \succ_g B$. By extension the use of C in S is compatible under use u (denoted $C \succ_u S$) iff the use u is permitted by a grant g of $\mathbb{L}(C)$, and $\mathbb{L}(C) \succ_g \mathbb{L}(S)$.

To illustrate these concepts, let us assume that $S \in \Delta(C)$, $\mathbb{L}(S) = MPL_{1.1}$, and $\mathbb{L}(C) = GPL_2$. The condition *any derivative work of C should be licensed under the GPL_2* (modeled as the conjunct $\forall A \in \Delta(C), \mathbb{L}(A) = GPL_2$) is unsatisfiable in this particular case. We therefore say that under the grant of *creation and distribution of derivative works* (δ) the GPL_2 is incompatible with the $MPL_{1.1}$, $GPL_2 \not\succ_\delta MPL_{1.1}$. For this particular grant we will omit the subscript $GPL_2 \not\succ MPL_{1.1}$. Similarly, when C is used to create a derivative work S that is to be distributed (its use is υ) $C \not\succ_\upsilon S$, (for this use we omit υ and write $C \not\succ S$).

For other grants the GPL_2 is not necessarily incompatible with other licenses. We will denote as ρ the grant to create and run a derivative work but not distribute it. The GPL_2 imposes no condition for ρ if S is never distributed. We model this condition as the conjunct *use in-house only*. In other words, an integrator can create any derivative work from GPLed components as long as the resulting system is used in-house only. One of the conditions of the Microsoft Public License (Ms-PL) is that the user must accept the license before the software is used. We model this condition of ρ as the conjunct "accept license". Another condition of the (Ms-PL) is that the license of the derivative work is the Ms-PL exclusively. We model this conjunct of the ρ as "$\mathbb{L}(S) = Ms\text{-}PL$". Assume S is a derivative work of two components G and M, $\mathbb{L}(G) = GPL_2$ and $\mathbb{L}(M) = Ms\text{-}PL$, and S is never going to be distributed so it will only be used in-house. Under this scenario the grant needed is ρ. The conjunct of such grant from the GPL_2 is always satisfied ("only use in-house"); simultaneously it is possible to satisfy the conjuncts of ρ from the Ms-PL (the user "accepts license", and $\mathbb{L}(S) = Ms\text{-}PL$). S will contain code licensed both under the GPL_2 and the Ms-PL without any copyright infringement: $\forall D \in \Delta(M), G \succ_\rho D$, and $GPL_2 \succ_\rho Ms\text{-}PL$. However, it is impossible for the integrator to distribute S. If $S \in \Delta(G)$ then one conjunct of this grant (δ, create and distribute derivative works) under the GPL_2 is "$\mathbb{L}(S) = GPL_2$", and if $S \in \Delta(M)$ then one of Ms-PL δ conjuncts is "$\mathbb{L}(S) = Ms\text{-}PL$". If $\mathbb{L}(S) = GPL_2$ then the conjuncts of the Ms-PL are unsatisfiable ($Ms\text{-}PL \not\succ GPL_2$). Likewise, if $\mathbb{L}(S) = Ms\text{-}PL$ then the conjuncts of δ from GPL_2 are unsatisfiable ($GPL_2 \not\succ Ms\text{-}PL$). A derivative work of both G and M cannot be distributed, though the work can be used in-house.

4. A system of patterns for interconnecting open source components

Open source and proprietary applications incorporate components with different licenses, and frequently the resulting application has a different license. Under which circumstances could a software application be *legally* created with components with different licenses (the license-mismatch problem)? To answer this question we needed to know: a) the components used by an application; b) the licenses of such components and the application; c) the technique used to address and resolve the license mismatches, if such mismatches exist. Our goal is to identify a system of patterns that encapsulates the methods used to solve the license-mismatch problem.

4.1. Methodology

We examined the licensing for 124 OSS projects. We started from the following eight popular software applications: Apache version 2.2, GIMP version 2.2.0, MySQL 5.0.38, Koffice 1.6.3, GNOME desktop 2.14.0, GCC 4.3.2, PostgreSQL 8.2.4, and Bugzilla 3.0.2. We then built a dependency graph which recursively showed all the packages which these applications depend on. We determined the packages and dependencies by using the open source packaging systems which is used to ease the installation of open source packages [27, 36]. Our method is described in detail in [9]. In particular, we used the information from the packaging systems of Debian 4.0 and Fink 0.8.1 to create the interdependency graphs of these eight applications.

To fully understand the licensing terms of each of these software packages, we downloaded their source code packages and manually inspected their documentation. We documented their license and any peculiarities in their licensing terms.

Many packages contained a file in its source code that described in detail the license. For example, the *netpbm* project listed every single file and the license under which it was being offered–a total of 9 licenses were mentioned. Sometimes identifying the type of license was not trivial. For example, the MIT and the BSD licenses are based upon templates that must be filled-in with the name of the copyright owner and her organization, and frequently the licensors further edit the license. Occasionally one of the resulting licenses becomes known by the name of the project that uses it. For instance, *libJPEG* uses a license that appears to be derived from the *original BSD*, with other projects, like *netpbm*, refering to that license as the libJPEG license.

The licensing terms of packages vary significantly. Some packages, such as PostgreSQL, have only one license; while others, such as GCC, have different licenses for different parts. In the case of GCC, the C runtime library, known as glibc, is licensed under the $LGPL_{3+}$ (a + after a version number of a license indicates that the licensor allows the licensee to choose a newer version of the license) while the programs, including the compiler, are licensed under the GPL_{3+}.

Freq	Name of License	Version	Abbrev.
2	Aladdin		
2	Apache	2.0	AL_2
2	Artistic	1.0	$ArtL_{1.0}$
15	BSD style	New	*new BSD*
6		Original	*original BSD*
4		Other	
1	General Public	1+	GPL_{1+}
12		2	GPL_2
31		2+	GPL_{2+}
1		3+	GPL_{3+}
4	Library General	2	$LGPL_2$
37	Public	2+	$LGPL_{2+}$
4	Lesser General	2.1	$LGPL_{2.1}$
11	Public	2.1+	$LGPL_{2.1+}$
1		3+	$LGPL_{3+}$
16	MIT/X11-style		MIT
5	Mozilla Public	1.1+	$MPL_{1.1+}$
17	Other		
2	Public domain		

Table 1. Licenses found in the 124 studied OSS packages. The first column lists the number of packages using a license. A + after a version number of a license indicates that the licensor allows the licensee to choose a newer version of the license. The last column shows the abbreviations used in this paper. The MIT/X11 and the BSD are templates, and each instance is expected to be different from others (e.g., the name of the copyright owner will differ). The *original BSD* is also known as the 4-clauses BSD, and the *new BSD* as the 3-clauses with one clause– the so-called "advertising clause" removed.

Sometimes a package is explicitly licensed under the terms of more than one license. For example, Perl lets the user choose between the terms of the original Artistic License, or the terms of the GPL_{1+}. We found many licenses that were not OSI-approved; some of them were complex, while others very simple. For example the "Beer-ware" is a one-paragraph long license which asks the user to buy beer to the author if she likes the software. Table 1 summarizes the licenses found, and their frequency of occurrence.

Once we determined the license of all the components, we proceeded to identify any license mismatches. When there was a license mismatch, we documented the rationale for allowing it. In a few cases, we contacted authors of the packages seeking clarification. It was clear that most software package maintainers were concerned with the licensing issues surrounding the use of their packages, and how they used other packages.

Not surprisingly, a few common methods, used to address the license-mismatch problem, have emerged over the years and across projects.

We identified 12 patterns, and classified them into two types: patterns for the licensor (the creator of the component), and patterns for the licensee (the integrator or user of the component). These are summarized in table 2. Due to space restrictions we only present 4 patterns.

4.2. Patterns for Licensor

Exception

Intent: To allow a particular use by expanding the terms of the license in an addendum, without modifying the text of the license itself.
Motivation: The copyright owner of a component wants to allow its use in a situation that is incompatible with its license. Rather than modifying the license (or using a different license), the copyright owner issues an **exception** that explicitly states certain extra conditions under which it will allow such use.

Applicability:

- A potential derivative work D of the product P cannot be created because the licensing terms of D are incompatible with the license of P. Yet, the copyright owner of P wants this derivative work to exist, but does not want to re-license P under al compatible license. This is a common problem in the FOSS world, where many licenses are not compatible among them for the grant of creation and distribution of derivative works, even if they have similar philosophical goals.
- A complementary situation arises when a product P needs to use a component C and the license of C is incompatible with the license of P (usually due to few, minor clauses that are not satisfiable by the license of P, but that the copyright owner of P is willing to satisfy). The copyright owner of P issues an exception to P's license making it legal for P to use C.

Advantages:

- It facilitates the integration of modules with otherwise incompatible licenses.
- It avoids the need to modify the original license of the component (avoiding the proliferation of versions of the license).

Disadvantages:

- Only the copyright owner is capable of granting an exception. If more than one owner, then each one of them should consent to the exception.
- The exception might create a legal loophole with unintended consequences.

Known Uses:

- **Trolltech GPL Exception.** Trolltech distributes several products under the GPL_2 and a commercial license. Aware that the GPL_2 is incompatible with many FOSS licenses, Trolltech has issued its GPL Exception [35]. It explicitly allows linking of its libraries by software products released under 22 different open source licenses. The main goal of this exception is to allow reuse of its libraries by as many OSS as possible.[1]
- **MySQL AB FLOSS License Exception.** This exception is very similar in objectives to the Trolltech GPL Exception. It allows linking to its MySQL Client Libraries by software products released under 23 different open source licenses (the same 22 licenses listed under the Trolltech GPL Exception plus one more). Although the *MySQL AB FLOSS License Exception* and the *Trolltech GPL Exception* have very similar objectives, they are drafted in very different terms. Trolltech's exception addresses and permits linking, while MySQL's addresses the issue of what constitutes a derivative work[22].
- **OpenSSL GPL Exception.** Any program licensed under the GPL that links to the OpenSSL library requires this exception. OpenSSL is a cryptographic implementation of the SSL and TLS protocols. OpenSSL is FIPS 140-2 compliant, an important requirement for certain organizations which use cryptographic software, making it desirable for applications to link to it [23]. OpenSSL is released under the terms of both the OpenSSL License and the SSLeay License. These licenses are incompatible with the GPL_2+ under the grant of creation and distributions of derivative works. Without this exception a program licensed under a GPL license would not be capable of linking to the OpenSSL [40]. The Free Software Foundation has published guidelines that describe how this type of exception should be worded (see [6]). *wget* and *climm* are two products that use this exception.
- **Java Classpath exception.** Until recently Sun distributed its Java JDK under the Common Development and Distribution License (CDDL), an OSI-approved license that is not compatible with the GPL_2 under the grant of creation and distribution of derivative works. Sun wanted to increase the potential use of Java, and decided to change the license of the JDK to the GPL_2. But there was a problem:

[1] In 2008 Trolltech was acquired by Nokia, and in January of 2009 Nokia change the license of QT, its flagship product, to the $LGPL_{2.1}$.

Type	Name	Intent
Licensor	Exception	To allow a particular use by expanding the terms of the license in an addendum, without modifying the text of the license itself.
	Disjunctive	To give the option to the licensor to choose one of several licenses that will best suit her purpose.
	Clarification	Give an interpretation of contentious or ambiguous parts of the license.
	Permit Relicensing	Allow the derivative work to be licensed under a different license than the one under which the component is made available.
	Add-on	Allow components under a non-compatible license to extend the functionality of another component via a well-defined API.
	Indirect License	A component indicates that its license will be the same as another one.
	Different parts, different licenses	Provide different parts or features of the system under different licenses.
Licensee	Patch	Issue a patch that the user can apply to the component to create the derivative work.
	Component with Compatible License	Find a component that can be licensed in a manner that is compatible with the intended use.
	Create collective	Make sure the work is considered a collective that includes the component.
	Ask for exception	Request the licensor to give you an exception to one or more conditions imposed by the license. Results in the Exception Pattern, above.
	Ask for clarification	Request the licensor to clarify her interpretation of any ambiguous or contentious parts of the license. Results in the Clarification Pattern, above.

Table 2. Identified patterns to address the *license-mismatch* problem.

any program that runs under the Java Virtual Machine (JVM) dynamically links to the runtime library. The runtime library is part of the JDK, and would end up licensed under the GPL_2. As a consequence any program running under the JVM would need to be licensed under the GPL_2. To avoid this issue Sun added the Classpath exception to the GPL_2. This exception, authored by the Free Software Foundation, explicitly states that linking to the provided library (part of the JDK) is not considered a derivative work [7, 30].

Modeling a pattern

We demonstrate the usefulness of the model and concepts presented in section 3.2 by describing this pattern. Due to space limitations, we omit this demonstration for other patterns.

Assume S uses component P, g is the grant required on P to create S and $\mathbb{L}(P) \not\succ_g \mathbb{L}(S)$, hence there exists one or more conjuncts in g that are not satisfiable. The copyright owner of C adds exceptions to one or more clauses of $\mathbb{L}(P)$ to remove these conjuncts (and likely add new ones). The new set of conjuncts for the grant g becomes satisfiable.

In the case of the *Trolltech GPL Exception*, Trolltech produces QT, a multi-platform GUI library, and $\mathbb{L}(QT) = GPL_2$. An application S that wants to use QT has two alternatives: 1) to copy the source code of QT (potentially modifying it) or link to it. As we have seen above both methods are possible if $\mathbb{L}(S) = GPL_2$. But QT wants open source projects under 22 other licenses (we denote this set as OSS) to be able to link to QT as long as they do not modify it. One solution would be to provide QT under each of these licenses, but some of these licenses are too permissive from the point of view of Trolltech. For example, the *new BSD* will not only allow linking, but also the modification and distribution of the library under commercial licenses, undermining Trolltech's business model. Trolltech solves this problem by creating an exception to the grant of distribution of derivative works of the GPL_2: if $\mathbb{L}(S) \in OSS$ and the interconnection type is linking then S then one can use QT in S, otherwise the $\mathbb{L}(S) = GPL_2$. The conjuncts of this grant have become more complex, but they are satisfiable under these circumstances.

Disjunctive

Other names: Dual licensing, tri-licensing.
Intent: To give the option to the licensor to choose one of several licenses that will best suit her purpose.
Motivation: The licensor has two or more groups of users and each requires a different incompatible license.

Applicability: The licensor wants both groups to use the component, but writing a license to satisfy them simultaneously might be error prone or even impossible, for instance when one license is incompatible with the other.

Advantages:

- Each license will provide different rights, benefits and restrictions to the licensor, allowing the integrator to choose the license that best fits her uses (and ignore the others).
- It is used as the basis of a business model based for open source software where at least one of the licenses to the software is proprietary and another is open source.
- It avoids the use of a more complex license.

Disadvantages:

- The licensor and the licensee need to understand the differences between each license.
- The legal repercussions of accepting large external contributions to the product (e.g., a patch to fix a defect, or to add a new feature) need to be carefully evaluated.

Known uses:

- **Perl.** Perl is licensed under the terms of the GPL_{1+}, or the Artistic License (original version).
- **Mozilla Core.** The Mozilla Core project is licensed under three different licenses: $MPL_{1.1+}$, GPL_{2+}, and $LGPL_{2.1+}$. According to the project, its main motivation for this licensing scheme is to allow others to use its code in as many projects as possible. [20]
- **MySQL and QT.** Both are offered under the GPL_2 and various commercial licenses.

Clarification

Intent: Give an interpretation of contentious or ambiguous parts of the license.

Motivation: The terms of a license might be confusing or ambiguous, leading to legal uncertainty by its potential users. The licensor of the product issues a **clarification** of the terms of the license to address this uncertainty.

Applicability: It is not uncommon for software licenses to have terms and conditions that might be ambiguous, confusing, or potentially misunderstood. This could lead to different interpretations of the same license by the licensee and licensor. For example, the definition of derivative work is not well-defined–see section 2.2. The Clarification allows the licensor to explicitly state its understanding of one or several parts of the license.

Advantages:

- It makes clear the intention of using such a license.
- The licensee knows, in advance, the interpretation that the licensor has of the licensing terms.

Disadvantages:

- A clarification might be contradictory to the license itself with licensors often issuing clarifications without legal advice.
- The interpretation of the licensee is biased and might be overreaching or contradictory to IP law.

Known uses:

- **Linux GPL clarifications.** Linus Torvalds, the main copyright owner of the Linux kernel, has stated that programs that only use the services of the kernel are not considered derivative works of the kernel [33]. In another clarification, he stated that the files of the kernel for which he owns the copyright are released under the GPL_2 only, even though the files do not explicitly state a license—as the FSF recommends–instead he only includes a COPYING file with the license in the distributions of the kernel. [34].
- **Perl GPL clarification.** Larry Wall, the original author and copyright owner of Perl, includes a clarification to Perl's license. Perl is licensed under a disjunctive license–GPL_{1+} or Artistic, see the Disjunctive pattern in page 8.The Perl GPL clarification states "my interpretation of the GNU General Public License is that no Perl script falls under the terms of the GPL unless you explicitly put said script under the terms of the GPL yourself." [39].
- **Eclipse clarification.** As long as a plug-in for Eclipse uses the plug-in API, without any modifications, to communicate with Eclipse, the plug-in does not create a derivative work of Eclipse and can be licensed under any terms [5].

4.3. Patterns for Licensee

Patch

Intent: Issue a patch that the user can apply to the component to create the derivative work.

Motivation: To modify a component without having to publish a derivative work.

Applicability: This pattern applies in two scenarios. In the first the licensor of the main component does not allow redistribution of a particular derivative work. In the second, the integrator is not interested in creating and maintaining a derivative work of the original. In such cases, the integrator decides to issue its modifications as a "patch" that can be applied, either in binary or source code form to the original component.

Applicability: This pattern is useful when a third party is interested in providing a feature not in the product, and yet, does not want to (or cannot) redistribute the modified product.

Advantages:

- It circumvents the need to distribute a derivative work.
- It permits the modification of the original component without any approval of its copyright owner, although it might be restricted by other laws; see disadvantages.
- It can be performed in source code or binary form.
- For open source projects, patches allow the testing of new and experimental features before they are integrated into the main product. A patch might be later integrated into the product.
- A patch can have a different license than the original product.

Disadvantages:

- A new version of the original product might render the patch ineffective, and a new patch might need to be created.
- The end-user is expected to be capable and willing to apply the patch.
- In some cases, the patch might be considered illegal (for example, if the patch is considered–in the USA–a circumvention device under the DMCA).

Known uses:

- MySQL AB accepts some contributions for its mysqMySQLl database, but it does not accept all of them. The authors of those accepted contributions must sign a copyright transfer to MySQL AB [21]. The *Google MySQL Tools Mailing List* maintains a repository of patches for features that are either not accepted or have not been submitted to MySQL AB but some users find useful [12].
- Trolltech's QT was originally released under the Q Public License Version 1.0. This license is OSI-approved, and does not permit the distribution of derivative works, but it allows the distribution of patches. Trolltech wanted to restrict the possibility that other companies would create commercial derivatives of their library, and wanted to maximize their chance to sell licenses. Trolltech stopped using this license when it released version 4.0 of QT under the GPL_2 (see [15]).

5. Conclusion and future work

The license-mismatch problem poses many challenges to developers of modern software application. With the rapid growth of open source components with varying and conflicting licenses, developers have created methods to address this problem. We manually analyzed over 124 OSS packages on which several popular open source applications such as Bugzilla, Apache, and GIMP depend. Based on our study we identified several patterns which permit the interconnection of components with conflicting licenses while complying with all licenses.

The basic model presented in this paper shows the potential benefits of formally describing licenses. We believe our work tackles an important yet rarely investigated aspect of building large component-based software applications. With the widespread and easy access to open source components online, the need for practitioners to have a good understanding of the legal implications of re-using particular components is becoming extremely important and vital for the success of large software projects.

Acknowledgements

We are greteful to Bob Gobeille from the FOSSology Project, and Jesús González-Barahona from the University Rey Juan Carlos for helpful comments and previous conversations, and to the European Legal Network Mailing List members for their always enlightening discussions. The work of D.M.German has been funded by Hewlett-Packard to support the FOSSology Project.

References

[1] M. Bayersdorfer. Managing a project with open source components. *interactions*, 14(6):33–34, 2007.

[2] A. Becerman-Rodau. Protecting Computer Software: after Apple Computer Inc. v. Frankin Computer Corp., 714 F.2d 1240 (3d Cir. 1983) does copyright provide the best protection? *Temple Law Review*, 57(527), 1984.

[3] J. Bhuta, C. Mattmann, N. Medvidovic, and B. W. Boehm. A Framework for the Assessment and Selection of Software Components and Connectors in COTS-Based Architectures. In *WICSA*, page 6, 2007.

[4] Y. B. Dang, P. Cheng, L. Luo, and A. Cho. A code provenance management tool for IP-aware software development. In *ICSE Companion '08: Companion of the 30th international conference on Software engineering*, pages 975–976, New York, NY, USA, 2008. ACM.

[5] Eclipse Foundation. Eclipse Public License (EPL) Frequently Asked Questions, 2007. Accessed Dec. 2007.

[6] Free Software Foundation. What legal issues come up if I use GPL-incompatible libraries with GPL software? `http://www.gnu.org/licenses/gpl-faq.html`, 2007.

[7] Free Software Foundation. GNU Classpath. `http://www.gnu.org/software/classpath/license.html`, 2008. Accessed Sept. 2008.

[8] D. Garlan, R. Allen, and J. Ockerbloom. Architectural Mismatch or Why It's Hard to Build Systems Out Of Existing Parts. In *ICSE*, pages 179–185, 1995.

[9] D. M. German, J. M. González-Barahona, and G. Robles. A model to understand the building and running

inter-dependencies of software. In *"Proc. 14th Working Conference on Reverse Engineering"*, pages 130–139, 2007.

[10] R. Gobeille. The FOSSology project. In *MSR '08: Proceedings of the 2008 International Conference on Mining Software Repositories*, pages 47–50, New York, NY, USA, 2008. ACM.

[11] P. Goldstein. *International Copyright: Principles, Law, and Practice*. Oxford University Press US, 2001.

[12] Google MySQL Tools Mailing List. Patches for MySQL 5. `http://code.google.com/p/google-mysql-tools/wiki/Mysql5Patches`.

[13] Jacobsen v. Katzer, No. 2008-1001 (Fed. Cir. 8/13/2008). U.S. Court of Appeals for the Federal Circuit, 2008.

[14] S. Lai. *The Copyright Protection of Computer Software in the United Kingdom*. Hart Publishing, 2000.

[15] A. M. S. Laurent. *"Understanding Open Source and Free Software Licensing"*. O'Reilly, 2004.

[16] T. Madanmohan and R. De'. Open Source Reuse in Commercial Firms. *IEEE Software*, 21(6):62–69, 2004.

[17] D. McGowan. Legal Aspects of Free and Open Source Software. In J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, editors, *Perspectives on Free and Open Source Software*, pages 211–226. MIT Press, 2005.

[18] N. J. Mertzel. Copying 0.03 percent of software code base not "de minimis". *Journal of Intellectual Property Law & Practice*, 9(3):547–548, 2008.

[19] M. Morisio, editor. *Reuse of Off-the-Shelf Components, 9th International Conference on Software Reuse, ICSR 2006, Turin, Italy, June 12-15, 2006, Proceedings*, volume 4039 of *Lecture Notes in Computer Science*. Springer, 2006.

[20] Mozilla Foundation. Mozilla Relicensing FAQ Version 1.1. `http://www.mozilla.org/MPL/relicensing-faq.html`, Aug. 2008. Accessed Aug. 2008.

[21] MySQL AB. MySQL Contributor License Agreement v0.3. `http://forge.mysql.com/contribute/cla.php`. Accessed Sept. 2008.

[22] MySQL AB. MySQL AB FLOSS License Exception. `http://www.mysql.com/company/legal/licensing/foss-exception.html`, March 2007. Accessed Dec. 2007.

[23] National Institute of Standards and Technology. Validated FIPS 140-1 and FIPS 140-2 Cryptographic Moduples 2007. `http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401val2007.htm`.

[24] M. B. Nimmer and D. Nimmer. *Nimmer on Copyright*. Matthew Bender & Company, 2002.

[25] Z. Obrenovic and D. Gasevic. Open Source Software: All You Do is Put it Together. *IEEE Software*, 24(5):86–95, 2007.

[26] R. C. Osterberg. *Substantial Similarity in Copyright Law*, chapter 8. Practising Law Institute, 2003.

[27] G. Robles, J. M. Gonzalez-Barahona, M. Michlmayr, and J. J. Amor. Mining large software compilations over time: another perspective of software evolution. In *MSR '06: Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 3–9, New York, NY, USA, 2006. ACM Press.

[28] L. Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall, 2004.

[29] C. Ruffin and C. Ebert. Using open source software in product development: a primer. *IEEE Software*, 21(1):82–86, 2004.

[30] Sun Microsystems. Free and Open Source Java. `http://www.sun.com/software/opensource/java/faq.jsp`, 2008. Accessed Sept. 2008.

[31] The Mozilla Organization. Bugzilla. `www.bugzilla.org`, 2008. Accessed Sep. 2008.

[32] The Open Source Initiative. The Open Source Definition. `http://opensource.org/docs/osd`, 2006.

[33] L. Torvalds. Note to the GNU General Public License. ./COPYING file in the Linux kernel version 2.6.23. Accesed Dec. 2007.

[34] L. Torvalds. Re: GPL V3 and Linux - Dead Copyright Holders. `http://lkml.org/lkml/2006/1/27/339`, Jan 2006.

[35] Trolltech ASA. Trolltech GPL Exception version 1.0. `http://trolltech.com/products/qt/gplexception`. Accessed Dec. 2007.

[36] C. Tucker, D. Shuffelton, R. Jhala, and S. Lerner. Opium: Optimal package install/uninstall manager. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 178–188, 2007.

[37] United States Copyright Office. Circular 92 Copyright Law of the United States of America and Related Laws Contained in Title 17 of the United States Code, June 2003.

[38] United States Copyright Office. Circular 14 Copyright Registration for Derivative Works, June 2008.

[39] L. Wall. Perl Kit Version 5. ./README file in Perl version 5.6.10, available at cpan.org. Accesed Dec. 2007.

[40] Wikipedia. OpenSSL. `http://en.wikipedia.org/wiki/OpenSSL_exception#GPL_exception`. Accessed Dec. 2007.