Managing Open Source Legality Concerns – A Sustainability Catalyst

Alexander Lokhman, Salum Abdul-Rahman, Antti Luoto, Imed Hammouda
Department of Software Systems
Tampere University of Technology
Tampere, Finland
firstname.lastname@tut.fi

Abstract. As more and more software companies are integrating different Free/Libre and open source software (FLOSS) components in their products, it became more probable that a single software solution uses numerous licenses. Mixing together different open source and proprietary licenses may lead to legality complications as different licenses introduce different privileges and requirements on the use of the composed code. In this paper, we address the multi-facets of the legality concerns of open source. We further propose an open tool architecture to address such concerns.

1 Introduction

Over the last decades, Free/Libre and Open Source Software (FLOSS) has emerged as one of the most important phenomena in software engineering. In this trend, more and more companies are putting FLOSS at the center of their business strategies. Although there are many benefits to going open source, companies need to be aware of the risks associated with FLOSS. One of such risks is the legal obligations that both consumers and producers of FLOSS need to fulfill. Unfortunately, for many companies, software developers are still unaware of these issues. This may cause trouble to the corresponding companies, especially in the absence of legal departments and external legal consultants.

In this position paper, we address the various facets of open source legality compliance, arguing that the legal risks of open source have a critical influence on the sustainability of the open source movement as a whole. We further argue that handling the legality risks through shared knowledge bases and automated tools may boost the adoption of open source. Towards the end of the paper we briefly present an open tool architecture for open source legality compliance.

2 Legality Tension of FLOSS intensive systems

When addressing the legality compliance issue of FLOSS intensive systems, there are a number of factors that must be taken into account. These factors not only stem from the nature and terms of the licenses themselves, but also are related to the way the subject software is implemented, packaged, and deployed.

There are plenty of licenses and license models. A straight forward observation when working with open source licenses is that there are many of them. The Open Source Initiative [OSI] lists about 70 licenses. Popular licenses include the GNU General Public License (GPL), the Lesser GNU General Public License (LGPL), the Apache license, the Massachusetts Institute of Technology license (MIT), and the Berkeley Software Distribution license (BSD). The terms of different licenses vary considerably. To give an example, some licenses such as MIT are classified as permissive, granting very broad rights to licensees and allowing almost unlimited use of the licensed code. Other licenses such as GPL are classified as strong copyleft, requiring that works based on the licensed code be published and relicensed to others on the same terms of the initial license. In the middle are weak copyleft licenses such as LGPL, which is a compromise between permissive and strong copyleft. The LGPL grants flexibility to users when linking to licensed software libraries. However, any modifications to the original library should be contributed back on the same terms of the license. Moreover, some licenses have several versions, and there are subtle changes between different versions. A good example is the case of GPL v2 and GPL v3. In addition, the list is by no means complete, and new licenses can be introduced if so desired. For example, a new license can add some minor differences to an earlier one, thus generating a discrepancy between the licenses, or a completely new license can be introduced.

Licenses can be conflicting [Ham10]. To give an example of possible legal incompatibilities between software components, Table 1 presents a number of open source licenses and their compatibility properties (across open source components themselves) categorized into three cases: mixing and linking is permissible, only dynamic linking is permissible, and completely incompatible.

Table 1. Example Open Source Licenses and their Compatibility

	PHP	Apache	IPL	SSPL	Artistic
GPL	3	3	3	1	3
LGPL	2	2	2	1	2
BSD	1	1	1	1	1

1- Mixing and linking permissible

2- Only dynamic linking is permissible

3- Completely incompatible

As an example, a software component under the terms of GPL cannot be directly linked with another under the terms of the Apache license. In this case, the main reason is that GPL’ed software cannot be mixed with software that is licensed under the terms of a license that imposes stronger or additional terms, in this case the Apache license. The Apache 2.0 license allows users to modify the source code without sharing modifications, but they must sign a compatibility pledge promising not to break interoperability.

Is it derived or combined work? When integrating third party open source components, possibly together with own work, the restrictions and obligations which the used licenses impose may depend on whether the work is considered as derived (derivative) or combined (collective) [Ger09]. A simple example of derived work is a modified version of the original software. However, the distinction between derived and combined works becomes trickier when producing new work by combining or linking multiple software components, possibly distributed under the terms of different licenses. Take the example of a software system *S* which is the result of linking together an open source component *C1* and an own developed component *C2*. A common interpretation is that system *S* is considered to be derived work if *C1* and *C2* link statically (linked during compile or build time) and that S is considered to be combined work if *C1* and *C2* link dynamically (the two libraries are loaded into a client program at runtime). In a typical case, however, only a judge in a court of law can make the final decision. As a matter of fact, the court decision might depend on the specific legal framework of the jurisdiction in which the case arises.

There are thousands of open source components with different risk levels depending on their usage scenario. The number of open source components has grown at an exponential rate during the last decade. This has given software developers a jump on creating software based on existing code. However, many companies are reluctant to use open source software due to the legal risks associated with the use of those components. There have been attempts to classify open source components according to their risk level [Wil10]. Table 2 gives an example categorization. Four usage scenarios are identified: using the component as a redistributable product, as part of service offering, as a development tool, and for internal use. Three levels of risks have been proposed.

Table 2. Example Software Components and their Risk Level

Component	License	Redistribution	Service offering	Development tool	Internal use
Agent++	Agent++ license	3	3	2	1
SwingX	LGPL	3	3	3	3
Libxml2	MIT	1	1	1	1
Cglib	Apache	2	1	1	1

(1) Valid (2) Possible risk (3) Clear risk

According to the authors of [Wil10], valid means that the package can be used as instructed and that no risk has been identified. Possible risk means an interpretation question has been found. This type of issues can be solved by either 1) removing/replacing the problematic files or 2) acquiring additional permissions from the respective right holder or 3) not using the package at all or 4) based on the particular company's risk preferences in such project, a company could accept the risk. Legally, an interpretation question means that an eventual realizing risk would be civil law risk, e.g. monetary (not criminal). Clear risk means that a risk that cannot be interpreted in a way that would not include the risk has been found. This type of issues can be solved only by 1) removing/replacing the problematic files or 2) acquiring additional permissions from the respective right holder or 3) not using the package at all. A company normally cannot accept this type or risk, since it means the possibility of not only civil law risks, but criminal risks.

As an example, component Agent++ can be used internally with no risk, has a possible risk when used as a development tool, but exhibits a clear risk when used as part of service offering or a redistributable product.

Open Source legality interpretations are subject to the way software is implemented, packaged, and deployed [Ham10, Mal10]. The legality requirements imposed by FLOSS licenses, such as the requirement to publish source code (i.e. the copyleft rule of GPL), may depend for instance on the interaction type of the components (data-driven versus control-driven communication). In the case of mere data exchange between components, there is no copyleft obligation as the two components are considered as separate programs. Also, the copyleft obligation of GPL does not hold if the FLOSS component (or a modified version of it) is deployed as a hosted service. However, if the hosted code is licensed under the terms of AGPL (Affero General Public License), the copyleft requirement does hold, but only in the case of user interaction with the hosted service (in contrast to service to service interaction). In addition, the copyleft requirement of GPL may not hold in case of interactions through standardized interfaces such as the use of operating system public API, in contrast to system hacks which make the two communication components strongly coupled. Finally, compatibility concerns among different licenses may be circumvented if the packaging of components is done by the user instead of building the entire system at the vendor site.

3 Towards an Open Architecture for FLOSS Compliance

The ultimate goal of this work is to design and implement a new kind of tool for addressing the various legality compliance concerns identified in the previous section.

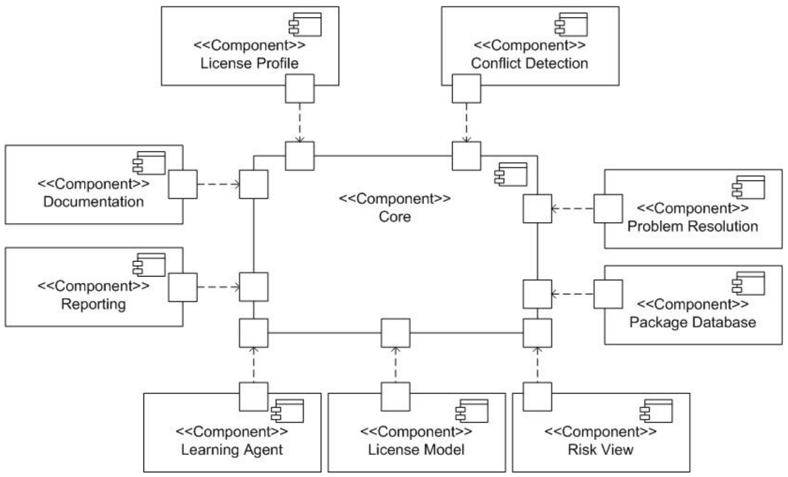

Figure 1. An Open Architecture for Open Source Compliance

Figure 1 proposes an example overall architecture for such a tool. Here we assume that the tool is capable of managing the legality concerns at the architectural level (i.e., application design is expressed as an UML component diagram for example). Table 3, in turn, explains each of the architectural components and lists example existing works that could be used as implementation guides.

Table 3. Architectural Components

Component	Description	Resource
Core	Handles interactions between the application model, licensing information and the user.	[Wil10]
License Profile	A UML extension to include license information.	[SPDX], [Hoe07], [OSI]
License Model	Describes in computable format the clauses, restrictions, rights and their interdependencies of a license.	[Als09], [Tuu09], [Hoe07], [Gom08]
Package Database	A repository of containing information on which license and copyright information is associated with which package.	[SF]
Risk View	Assess legal risks related to use of component for variable purposes re-licensing, sale, internal use etc.	[Als09], [Hoe07], [Gom08]
Conflict Detection	Analysis whether license terms of different licenses conflict when linked into the same software.	[Ham10], [Als09], [Tuu09], [FOS10], [OSLC], [Ninka]
Problem Resolution	Suggests operations that can be performed to remove license conflicts from model.	[Ger09], [Ham10], [Mal10]
Learning Agent	Records user actions so that they can be later used to improve program performance.	[Ham10]
Reporting	The analysis results from the different components can be output in different formats.	[FOS10], [Tuu09], [OSLC]
Documentation	Linking to internal and external documentation on open source licensing concerns.	[IFOSS]

A part from *Core*, each component is associated with an extension point. The architecture is made extensible so that the tool is able to work with different licenses. The *License Profile* component allows for attaching different licensing concepts to the architectural model. Different implementations of *License Model* give different interpretations of clauses based on local law. Different open source components can be registered to the tool via the *Package Database* component. The *Risk View* extension point allows the plug-in of different risk analysis methods. The tool also integrates different techniques for detecting conflicts among licensed components (*Conflict Detection*) and proposes remedial actions (*Problem Resolution*). These actions can be recorded for future exploitation (*Learning Agent*). Finally, the tool is capable to report the analysis results in different pluggable formats (*Reporting*) and links to relevant documentation resources (*Documentation*). We argue that the described architecture allows the building of an open knowledge base related to open source licensing.

4 Conclusions

There has been a growing interest in studying the compliance of software systems with respect to the legality restrictions and obligations of open source licenses. This came in response to the increasing concerns about the legal risks of using FLOSS components. We argue that if such issues are not addressed by both legal experts and software developers, the whole open source ecosystem may face sustainability challenges. In this paper we have presented an overview of the main dimensions involved in open source compliance. Based on the analysis, we have outlined an open architecture for managing open source legality concerns at the architectural level. As future work, we plan to exploit the ideas presented in this paper to develop concrete tool infrastructure.

References

[Als09] Alspaugh, T. A., Asuncion, H. U. and Scacchi, W. Analyzing Software Licenses in Open Architecture Software Systems. In proc. of FLOSS 2009, pp 54-57.

[FOS10] FOSSology. http://fossology.org/. Last accessed Sep. 2011.

[Gam05] Gamma, E.,Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison_Wesley, 1995.

[Ger09] German, D. M.; Hassan, A. E. License integration patterns: Addressing license mismatches in component-based development. In proc. of ICSE 2009, pp 188-198. May 2009.

[Ham10] Hammouda, I., Mikkonen, T., Oksanen, V. and Jaaksi, A. Open Source Legality Patterns: Architectural Design Decisions Motivated by Legal Concerns. In proc. of AMT 2010. Tampere, Finland. October 2010. ACM Press.

[Hoe07] Hoekstra, R., Breuker, J., Di Bello, M. and Boer, A. The LKIF Core Ontology of Basic Legal Concepts . In proc. of LOAIT 2007, pp 43-63.

[IFOSS] International Free and Open Source Software Law Review. http://www.ifosslr.org. Last accessed Sep. 2011.

[OSI] Open Source Initiative. http://www.opensource.org. Last accessed Sept. 2011.

[OSLC] OSLC, Open Source License Checker. http://sourceforge.net/projects/oslc. Last accessed Sept. 2011.

[Mal10] Malcolm, B. Software Interactions and the GNU General Public License. IFOSS L. Rev, 2(2), pp 165 - 180. 2010.

[Ninka] Ninka, a license identification tool for Source Code. http://ninka.turingmachine.org/. Last accessed Sep. 2011.

[SF] Sourceforge.net. http://sourceforge.net/. Last accessed Sep. 2011.

[SPDX] Software Package Data Exchange (SPDX). http://spdx.org/. Last accessed Sep. 2011.

[Tuu09] Tuunanen, T., Koskinen, J. and Kärkkäinen, T. Automated software license analysis. Automated Software Engineering 16 (3-4), 455-490, Dec. 2009.

[Wil10] von Willebrand, M. and Partanen, M. P. Package Review as a Part of Free and Open Source Software Compliance. IFOSS L. Rev, 2(2), pp 39 – 60. 2010.

[Gom08] Gomez, F. P. and Quiñones, K. S. Legal Issues Concerning Composite Software. In proc. of ICCBSS 2008, pp 204-214, 2008.