OSSLI: Architecture Level Management of Open Source Software Legality Concerns

Alexander Lokhman, Antti Luoto, Salum Abdul-Rahman,
and Imed Hammouda

Tampere University of Technology
{firstname.lastname}@tut.fi

Abstract. This paper presents a tool that addresses the legality concerns of open source at the level of software architecture, early in the development activity. The tool demonstrates the significance of licensing concerns at the architectural level by extending existing modeling tools with support for open source licensing issues.

1 Introduction

The overall research problem we investigate in this work is twofold. First, we study the significance of licensing concerns at the architectural design phase. Second, we investigate what kind of tool concepts are needed for addressing open source licensing issues in architectural models.

Our goal is to develop a new kind of open source legality support tool, named OSSLI (Open Source Software Licensing)[1], focusing especially on validating architectural models against open source legality and proposing remedial architectural solutions. The OSSLI tool provides mechanisms to express licensing constraints, detect violations while application model is being constructed, and provide possible remedies and alternative solutions to resolve possible license violations. The tool is implemented as a plug-in to existing CASE tools.

2 Background

The Open Source Initiative lists about 70 licenses[2]. 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. Typically licenses are categorized as permissive (e.g. MIT), weak copyleft (e.g. LGPL), and strong copyleft (e.g. GPL).

There are a number of ontologies and standards proposed for documenting the legal rules and constraints of software systems, with respect to open source

[1] http://ossli.cs.tut.fi/
[2] http://opensource.org/licenses

software. Examples include Legal Knowledge Interchange Format (LKIF [4]), Software Package Data Exchange (SPDX)[3], and QualiPSo Intellectual Property Rights Tracking (IPRT)[4]. These works could contribute to the foundation of our work, but nevertheless should be enhanced for better ties with the work processes and methods of software architects.

Licenses can be conflicting [3]. As an example, a software component under the terms of GPL cannot be directly linked with another under the terms of the Apache license. The 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.

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) [1]. Also the interpretation may depend on how the component is used: as a redistributable product, as a hosted service, as a development tool, or for internal use [8]. Furthermore, Open Source legality interpretations are subject to the way software is implemented, packaged, and deployed [3, 5].

The legality challenge of FLOSS has been partly addressed using so-called license analysis techniques and tools. These tools provide functionality to identify the licenses (e.g. Ninka[5]) used and to verify license compliance in source code packages (e.g. FOSSology[6], OSLC[7], and ASLA [7]). These techniques however are mostly useful in analyzing ready packaged software systems but give little guidance, with respect to licensing issues, for software developers during the development activity itself.

3 OSSLI Tool Architecture

Our goal is to develop a new kind of open source legality support tool, OSSLI, focusing especially on validating architectural models against open source legality constraints and proposing remedial architectural solutions. Figure 1 depicts the overall architecture of the 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 1, in turn, explains each of the architectural components. 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

[3] http://spdx.org/
[4] http://qualipso.org/licenses-champion
[5] http://ninka.turingmachine.org/
[6] http://fossology.org/
[7] http://sourceforge.net/projects/oslc

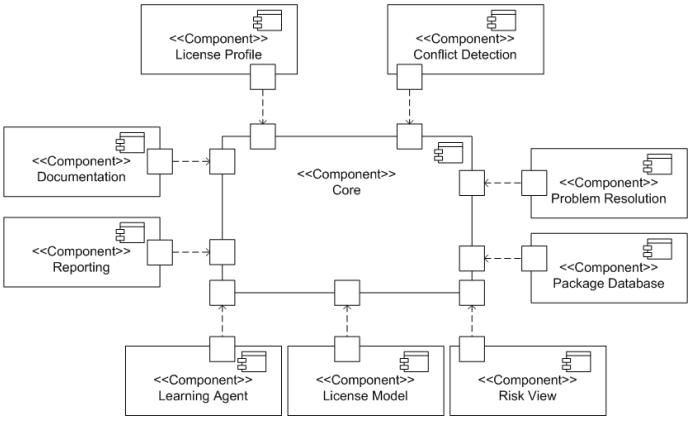

Fig. 1. An Open Architecture for Open Source Compliance

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).

The OSSLI tool is built on top of Eclipse and is integrated to Papyrus[8], which is an open source tool for graphical UML2 modeling. The plug-in architecture of the tool allows users and developers to extend its functionality beyond the support that comes as part of the base version. Any new feature that corresponds to any of the extension point shown in Figure 1 can be developed as an Eclipse plug-in and deployed to an own version of Eclipse. A video recording of a number of usage scenarios of the tool is available in the OSSLI project website.

4 Example Usage Scenarios

In the following we present a number of usage scenarios of the OSSLI tool.

4.1 Annotating design models with licensing information

Using the OSSLI tool, UML models can be extended with license and IPR related information by applying a UML profile. The profile introduces concepts related to the properties of open source licenses in the form of stereotypes

[8] `http://papyrusuml.org/`

Table 1. Architectural Components

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

and meta-attributes. This allows the user to create a UML model that takes into account what licenses each component is associated with. For example, a software package could be annotated with information such as copyright holder, license type, and the risks associated with its use in different usage scenarios. Figure 2 shows an example annotation of example components.

4.2 Visualizing the risk levels of open source components

Once the components in the UML diagram have been annotated with the information defined in the profile, the user can activate a risk view plug-in. The user selects the concrete risk evaluator s/he wants to use. The user can also choose to analyze part of the model. The user is prompted for the intended application domain, e.g. “internal use”, “redistribution” or “as a hosted service”. The tool analyzes the information in the UML diagram using the fields defined in the profile to evaluate the level of legal risk of using each component for the selected application domain. The legal risk level is displayed by changing the background color of the components analyzed. In Figure 2, for example, three packages show no risk (green color) and one package is of a clear risk.

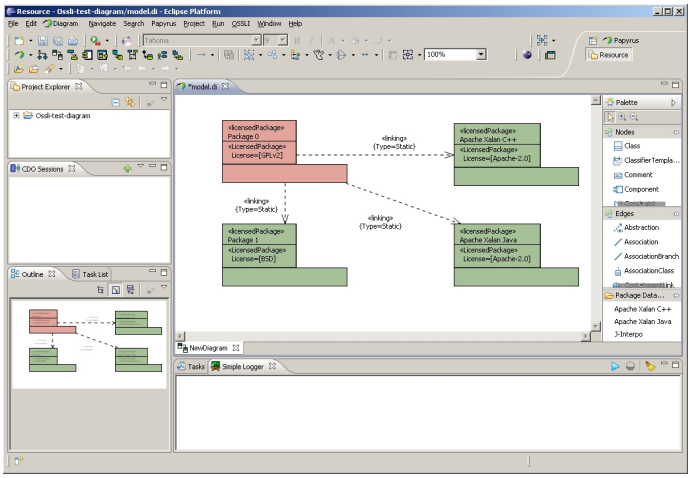

Fig. 2. Risk view analysis

4.3 Identifying license conflicts

Using the OSSLI tool, it is possible to identify conflicts related to how software components are integrated. This is based on the licensing information attached to the components themselves and the inter component links. The tool can either analyze the whole diagram or just a selected set of components. The detected conflicts are presented to the user and are highlighted in the UML diagram in red color.

The identified conflicts can be analyzed and possible remedial actions can be suggested. For example the way two conflicting components communicate could be changed from static linking to dynamic linking. Another example remedial action could be to add an intermediate layer with a neutral license between two conflicting components.

4.4 Logging decisions for future recommendation

The OSSLI tool is capable to record the detected conflicts and the corresponding remedial actions for future recommendation. An example logged session is given in Figure 3. In the background picture shows a conflict detected between two software packages. In order to address the conflict, the user has changed the type of dependency (named DEP 9) from static to dynamic.

In the foreground dialog, a new conflict has been detected between two other software packages licensed under the terms of Apache 2.0 and LGPL v2.1 licenses respectively. The OSSLI Tool suggests two solutions of which the first one corresponds to the recorded remedial action taken in the background picture, i.e. hanging the link type from static to dynamic. The second recommendation is to relicense Barcode 4J under the terms of LGPL v2.1. This is of course subject to ownership of the copyrights.

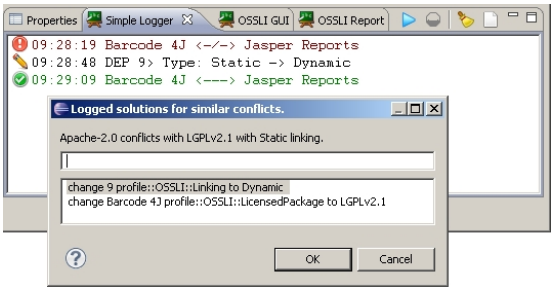

Fig. 3. Logging solutions and recommendation

5 Discussion

In this paper we have presented a tool that is capable of expressing, analyzing, and managing open source licensing constraints in architectural design models. As future work, we are planning to enhance the OSSLI tool with new features. For example we plan to investigate how the proposed concepts could be the basis for building novel techniques to devise optimal architectural solutions taking into consideration the legality constraints. This could be achieved, for instance, through the use of genetic algorithms [6]. We are also enhancing the tool recommendation capabilities with more advanced techniques, for example by applying search-based heuristics such as Tabu search [2]. Also we plan to apply the tool to a number of industrial case studies.

References

1. D. German and A. Hassan. License integration patterns: Addressing license mismatches in component-based development. *In proc. ICSE09*, pages 188–198, 2009.
2. F. Glover and M. Laguna. *Tabu Search*. Reading, Kluwer Academic Publishers, Boston/Dorderecht/London, 1997.
3. I. Hammouda, T. Mikkonen, V. Oksanen, and A. Jaaksi. Open source legality patterns: Architectural design decisions motivated by legal concerns. *In proc. AMT10*, pages 207–214, 2010.
4. R. Hoekstra, J. Breuker, M. Di Bello, and A. Boer. The LKIF core ontology of basic legal concepts. *In proc. LOAIT07*, pages 43–63, 2007.
5. B. Malcolm. Software interactions and the GNU General Public License. *IFOSS L. Rev*, 2(2):165–180, 2010.
6. O. Raiha, Hadaytullah, K. Koskimies, and E. Makinen. Synthesizing architecture from requirements: A genetic approach. *In: Relating Software Requirements and Architecture (eds. P. Avgeriou, et al.)*, Chapter 18:307–331, 2011.
7. T. Tuunanen, J. Koskinen, and T. Karkkainen. Automated software license analysis. *Automated Software Engineering*, 16 (3-4):455–490, 2009.
8. M. von Willebrand and M. Partanen. Package review as a part of free and open source software compliance. *IFOSS L. Rev*, 2(2):39–60, 2010.