*2nd International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS),* pp. 1–4

# Analyzing Software Licenses in Open Architecture Software Systems

Thomas A. Alspaugh
*Department of Computer Science*
*Georgetown University*
*Washington, DC 20057 USA*
*alspaugh@cs.georgetown.edu*

Hazeline U. Asuncion and Walt Scacchi
*Institute for Software Research*
*University of California, Irvine*
*Irvine, CA 92697-3455 USA*
*{hasuncion,wscacchi}@ics.uci.edu*

## Abstract

*A substantial number of enterprises and independent software vendors are adopting a strategy in which software-intensive systems are developed with an open architecture (OA) that may contain open source software (OSS) components or components with open APIs. The emerging challenge is to realize the benefits of openness when components are subject to different copyright or property licenses. In this position paper, we identify key properties of OSS licenses, present a license analysis scheme, and discuss our approach for automatically analyzing license interactions.*

## 1. Introduction

Open architectures have generally referred to the ability to use third party components to create a software system. Oreizy uses the term to refer to his customization technique of making the architecture model an explicit and malleable part of the deployed system [16], while the Department of Defense community uses the term to refer to guidelines on acquiring and composing third party components into a software system [18]. Today, we see more and more software-intensive systems developed using an OA strategy not only with open source software (OSS) components but also proprietary components with open APIs (e.g. [20]). Developing systems using the OA technique can lower development costs [18]. Composing a system with heterogeneously-licensed components, however, increases the likelihood of liabilities stemming from incompatible licenses. Thus, in this paper, we define an OA system as *a software system consisting of components that are either open source or proprietary with open API, whose overall system rights at a minimum allow its use and redistribution.*

OA systems were formerly composed solely of homogenously-licensed OSS components. These OSS projects have commonly required developers to contribute their work under conditions that ensure the project can license its products under a specific OSS license. This is changing, however. More systems are being composed of software components associated with different licenses. The resulting system may not have any recognized OSS license at all—but if the system is designed well and if the corresponding obligations are met, copyright rights may be available to allow its redistribution and sublicensing.

Due to the sheer number of license types, variants, versions, and the various stipulations attached to each of these licenses, analyzing the compatibility or lack thereof between the various licenses in a system is extremely difficult. Licenses are often incomplete or hard to understand. Licenses are also legally binding.

Thus, we aim to identify principles of software architecture and software licenses that facilitate success of an OA system. We present a systematic approach to analyzing license interaction within a system using a formal license model that can adequately express the majority of current license types. We then incorporate this model into xADL, an extensible architecture description language that rigorously represents a software system [10]. We discuss our automated support for analyzing licenses within ArchStudio4 [11].

## 2. Background

There is little explicit guidance on how best to develop, deploy, and sustain complex software systems when different OA and OSS objectives are at hand. Ven [21] and German [8] are recent exceptions.

OA may simply seem to mean software system architectures incorporating OSS components and open application program interfaces (APIs). But not all software system architectures incorporating OSS components and open APIs will produce an OA, since the available license rights of an OA depend on: (a) how/why OSS and open APIs are located within the system architecture, (b) how OSS and open APIs are

implemented, embedded, or interconnected, (c) whether the licenses of different OSS components encumber all/part of a software system's architecture into which they are integrated, and (d) the fact that many alternative architectural configurations and APIs exist that may or may not produce an OA system (cf. [3, 18]). Thus, new software development or acquisition requirements may stipulate a software system with an OA and OSS, but the resulting software system may or may not have the rights needed to embody an OA system.

## 3. Understanding open architectures

Stating that an OA system comprises OSS and open API components does not clearly indicate what possible mixes of software elements may be configured into such a system. To help explain this, we first identify software elements included in common software architectures that affect whether they are open or closed [5].

***Software source code components*** – These can be either (a) standalone programs, (b) libraries, frameworks, or middleware, (c) inter-application script code (e.g., C shell scripts) and (d) intra-application script code (e.g., to create Rich Internet Applications using domain-specific languages such as XUL for Firefox Web browser [6] or "mashups" [15]).

***Executable components*** -- These are programs for which the software is in binary form, and its source code may not be open for access, review, modification, and possible redistribution. Since executable binaries are considered as a compilation of source code, they can be viewed as "derived works" [17].

***Application program interfaces/APIs*** – The availability of externally visible and accessible APIs is the minimum requirement to form an "open system" [14].

***Software connectors*** – Software whose intended purpose is to provide a standard or reusable way of communication through common interfaces, e.g. High Level Architecture (HLA) [12], CORBA, MS .NET, Enterprise Java Beans, and GNU Lesser General Public License (LGPL) libraries.

***Configured system or sub-system architectures*** – These are software systems whose internal architecture may comprise of components with different licenses, affecting the overall system license. To minimize license interaction, a configured system or sub-architecture may be surrounded by a *license firewall,* a layer of dynamic links, client-server connections, license shims, or other connectors that block the propagation of reciprocal obligations. A license that prohibits the use of license firewalls is the Affero General Public License (AGPL) [2].

## 4. Understanding open software licenses

A particularly knotty challenge is the problem of heterogeneous licenses in software systems. There has been an explosion in the number, type, and variants of software licenses, especially with open source software (cf. [1]). License types include General Public License (GPL), Mozilla Public License (MPL), Apache Public License, (APL), academic licenses such as Berkeley Software Distribution (BSD) and MIT, Creative Commons, Artistic, and Public Domain (either via explicit declaration or by expiration of prior copyright license). Within each license types are numerous variants. Furthermore, licenses can evolve, resulting in new license versions over time. Finally, each license stipulates different constraints to software components that bear it. Discussions of many different licenses currently used with OSS are available [1, 7, 17, 19].

The way components are configured also affects the license of the overall system. Furthermore, the component configurations at build time and run time may have different license implications. For instance, components may be statically bound or interconnected at build-time, while other components may only be dynamically linked for execution at run-time, and thus might not be included as part of a software release or distribution. On top of this, software maintenance such as architectural refactoring, alternative component interconnections, and component replacement (via maintenance patches, installation of new versions, or migration to new technologies) can all have effects on the overall license of the system.

### 4.1. Software licenses: rights and obligations

Copyright, the common basis for software licenses, gives the original author of a work certain exclusive rights, e.g. right to use, copy, modify, merge, publish, distribute, sub-license, and sell copies. These rights may be licensed to others, individually or in groups, and either exclusively or non-exclusively. After a period of years, the rights enter the public domain. Until then copyright may only be obtained through licensing.

Licenses may impose obligations that must be met in order for the licensee to realize the assigned rights. Commonly cited obligations include the obligation to publish at no cost the source code you modify (MPL) or the reciprocal obligation to publish all source code included at build-time or statically linked (GPL). The obligations may conflict, as when a GPL'd component's reciprocal obligation to publish source code of other components is combined with a proprietary license's prohibition of publishing source code. In this case, rights may not be available for the system as a

whole, not even the right of use, because the two obligations cannot simultaneously be met.

The basic relationship between software license rights and obligations can be summarized as follows: if the specified obligations are met, then the specified rights are granted. For example, if you publish modified source code and sub-licensed derived works under MPL, then you get all the MPL rights for the original and modified code. However, license details are difficult to comprehend and track—it is easy to get confused or make mistakes. Licenses written by developers are often incomplete and legally ambiguous, while those written by lawyers, usually more recently, are more exact and complete but can be difficult for non-lawyers to grasp. The challenge is multiplied when dealing with configured system architectures that compose multiple components with heterogeneous licenses, so that the need for legal interpretations begins to seem inevitable (cf. [7, 17]).

### 4.2. Expressing software licenses

We propose a scheme for expressing software licenses that is more formal and less ambiguous than natural language, and that allows us to identify conflicts arising from the various rights and obligations pertaining to two or more component's licenses. We considered relatively complex structures (such as Hohfeld's eight fundamental jural relations [9]) but, applying Occam's razor, selected a simpler structure. We start with a tuple *<actor, operation, action, object>* for expressing a right or obligation. The *actor* is the "licensee" for all the licenses we have examined. The *operation* is one of the following: "may", "must", or "must not", with "may" expressing a right and "must" and "must not" expressing obligations. A copyright right is only available to entities who have been granted a sublicense. Thus, only the listed rights are available, and the absence of a right means that it is not available. The *action* is a verb or verb phrase describing what may, must, or must not be done, with the *object* completing the description. We specify an object separately from the action to minimize the set of actions. A license may be expressed as a set of rights, with each right associated with zero or more obligations that must be fulfilled in order to enjoy that right. Figure 1 displays the tuples and associations for two of the rights and their associated obligations for the academic BSD software license. Note that the first right is granted without corresponding obligations.

When designing an OA software system, there are heuristics that can be employed to enable architectural design choices that might otherwise be excluded due to license conflicts. First, it is possible to employ a license firewall that serves to limit the scope of reciprocal obligations. Rather than simply interconnecting conflicting components through static linking of components at build time, such components can be logically connected via dynamic links, client-server protocols, license shims (e.g., via LGPL connectors), or run-time plug-ins. Second, the source code of statically linked OSS components must be made public. Third, it is necessary to include appropriate notices and publish required sources when academic licenses are employed. However, even using design heuristics such as these (and there are many), keeping track of license rights and obligations across components that are interconnected in complex OAs quickly become too cumbersome. Thus, automated support is needed to manage the multi-component, multi-license complexity.

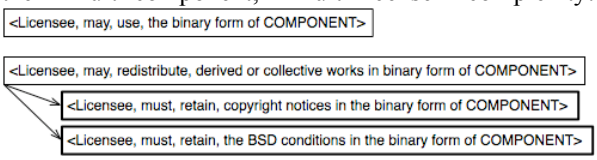


**Figure 1. A portion of the BSD license tuples**

## 5. Automating software license analysis

If we start from a formal specification of a software system's architecture, we can associate software license attributes with the system's components, connectors, and sub-system architectures and calculate the copyright rights and obligations for the system's configuration. Accordingly, we use an architectural description language specified in xADL [10] to describe OAs that can be designed and analyzed with a software architecture design environment [13], such as ArchStudio4 [11]. ArchStudio4 currently has software traceability tool support (cf. [4]) and we have extended it with a Software Architecture License Traceability Analysis module. This allows for the specification of licenses as a list of attributes (license tuples) using a form-based user interface in ArchStudio4 [11, 13].

We analyze rights and obligations as follows:

***Propagation of reciprocal obligations.*** We follow the widely-accepted interpretation that build-time static linkage propagate the reciprocal obligations, but the "license firewalls" do not. Analysis begins, therefore, by propagating these obligations along all connectors that are not license firewalls.

***Obligation conflicts.*** An obligation can conflict with another obligation, or with the set of available rights, by requiring a copyright right that has not been granted. For instance, a proprietary license may require that a licensee must not redistribute source code, but GPL states that a licensee must redistribute source code. Thus, the conflict appears in the modality of the

two otherwise identical obligations, "must not" in a proprietary software and "must" in GPL.

***Rights and obligations calculations.*** The rights available for the entire system (use, copy, modify, etc.) are calculated as the intersection of the sets of rights available for each component of the system. If a conflict is found involving the obligations and rights of linked components, it is possible for the system architect to consider an alternative linking scheme, e.g. using one or more connectors along the paths between the components that act as a license firewall. This means that the architecture and the environment together can determine what OA design best meets the problem at hand with available software components. Components with conflicting licenses do not need to be arbitrarily excluded, but instead may expand the range of possible architectural alternatives if the architect seeks such flexibility and choice.

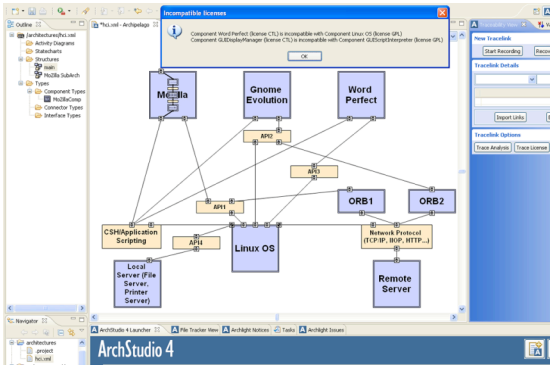


**Figure 2: License traceability analysis tool**

## 6. Ongoing work

We are currently encoding major license types such as GPL, MPL, CTL to examine the effectiveness of the license tuple encoding and the calculations based upon it. Thus far, we are finding that the tuple representation is sufficiently expressive for our needs. We are also currently evaluating the effectiveness of our automated license analysis on an actual heterogeneously licensed system. In addition, we are exploring the impact of patent and other provisions in licenses. Finally, we are studying how the design time and build time analysis of component configuration relates to the eventual run-time license of a system.

## 8. Acknowledgments

This research is supported by grants #0534771 and #0808783 from the U.S. National Science Foundation and Acquisition Research Program at the Naval Postgraduate School. No endorsement implied.

## 9. References

[1] *The Open Source Initiative.* http://www.opensource.org/, 2008.

[2] Affero Inc. *Affero General Public License.* http://www.affero.org/oagpl.html, 2007.

[3] Alspaugh, T.A. and Antón, A.I. Scenario Support for Effective Requirements. *Information and Software Technology.* 50(3), p. 198-220, February, 2007.

[4] Asuncion, H. Towards Practical Software Traceability. In *Proc. of the 30th International Conf on Software Engineering Doctoral Symposium.* Leipzig, Germany, 2008.

[5] Bass, L., Clements, P., et al. *Software Architecture in Practice.* 2nd ed. Addison-Wesley Professional: New York, 2003.

[6] Feldt, K. *Programming Firefox: Building Rich Internet Applications with XUL.* O'Reilly Press: Sebastopol, CA, 2007.

[7] Fontana, R., Kuhn, B.M., et al. *A Legal Issues Primer for Open Source and Free Software Projects.* http://www.softwarefreedom.org/resources/2008/foss-primer.pdf, Software Freedom Law Center, Report Version 1.5.1, 2008.

[8] German, D.M. and Hassan, A.E. License Integration Patterns: Dealing with Licenses Mismatches in Component-Based Development. In *Proc. of the 31st International Conference on Software Engineering (ICSE 2009).* Vancouver, Canada, May 16-24, 2009.

[9] Hohfeld, W.N. Some Fundamental Legal Conceptions as Applied in Judicial Reasoning. *Yale Law Journal.* 23(1), p. 16-59, 1913.

[10] Institute for Software Research. *xADL 2.0.* University of California, Irvine. http://www.isr.uci.edu/projects/xarchuci/

[11] Institute for Software Research. *ArchStudio 4.* University of California, Irvine, 2006. http://www.isr.uci.edu/projects/archstudio/

[12] Kuhl, F., Weatherly, R., et al. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture.* Prentice-Hall PTR: Upper Saddle River, New Jersey, 2000.

[13] Medvidovic, N., Rosenblum, D.S., et al. A Language and Environment for Architecture-Based Software Development and Evolution. In *Proc. of the 21st International Conference on Software Engineering (ICSE '99).* p. 44-53, Los Angeles, CA, May 16-22, 1999.

[14] Meyers, B.C. and Obendorf, P. *Managing Software Acquisition: Open Systems and COTS Products.* Addison-Wesley: New York, 2001.

[15] Nelson, L. and Churchill, E.F. Repurposing: Techniques for Reuse and Integration of Interactive Services. In *Proc. of the IEEE Intern. Conf. Information Reuse and Integration.* Sep, 2006.

[16] Oreizy, P. *Open Architecture Software: A Flexible Approach to Decentralized Software Evolution.* Thesis (Ph. D., Information and Computer Science), University of California, 2000. http://www.ics.uci.edu/~peymano/papers/thesis.pdf.

[17] Rosen, L. *Open Source Licensing: Software Freedom and Intellectual Property Law.* Prentice-Hall PTR: Upper Saddle River, New Jersey, 2005.

[18] Scacchi, W. and Alspaugh, T.A. Emerging Issues in the Acquisition of Open Source Software by the U.S. Department of Defense. In *Proc. of the 5th Annual Acquisition Research Symposium.* May 13-15, 2008.

[19] St. Laurent, A.M. *Understanding Open Source and Free Software Licensing.* O'Reilly Press: Sebastopol, CA, 2004.

[20] Unity Technologies. *End User License Agreement.* http://unity3d.com/unity/unity-end-user-license-2.x.html, 2008.

[21] Ven, K. and Mannaert, H. Challenges and Strategies in the Use of Open Source Software by Independent Software Vendors. *Information and Software Technology.* 50, p. 991-1002, 2008.