Journal of the Association for Information Systems
JAIS

Special Issue

Software Licenses in Context: The Challenge of Heterogeneously-Licensed Systems

Thomas A. Alspaugh
Georgetown University
thomas.alspaugh@acm.org

Walt Scacchi
University of California, Irvine
wscacchi@ics.uci.edu

Hazeline U. Asuncion
University of Washington, Bothell
hazeline@u.washington.edu

Abstract

The prevailing approach to free/open source software and licenses has been that each system is developed, distributed, and used under the terms of a single license. But it is increasingly common for information systems and other software to be composed with components from a variety of sources, and with a diversity of licenses. This may result in possible license conflicts and organizational liability for failure to fulfill license obligations. Research and practice to date have not kept up with this sea-change in software licensing arising from free/open source software development. System consumers and users consequently rely on ad hoc heuristics (or costly legal advice) to determine which license rights and obligations are in effect, often with less than optimal results; consulting services are offered to identify unknowing unauthorized use of licensed software in information systems; and researchers have shown how the choice of a (single) specific license for a product affects project success and system adoption. Legal scholars have examined how pairs of software licenses conflict but only in simple contexts. We present an approach for understanding and modeling software licenses, as well as for analyzing conflicts among groups of licenses in realistic system contexts, and for guiding the acquisition, integration, or development of systems with free/open source components in such an environment. This work is based on an empirical analysis of representative software licenses and of heterogeneously-licensed systems. Our approach provides guidance for achieving a "best-of-breed" component strategy while obtaining desired license rights in exchange for acceptable obligations.

Keywords: *Open source software, software licenses, case study, semantic modeling, system architecture, design theory*

* Michael Wade and Kevin Crowston were the accepting senior editors. This article was submitted on October 23, 2009 and went through two revisions.

Volume 11, Special Issue, pp. 730-755, *November* 2010

Software Licenses in Context: The Challenge of Heterogeneously-Licensed Systems

1. Introduction

The hallmark of Free/Open Source Software (FOSS) is that the *source code* is available for remote access, open to study and modification, and available for redistribution to others with few constraints, except the rights and obligations that insure these freedoms. FOSS sometimes adds or removes similar freedoms or copyright privileges depending on which FOSS copyright and end-user license agreement is associated with a particular FOSS code base (Fontana et al., 2008; Rosen, 2005). Some licenses for "free software" such as the group of GNU General Public License (GPL) and Lesser General Public License (LGPL) versions (Free Software Foundation, 1991, 1999, 2007a, 2007b, 2007c) are well known and widely used, while others are obscure and sometimes specific to a particular software vendor. The Open Source Initiative (OSI, 2009), whose web portal gives information on many facets of FOSS, especially FOSS licenses, currently certifies more than 50 FOSS licenses, thus indicating a diverse ecology of freedoms, copying rights, and other license obligations and constraints.

Some FOSS licenses overlap or subsume one another's rights, while others present potential conflicts when comparing one license to another. Consequently, FOSS developers generally choose a single license to apply to their FOSS project, as part of their governance regime (de Laat, 2007). The choice of FOSS license to apply has been a defining characteristic of most FOSS projects, where the license chosen may connote not only an intellectual property sharing regime, but also a statement about beliefs, values, and norms expected to be shared by FOSS project developers, as well as affiliation within a larger social movement (Elliott and Scacchi, 2003, 2008, Roberts et al., 2006). However, a single license may not be sufficient to provide "copyleft" access to non-software specific data objects, representations, processing rules, or visual renderings. Similarly, with ever more FOSS components becoming available with different FOSS licenses — and some now even offered under multiple licenses[1] — and given that different versions of a particular software component may have different licenses or license constraints, software and information system developers face a growing challenge: to determine how multiple software licenses interact, whether during system design (at "design time"), while compiling and linking source code to produce an executable program/binary (at "build time"), or when installing and running a newly acquired/downloaded version of software from a FOSS project or other provider that may need to interoperate with other software programs (at "run time").

The problem we address in this article is that of understanding and analyzing what happens when software systems are developed from FOSS or proprietary components that are not all under the same license. What license applies to the resulting system? What rights or obligations apply? How can one determine which license constraints match, subsume, or conflict with one another? We refer to this as *the challenge of heterogeneously-licensed systems* (HLSs), and we find that a growing number of firms and government agencies must increasingly address this challenge. Consider the following two examples: one short, and the second more detailed.

First, when Bank of America took over operation of the Merrill Lynch (ML) trading firm in 2008, ML was known as a leading developer of in-house financial and trading systems incorporating FOSS components. However, the bank now had to rapidly determine whether this corporate takeover constituted a "software distribution" by ML, as well as what consequences might arise from integrating the ML systems with its own (Asay, 2008), since this would likely create an HLS.

Second, Unity3D is an interactive environment for modeling and animating 3D graphic objects and object compositions within computer games or virtual worlds. Unity3D is an HLS, as seen in its software copyright license agreement that lists 18 externally produced components or groups of components, apparently distributed under eleven distinct licenses (Unity Technologies, 2009). So

[1] MySQL (2006) is offered under a dual-license choice of GPL (any version) or a proprietary license, and Mozilla now offers a tri-license choice of the Mozilla Public License (MPL) version 1.1 or later, GNU General Public License version 2.0 or later, or GNU Lesser General Public License version 2.1 or later, for its core software offerings (Mozilla, 2009).

what "license" rights and obligations apply to this software? Is it the concatenation, union, or intersection of the license constraints found in each of the identified license copyrights? Do any of these license constraints conflict with one another (e.g. stipulating no redistribution of software versus ensuring the right to software redistribution)? How does the architectural configuration of the software components associated with a given license affect which component licenses interact, and which do not? Understanding the rights and obligations incorporated into the Unity3D system license requires understanding and analyzing the corresponding terms and conditions of each of these licenses, and potentially understanding the architecture in which Unity3D incorporates them. This is the burden facing software consumers, and it appears to be one that is growing. It is also a burden facing software integrators, as they must ensure they can give appropriate rights (and impose acceptable obligations) for their consumers.

The current state of the art of research in the fields of law and FOSS does not address these concerns, as we will show in more detail in the Related Research section. Legal researchers have examined interactions between pairs of FOSS licenses in the abstract, but not in the context of real systems and the architectural components and configurations found there. FOSS research in this area has concentrated on recovering or confirming the license of an individual homogeneously-licensed software component, with only one group (other than ourselves) examining the application of FOSS licenses in the context of actual HLSs, and then only of pairs of licenses applied in a small fixed number of architectural contexts. None of these approaches provide answers to the questions posed above.

We believe these answers can be determined in part through systematic empirical means that rely on analysis of (a) the interpretation of software license terms and conditions within different FOSS licenses, along with (b) the configuration of software components (and licenses) that denote the architectural composition of the resulting system.[2] In particular, we argue that (b) requires an open source model of the architectural configuration of a system in which each component has an open (published) interface. We call such a configurational model an *open architecture* (OA) (Oreizy, 2000). In current practice, an OA is not available to external developers or users when proprietary software components/systems are included, as is the case for Unity3D. We find that the size and complexity of the HLSs we have examined, combined with increasingly large numbers of licenses involved, outstrip the ability of developers to manually determine the rights and obligations for the system and to identify potential and actual conflicts. Our research has thus been drawn to models and approaches that support automatic analysis, calculations, and guidance for licensing challenges, and that can be integrated into the processes and tools that HLS developers already use.

Our efforts are directed at theory building, since there is no existing theory for understanding HLSs and analyzing them to determine overall license rights, obligations, and conflicts. As such, we are developing a theory-based, empirically verified model as an operational theory for how to understand and analyze FOSS licenses in ways that determine where HLS architectures have conflicting rights and obligations, as well as how these architectures may be modified to remove the conflicts. Because existing theory, as outlined under Related Research, is not sufficient to explain or analyze the license interactions through system architecture that arise in HLSs, we extend it through a *design science* approach (Hevner et al., 2004), extending the existing "kernel" theories through a grounded-theory qualitative analysis of representative licenses to identify missing constructs (Glaser and Strauss, 1967), and validating the extended theory by embodying it in automated development tools and applying it in current-day contexts. The tools provide a proof of concept and starting point for a production-quality capability for analyzing, guiding, and verifying license governance and compliance of HLSs, and are discussed elsewhere (Alspaugh et al., 2009a). This capability supports the needs of large enterprises that increasingly develop complex information systems using or reusing existing FOSS systems/components rather than developing each system from a blank slate.

[2] Our efforts are not intended to be construed as offering legal advice, but our approach is adaptable to different legal framings or interpretations, which are beyond the scope of this paper. Thus, our approach does not claim to offer complete or definitive answers according to an existing legal interpretation.

The goal of this work is to develop an extensible, automatable theory for analyzing the obligations and rights of software licenses in the kinds of complex systems now seen in enterprises. The theory is based on the legal foundations of licenses and contracts, and accounts for the range of current and future license provisions. It makes it a practical possibility to produce systems of best-of-breed components, whether FOSS or proprietary and under whatever licenses, while addressing not only the classical requirements of functionality, reliability, testability, etc. but also the licensing-oriented requirements that arise for HLSs. These include production of a system that can be legally used; legally distributed; legally modified and evolved to meet changing requirements; and for which it can be reliably shown what rights are obtainable and what obligations must be met in order to obtain specific desired rights.

Our approach starts by constructing a semantic model of the rights and obligations found in the text of FOSS licenses. This modeling need only be done once for a license, unless or until the license or its interpretation is changed. Next, the analytical methods we propose determine whether conflicts arise when specific FOSS-licensed components are incorporated within an OA, working from component attributes that give the semantic models for their corresponding license. Last, our testbed environment demonstrates that our theory can be supported with automated tools, which offer the potential to analyze large, complex HLSs at design time, build time, or run time. As a result, our theory is operational in that it can be applied to a complex HLS to empirically determine whether its architecture possesses conflicting rights or obligations, given its components' licenses and interconnections. Finally, our theory is not limited to a single pre-determined interpretation of the meaning of terms for rights and obligations found in FOSS licenses. It allows for alternative legal interpretations of a license, which can then be analyzed and supported by our approach and with our automated tools, when so coded to reflect these interpretations.

Our theory provides answers to the questions posed above. By examining HLSs, we find that it is typical that no license, per se, applies to the resulting system. Instead, there is a collection of rights available for the system, and for each right there are corresponding obligations that must be met. We find that the collection of rights may be empty, and that if specific rights are desired, such as the right to use the system and the right to distribute it, these rights must be considered at each stage of development from design through distribution. The specific rights and obligations are determined by the licenses of the system's components and by the architectural configuration in which they are combined. Informal analysis of license constraints by experts can identify which ones may match, subsume, or conflict, but our theory supports a formal analysis that can be automated for non-experts. These issues are addressed in our other work, which implements and applies an initial version of the theory presented and elaborated here (Alspaugh et al., 2009b).

The remainder of the paper is organized as follows. In the next section, "A Motivating Example," we present a motivating example displaying some of the non-obvious ways in which FOSS license conflicts can manifest themselves. The "Related Research" section reviews prior research that helps inform our understanding of the problem we are addressing. Emphasis is placed on studies that rely on empirically grounded investigations of FOSS development projects and outcomes. The "Intellectual Property (IP) Basics" section summarizes IP law, and "Software Licensing and FOSS Licenses" does the same for licenses.

The section "A Theory of Software Licenses and their Application" summarizes existing legal theory and presents the basic meta-model and reasoning rules that can be derived from it. Following this, we present an empirical study of a selection of widely used software licenses, and a study approach that can be extended to additional licenses, and use the results to extend the meta-model and render it strong enough to support automated application of licenses and calculation involving them. In this regard, we seek to provide a semantic or logical model (a set of logical constraints) for specifying license rights and obligations, such that any use of a given license reuses its logical model.

The next section "License Architecture of Heterogeneously-Licensed Systems" presents the types of architectural components and connectors that our study showed as significant for FOSS licensing. Following this, "Embodying and Applying the Models and Analysis" presents an external validation of

our theoretical results, in which we embodied the theory in an interactive software architecture environment and applied it to HLSs. Such an environment demonstrates the potential to scale up the analysis to large, complex systems that may be too difficult to analyze without such automated support. The value of the environment is also demonstrated through detection of conflicting license constraints across components that can be resolved through changes to the system's architectural configuration, such as through the introduction of license firewalls or substitution of similar software components with different licenses. This section is then followed by the "Discussion" section that offers our view of any HLS as being subject to a "virtual license" that results from an empirical analysis of the different licenses superimposed on the architecture of a composed software system. Such a virtual license represents a union of the license constraints across the components, once conflicting license constraints have been resolved, and an intersection of the license rights that result. We then conclude our study with a final statement of the results we have accomplished through our approach.

2. A motivating example

Shortly before beginning the research described here, we prototyped a new multimedia content management portal that included support for videoconferencing and video recording and publishing. Our prototype was based on an existing Adobe Flash Media Server (FMS), for which we developed both broadcast and multi-cast clients for video and audio that shared their data streams through the FMS. FMS is a closed source media server for which the number of concurrent client connections is limited, with the limit determined by the license fee paid. We could invite remote users to try out our clients and media services (up to the connection limit), but since the FMS license did not allow redistribution, we found we could not offer interested users a copy of the run-time environment that included the FMS. We could distribute everything but the FMS, though our compiled components were built to use our copy of the FMS. Consequently, recipients would be unable to run the system even if they purchased their own FMS license. The only useful way to distribute our portal system was in the form of the source code of our locally developed clients and services. A potential user would need to license, download, and install his own copy of the FMS, configure our source code to use it, and rebuild our system. In our view, this created a barrier to sharing the emerging results of our prototyping effort.

We subsequently undertook to replace the FMS with Red5, an open source Flash media server, so we could distribute a complete run-time version of our content management portal to remote developers. Now these developers could install and use our run-time system as is, or if they preferred they could download the source code, revise, build, and configure it to suit their own needs, and share their own run-time version.

Our experience illustrates how heterogeneous licensing can interact with system architecture decisions to hamper common software research and development activities in surprising ways, even for experienced FOSS designers and developers, and if not planned for successfully can cause substantial unexpected costs and delays. Our license theory, embodied in the tool described later in the paper, would have highlighted the lack of distribution rights and modeled the non-substitutability of the FMS server at system design time rather than much later at system distribution and run time.

3. Related research

It has been typical until recently that each software or information system is designed, built, and distributed under the terms of a single proprietary or FOSS license, with all its components homogeneously covered by that same license. The system is distributed, with sources or executables bearing copyright and license notices, and the license gives specific rights while imposing corresponding obligations that system consumers (whether external developers or users) must honor, subject to the provisions of contract and commercial law. Consequently, there has been some very interesting research on the choice of FOSS license for use in a FOSS development project, and its

consequences in determining the likely success of such a project.

Brown and Booch (2002) discuss issues that arise in the reuse of FOSS components, such as that interdependence (via component interconnection at design time, or linkage at build time or run time) causes functional changes to propagate, and versions of the components evolve asynchronously, giving rise to co-evolution of interrelated code in the FOSS-based systems. If the components evolve, the OA system itself is evolving. The evolution can also include changes to the licenses, and the licenses can change from component version to version.

Stewart et al. (2006) conducted an empirical study to examine whether license choice is related to FOSS project success, finding a positive association with the selection of business-friendly licenses. Sen, Subramaniam, and Nelson in a series of studies (Sen, 2007; Sen et al., 2008; Subramaniam et al., 2009) similarly find positive relationships between the choice of a FOSS license and the likelihood of both successful FOSS development and adoption of corresponding FOSS systems within enterprises. These studies direct attention to FOSS projects that adopt and identify their development efforts through use of a single FOSS license. However, there has been little explicit guidance on how best to develop, deploy, and sustain complex software systems when heterogeneously-licensed components are involved, and thus multiple FOSS and proprietary licenses may be involved.

Legal scholars have examined FOSS licenses and how they interact in the legal domain, but not in the context of HLSs. St. Laurent (2004) examines twelve FOSS licenses, including several no longer in wide use, and compares them to a hypothetical proprietary license he created; license interactions and conflicts are only very briefly discussed, and only in general terms. Rosen (2005) surveys eight FOSS licenses and creates two new ones written to professional legal standards. He examines interactions primarily in terms of the general categories of reciprocal and non-reciprocal licenses, rather than in terms of specific licenses. Rosen was general counsel for the Open Source Initiative (OSI, 2009). Fontana et al. (2008) primarily focus on guidance for FOSS projects on a number of legal issues, but provide a good and authoritative survey of FOSS licenses, especially the GPL group of licenses. Fontana et al. are lawyers (with one exception) associated with the Software Freedom Law Center; Fontana and Moglen were two of the authors of the GPL, LGPL, and AGPL version 3 licenses. Finally, Kemp (2009) reviews significant court cases that have been pursued, and how they do or do not address issues concerning the propagation of rights and obligations across programs depending on how they are derived, contained, compiled, or linked at build time, especially when the GPLv2 license is involved. Common to this legal scholarship is an approach that analyzes licenses and the interactions among them abstractly rather than in the context of an HLS, and on at most a pairwise basis. The characteristics of the software in which the licenses interact are not taken into account, or at most in very general terms, even though almost every FOSS license is framed in terms of the software and architectural constructs in existence when the license was written.

Ven and Mannaert (2008) discuss the challenges faced by independent software vendors developing an HLS. They focus on the evolution and maintenance of modified FOSS components. Tuunanen et al. (2009) exemplify most work to date on HLSs, in that they focus on reverse engineering and recovery of individual component licenses on existing systems, rather than on guiding HLS design to achieve and verify desired license outcomes. Their approach does not support the calculation of HLS virtual licenses. Many more researchers have worked from this after-the-fact point of view (Gobeille, 2008; Di Penta et al., 2010).

German and Hassan (2009) describe a license as a set of grants, each of which has a set of conjoined conditions necessary for the grant to be given. Figure 1 is a meta-model equivalent to their notational license definition. They analyze interactions between pairs of licenses in the context of five types of component connection. They also identify twelve patterns for avoiding license mismatches, found in an empirical study of a large group of FOSS projects, and characterize the patterns using their model. Their license models, developed independently from our work at about the same time, are equivalent to the first level of our license models and provide confirmation that our work builds on accepted foundations. As we show below, our license model goes beyond German and Hassan's to address semantic connections between obligations and rights that existing FOSS licenses exhibit,

and to connect with the structure of software systems in a general and extensible way rather than a fixed set of cases.

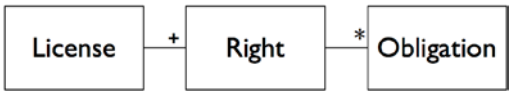


Figure 1. The meta-model used by German and Hassan (2009)

Other previous work examined how best to align acquisition, system requirements, architectures, and FOSS components across different software license regimes to achieve the goal of combining FOSS with proprietary software having open APIs when developing a composite "system of systems" (Scacchi and Alspaugh, 2008). This is particularly an issue for the U.S. Federal Government in its acquisition of complex software systems subject to Federal Acquisition Regulations (FARs) and military service-specific regulations. HLSs give rise to new functional and non-functional requirements that further constrain what kinds of systems can be built and deployed, as well as recognizing that acquisition policies can effectively exclude certain OA configurations, while accommodating others, based on how different licensed components may be interconnected.

4. Intellectual property (IP) basics

Software licenses are primarily concerned with copyright. Copyright is defined by Title 17 of the U.S. Code and by similar law in many other countries. It grants exclusive rights to the author of an original work in any tangible means of expression, namely the rights to reproduce the copyrighted work; distribute copies; prepare derivative works; distribute copies of derivative works; and (for certain kinds of work) perform or display it. Because the rights are exclusive, the author can prevent others from exercising them, except as allowed by "fair use." The author can also grant others any or all of the rights or any part of them; one of the functions of a software license is to grant such rights and define the conditions under which they are granted.

5. Software licensing and FOSS licenses

Traditional proprietary licenses typically grant a minimum of rights licensees need to use the licensed software, retaining control of software the licensor has produced and restricting the access and rights outsiders can have. In contrast, FOSS licenses are designed to encourage sharing and reuse of software, and typically grant as many rights as possible consistent with that goal. FOSS licenses are conventionally classified as academic or reciprocal. An *academic* FOSS license, such as the Berkeley Software Distribution (BSD) license, MIT license, or Apache Software License, grants nearly every copyright right for components and their source code, and imposes few obligations. Anyone can use the software, create derivative works from it, or include it in proprietary projects. Typical academic obligations are simply to not remove the copyright and license notices (University of California, 1998).

Reciprocal or *copyleft* FOSS licenses take a more active stance towards the sharing and reuse of software by imposing an obligation that reciprocally-licensed software only be combined (for various definitions of "combined") with software that is in turn also released under the reciprocal license. The primary goal of reciprocity in licenses is to increase the amount of FOSS by encouraging developers to bring additional software into the FOSS commons, and to prevent improvements to OSS software from vanishing behind proprietary licenses. Example reciprocal licenses are the GPL licenses, the Mozilla Public License (MPL), and the Common Public License (CPL).

It has been common practice for a FOSS project to choose a single license under which all its distributions are released, and to require developers to contribute their work under conditions compatible with that license. For example, the Apache Contributor License Agreement (Apache, 2009) grants enough of each contributor's rights as an author to the Apache Software Foundation for

it to license the resulting systems uniformly under the Apache license. This sort of rights regime, in which the rights to a system's components are homogeneously granted and the system has a single well-defined OSS license, was universal in the early days of FOSS and continues to be widely practiced.

6. A theory of software licenses and their application

6.1. Legal grounding

Under U.S. law and the law of most countries, a license can be either a bare license or a contract license. A *bare license* simply grants one or more copyright or patent rights from the copyright and/or patent holder to another person. A *contract license* is constructed in the form of a contract, involving an exchange of promises between the parties. In a contract license, the licensor grants one or more rights in exchange for some consideration from the licensee receiving them. The consideration given by the licensee may be very small, as little as "a peppercorn" in the traditional explanation, but it cannot be nothing. However, in addition to a payment or other obvious consideration, it can take the form of "(a) an act other than a promise, or (b) a forbearance, or (c) the creation, modification, or destruction of a legal relation" (American Law Institute, 1981). In FOSS licensing the fundamental consideration is typically interpreted as the licensee's "detrimental reliance" on the licensed rights, or in other words the reliance on the software that would be to the licensee's detriment if the software were withdrawn. A license, whether bare or contract, can also impose specific non-consideration obligations as a condition of the license grant. Most FOSS licenses are drawn as contract licenses in order to benefit from the well established case law on interpretation of contract provisions, with the exception of the GPL family of licenses which are drawn as bare licenses (Rosen, 2005; Gordon, 1989; Determann, 2006; Guadamuz, 2009; Hillman and O'Rourke, 2009; Kemp, 2009).

In a seminal article published in 1913 and cited up to the present day, Hohfeld presented a theory by which he proposed to resolve the imprecise terminology and ambiguous classifications he found in use for legal relationships (Hohfeld, 1913). He set out a system of eight *jural relations* to express and classify all legal relationships between parties. The eight relations are listed below followed by informal descriptions in parentheses. The first four regulate ordinary actions:

- *right* (may)
- *no-right* (cannot)
- *duty* (must)
- *privilege* (need not)

while the second four regulate actions that change legal relationships:

- *power* (may change relation R)
- *disability* (cannot change R)
- *liability* (must accept changes in R)
- *immunity* (need not accept changes in R).

Each relation has an *opposite* relation whose sense is its opposite, and a *correlative* relation whose sense is its complement. The relations are shown with their opposites and correlatives in Figure 2.

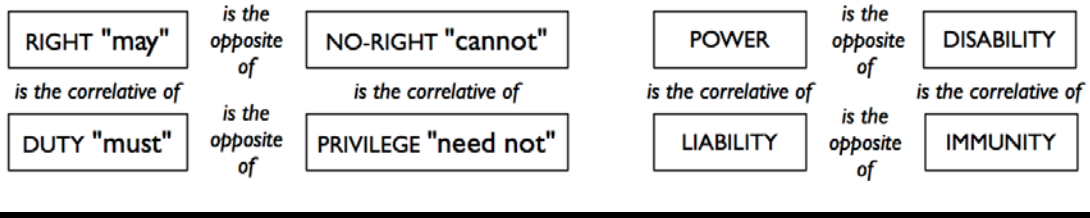


Figure 2. Hohfeld's jural relations, indicating opposites and correlatives

It has been argued that Hohfeld's jural relations provide a sound basis for legal expression, first by Hohfeld himself and more recently with specifics and supporting data by Allen and Saxon (1995), and the relations have been used by many other researchers in law and other fields, such as Balkin (1991), Gordon (1989), and Humphris-Norman (2009) in law, Daskalopulu and Sergot (1997) and McCarty (2002) in artificial intelligence, and Huhns and Singh (1998) and Siena et al. (2008) in software engineering.

6.2. A basic meta-model for licenses

We derived a basic meta-model for software licenses by analyzing the structures defined in law and dividing them into constituent parts, as shown in Figure 3. An enactable right or obligation is composed of the action performed, the actor performing it, and the modality expressing whether the action is allowed, required, or forbidden in terms of Hohfeld relations. The two parties involved in a license are the licensor and licensee. In addition, it is clear that while a right or obligation may involve virtually any action, those actions that are regulated by copyright and patent law are distinguished as of particular importance. Overall, a license is modeled as one or more rights to be granted, each right corresponding to zero or more obligations to be performed in exchange. Both rights and obligations are modeled as tuples of actor, modality, and action, with rights having modalities "may" or "need not" and obligations "must" or "cannot."

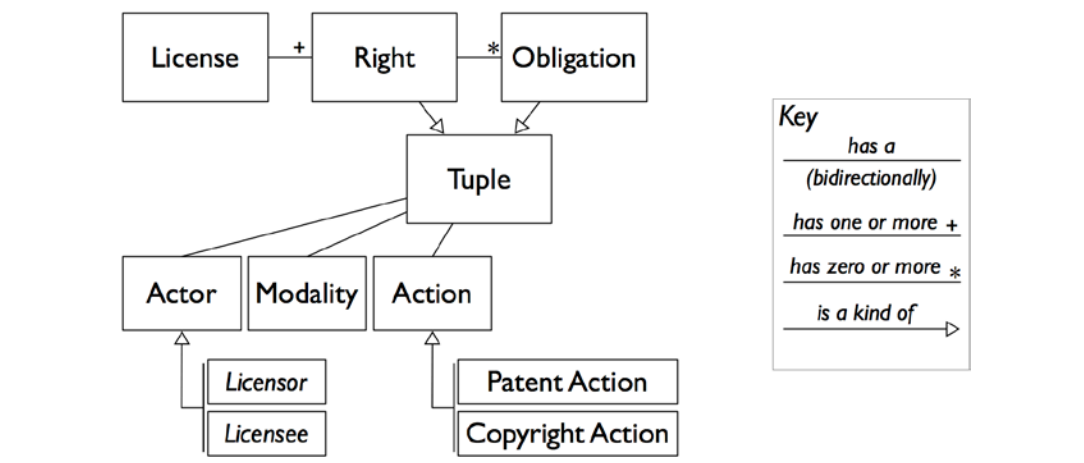


Figure 3. Basic meta-model for software licenses

6.3. Reasoning rules in this meta-model

The legal grounding suggests and supports two rules of implication for licenses following this model. The first rule expresses the contractual exchange of promises: a right is granted only if its corresponding obligations are fulfilled. The second rule expresses Hohfeld's correlatives: an obligation can be fulfilled only if its actor has the right to fulfill it. If that right is not immediately available, then the actor may have to obtain it by first fulfilling other obligations. The obligation's

correlative is that prerequisite right. If the obligation is that the actor "must" perform the action, then the prerequisite right is that he "may" perform it, while if the obligation is that the actor "cannot" perform it, then the requisite right is that he "need not."

The chaining of these two rules is sketched in Figure 4, in which the desired right depends (by rule 1) on a necessary obligation, which in turn depends (by rule 2) on the obligation's correlative right as a prerequisite. If the prerequisite right is not immediately available, it must be obtained by performing the obligations necessary for it, and so forth until all the obligations are performable.

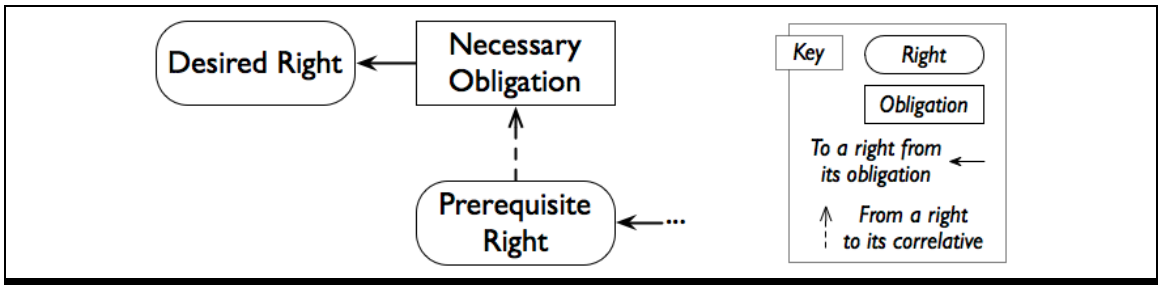


Figure 4. Chain of rights and obligations

This chaining is illustrated most clearly by the reciprocal licenses, in which (in general terms) the right to distribute a derivative work derived from a reciprocally-licensed original carries the obligation to distribute it only under that reciprocal license. The correlative right is the right to distribute copies under that license; if all the originals on which the derivative was based were obtained under the same license, then that right is immediately available, but if the originals were under two or more different licenses then it may not be immediately possible to distribute the derivative under all of them, as their provisions may conflict. If so the correlative right would have to be obtained by recursively fulfilling whatever obligations (if any) would grant it. It may not be possible to obtain the correlative right at all, either because the relevant licenses do not grant it under any circumstances, or because the necessary obligations are in conflict and cannot be simultaneously fulfilled. In such a case, the desired right is not available and there is no way to obtain it.

6.4. Extending the theory using empirical data

The view and analysis of software licenses as described in the previous section is appropriate as a basis for human reasoning about licenses in the abstract, but is not sufficient for formal analysis and automation. As an example, how does GPL version 2.0 apply to a specific software component `code.c`? A human being could (with some effort) list the rights and obligations for `code.c` using that view as a basis, but the process could not be automated using it. If the copyright right to distribute copies of `code.c` were desired, a human being could identify the provisions of GPL version 2.0 that applied, but again this could not be automated.

In addition, it was not clear to us whether the theory of the previous section fully accounted for actual software licenses, or whether (and if so to what extent) experts working with licenses use experience and shared tacit knowledge to account for features and interactions not covered by theory.

We conducted a grounded-theory qualitative study of software licenses in order to address these issues. The research questions for the study were:

- Do software licenses match the meta-model in Figure 3?
- What features, if any, are not accounted for by the meta-model?
- What software-architectural structures are referenced in licenses, and how do they interact with the structures in the meta-model?

- In what ways does the meta-model need to be extended in order to support automated calculation, inference, and application of abstract license provisions to concrete software?

The research questions for the study were developed and elaborated using the Goal Question Metric approach (Basili et al., 1994). The texts of ten licenses were analyzed first by semantic parameterization (Breaux et al., 2008) to account for differences in terminology and conceptualization between the studied licenses, and were then chunked and open-coded systematically in accordance with qualitative practice (Creswell, 2003; Miles and Huberman, 1994). The extensions to the meta-model were generated from the data using a grounded-theory approach (Glaser and Strauss, 1967; Corbin and Strauss, 2007).

The ten licenses analyzed were:

- Apache 2.0
- Berkeley Software Distribution (BSD)
- Common Public License (CPL)
- Eclipse Public License 1.0 (EPL)
- GNU General Public License 2 (GPL)
- GNU Lesser General Public License 2.1 (LGPL)
- MIT
- Mozilla Public License 1.1 (MPL)
- Open Software License 3.0 (OSL)
- Corel Transactional License (CTL)

These ten were chosen to represent a variety of kinds of licenses, and include one proprietary (CTL), three academic (Apache, BSD, MIT), and six reciprocal licenses (CPL, EPL, GPL, LGPL, MPL, OSL) that take varying approaches in implementing copyleft. The nine FOSS licenses account for approximately 75% of FOSS software at the time of this writing (Black Duck, 2009), and include the licenses presented by Rosen as representative (2005). The texts of the FOSS licenses are from the Open Source Initiative web site (OSI, 2009), and the text of the proprietary CTL license is from Corel's web site (Corel, 2009).

The four stages of the analysis were as follows. First, we performed a word and term level analysis of the texts. We disambiguated the text in several ways. We identified forward and backward references and linked them to their references, identified synonyms, and distinguished polysemes expressing different meanings with identical wording. We identified terms of art such as "Derived Work" from copyright law, and specialized terms such as "work based on the Program" for GPL and "Electronic Distribution Mechanism" for MPL that have a license-specific meaning. We then constructed (automatically) a concordance of the resulting text: an index giving each instance of the significant words in the licenses. We excluded minor words such as articles, conjunctions, and prepositions whose use in a particular license carries no specialized meaning. Then we (automatically) tagged each sentence with its section, paragraph, and sentence sequence numbers. Figure 5 shows a portion of the concordance for GPL.

0. S2.0p1s1 This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General_Public_License.
S2.0p1s2 The "Program", below, refers to any such program or work, and a "work_based_on_the_Program" means either the Program or any derivative_work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language.
S2.0p1s3 (Hereinafter, translation is included without limitation in the term "modification".) S2.0p1s4 Each licensee is addressed as "you".

Figure 5. GPL version 2.0 concordance, section 2.0 paragraph 1

Second, we identified parts of each license having no legal force, such as GPL 2's "Preamble" section, or that dealt with anything other than copyright, such as patents, trademarks, implied warranty, or liability. We iterated this step using the concordance to confirm our identifications. The remainder of our analysis focused on copyright.

Third, using the concordances across the licenses, and guided by legal work on FOSS licenses (Fontana et al., 2008; Rosen, 2005; St. Laurent, 2004), we identified words and phrases with the same intensional meaning, and textual structures parallel among the licenses. Working from these, we iterated to identify natural language patterns in the licenses, and patterns of reference among license provisions and to the licensed software and its context.

Fourth, we chunked and open-coded the text into segments addressing the same issues, with the set of issues arising from the texts themselves. We compared the categories and individual segments of text with the meta-model, noting cases of agreement and cases in which the meta-model did not account for significant features. As the iterative process progressed, new meta-model structures arose from the categories and from the structures expressed in the texts. The iteration converged on an extended meta-model shown in Figure 6 and elaborated in Figure 7 that summarizes what we found as a set of semantic relations. Our analysis confirmed that each license we examined consists of one or more *rights*, each of which entails zero or more *obligations*. Rights and obligations have the same structure, a relation expressed as a tuple comprising an actor (the licensor or licensee), a modality, an action, an object of the action, and possibly a license referred to by the action.

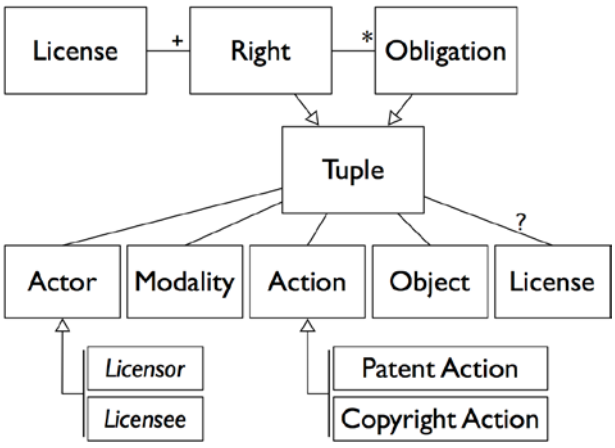
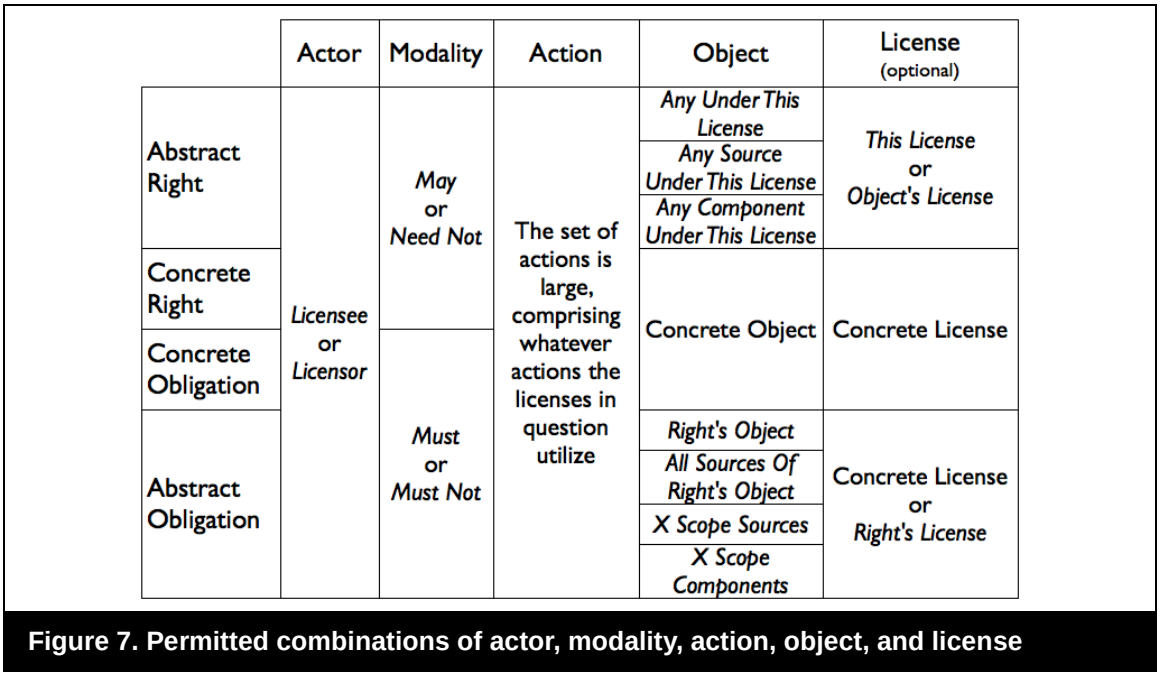


Figure 6. The extended meta-model for licenses, adding Object and License

We found a wide variety of license actions, some defined in copyright law and others defined by specific licenses. Many of the objects within actions are references, and the kinds of references we

identified in the ten analyzed licenses are listed in Figure 7, along with the contexts in which they occur. The objects for rights are self-explanatory from their names. If the object of an obligation is *"Right's Object"* then that obligation is instantiated with the same object as its corresponding right (e.g. hypothetical right "Licensee may modify X" with obligation "Licensee must publish modifications of X"); *"All Sources of Right's Object"* results in as many obligations as the right's object has source files, with an instance of the obligation for each one; *"X Scope Components"* for each reciprocal license X, results in as many obligations as there are components within the scope of propagation of X's obligations that includes the right's object, and *"X Scope Sources"* analogously for sources of components in that scope.



	Actor	Modality	Action	Object	License (optional)
Abstract Right	Licensee or Licensor	May or Need Not	The set of actions is large, comprising whatever actions the licenses in question utilize	Any Under This License	This License or Object's License
				Any Source Under This License	
				Any Component Under This License	
Concrete Right				Concrete Object	Concrete License
Concrete Obligation					
Abstract Obligation		Must or Must Not		Right's Object	Concrete License or Right's License
				All Sources Of Right's Object	
				X Scope Sources	
				X Scope Components	

Figure 7. Permitted combinations of actor, modality, action, object, and license

Note that while some objects such as *"Right's Object"* refer only to the same entity that the corresponding right does, others such as *"All Sources of Right's Object"* and *"X Scope Components"* induce more complex patterns. In order to instantiate an abstract right or obligation containing such an object, it is necessary to have information about the system they refer to, such as the way in which components are interconnected, in order to determine each license scope in that system, or the sources from which a component was built. We call the abstraction of the system that provides this information the *license architecture* of the system, and discuss it in the next section.

As a cross-check on the validity of our analysis, we implemented the license meta-model in Java and expressed the licenses using it, generated a restricted natural language (RNL) form for each license right and obligation, and compared them with the originals. The RNL text for an example *abstract right* (one not bound to a specific object and context) extracted from the BSD license is:

Licensee · may · distribute <*Any Source*> under <*This License*>

where "distribute under" is an action reserved by copyright law and the abstract object <*Any Source*> quantifies the right over all sources licensed under the license containing the right (here, BSD); an example concrete obligation (one bound to a specific context) is:

Licensee · must · retain the [BSD] copyright notice in [`file.c`]

where "retain the copyright notice" is an action that is not reserved by copyright law, BSD is the concrete license the action references, and `file.c` is the concrete object the action references. The

encoded actions contain tokens identifying where the tuple's object and (if present) license are inserted, for example in the GPL action "sublicense % under ^" which becomes "sublicense *OBJECT* under *LICENSE*" for a particular object and license. Figure 8 informally illustrates how actions may contain concrete objects or licenses, references to objects or licenses bound elsewhere, or quantifiers using the information in the license architecture to produce sets of concrete rights or obligations.

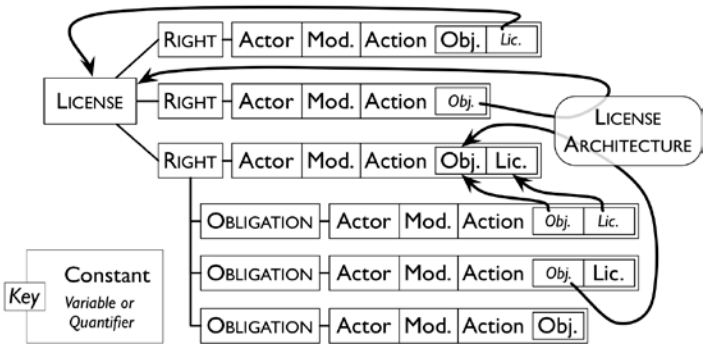


Figure 8. Object/license references, informally

This model of licenses gives a basis for reasoning about licenses in the context of actual systems. The additional information we need about the system is defined by the list of quantifiers that can appear as objects in the rights and obligations. We term this information the *license architecture* (LA). It is an abstraction of the system architecture:

- the set of components of the system;
- the relation mapping each component to its license;
- the relation mapping each component to its set of sources; and
- the relation from each component to the set of components in the same license scope, for each license for which "scope" is defined (e.g. GPL), and from each source to the set of sources of components in the scope of its component.

We tested the internal validity of the study as described above by implementing the meta-model as a Java package, expressing the licenses in the meta-model using the package, automatically re-generating text for each license, and comparing the regenerated text with the original for semantic equivalence (of course the two texts were syntactically dissimilar). At the time of writing, the meta-model has also been validated by a law student specializing in Intellectual Property, an international legal researcher in the area of comparative systems of Intellectual Property, and a law professor in Intellectual Property; these validations were done first informally through discussions of the meta-model and how the legal concepts in licensing and contracts and the specifics of various licenses align with its structures, and then formally through a qualitative analysis in which the legal researcher independently represented two licenses in terms of the meta-model's structure. The key point shown by these validations is that the meta-model is sufficiently expressive to support various legal interpretations. We tested the study's external validity by incorporating the Java package embodying the meta-model into a software architecture tool as described below, and using it to analyze HLS architectures.

In this study, we examined ten software licenses; we will examine additional licenses and fully expect that new license structures will be revealed and new elaborations of the meta-model will arise. However, this does not invalidate the results of the present study. We note that any successful theory will have to account for the features we identify here, and at most will add additional features, not remove any.

7. License architecture of Heterogeneously-Licensed systems

As noted in the previous section, certain elements of software architectures affect how license provisions take effect. A software architecture is composed of *components*, each of which is a "locus of computation and state" in a system, and *connectors* which link them and mediate interactions between them (Shaw et al., 1995). The components and connectors below are significant for one or more FOSS licenses and can affect the license interactions and overall rights and obligations for the system.

7.1. Components

Software source code components. A source code component is one whose source code is available so the component can be modified and rebuilt. These can be either:

- standalone programs,
- libraries, frameworks, or middleware,
- Inter-application script code that coordinates the actions of two or more applications, or
- intra-application script code that controls actions within its application, as for creating Rich Internet Applications using domain-specific languages such as XUL for the Firefox web browser (Feldt, 2007), or "mashups" that combine data, functionality, or presentations from two or more sources to create a new service (Nelson and Churchill, 2006)

Executable components. These components are in binary form, and the source code may not be available for access, review, modification, or possible redistribution (Rosen, 2005). If proprietary, they often cannot be redistributed, and if so will be present in the design- and run-time architectures but not in the distribution-time architecture.

Software services. An appropriate software service, such as Google Docs, provided through a client-server connection to a local or remote server, can be used in place of a source code or executable component.

Application programming interfaces (APIs). Availability of externally visible and accessible APIs is the minimum requirement for an "open system" (Meyers and Oberndorf, 2001). APIs are not and cannot be licensed (Rosen, 2005), and can be used to limit the propagation of some license obligations.

Configured system or subsystem architectures. These are software systems that are used as atomic components of a larger system, and whose internal architecture may comprise components with different licenses, affecting the overall system license.

7.2. Connectors

Methods of connection. These include:

- static linking performed permanently at system build time,
- dynamic linking performed temporarily at run time, and
- client-server connections.

Methods of connection affect license obligation propagation, with different methods affecting different licenses. Although there is some controversy over this question, and our statements here should not be taken as legal advice, there appear to be grounds to believe that a client-server connection serves as a license firewall for reciprocal obligations from all current licenses except the Affero GPL (AGPL) version 3.0, which was designed to address client-server connections explicitly, and certain related licenses not otherwise discussed in this article; and more controversially that a dynamic link rather

than a static link blocks the same obligations (Determann, 2006; Rosen, 2007; Stoltz, 2005). At the time of writing, no court has ruled definitely on these questions.

Software connectors. Some connectors are themselves software whose intended purpose is to provide a standard or reusable way of communication through common interfaces, e.g. CORBA, Microsoft .NET, Mono, and Enterprise Java Beans. Software connectors, by sending communication through a standard and thus uncopyrightable interface, may affect the propagation of license obligations.

To minimize license interaction, a problematic-licensed component may be surrounded or encapsulated by what we term a *license firewall,* namely a layer of dynamic links, client-server connections, license shims, or other connectors that block the propagation of specific reciprocal obligations.

7.3. Other information in a license architecture

As noted in the previous section, some license obligations require more information in order to be instantiated, namely the ones that incorporate any object other than *"Right's Object."* A system's license architecture thus includes the relationship from component to sources, which we will not discuss here since it is straightforward from a development point of view. The other such objects *"X Scope Sources"* and *"X Scope Components"* may be inferred from the information already in the license architecture. We note that we expect the license architecture abstraction to be elaborated in parallel with the meta-model as new license structures are identified.

8. Embodying and applying the models and analysis

Figure 9 presents a screenshot of a simple but not trivial HLS that raises the kinds of issues that come up in all HLSs. Its components include a web browser, an email and calendar system, a word processor, and a web application. In the screenshot these four components have been outlined with dashed rectangles. The web browser is Firefox, the email and calendar system is Gnome Evolution, the word processor is AbiWord, and the web application is running in a Red Hat Fedora Linux command window, but similar functionality (with possibly higher quality or better adapted functionality) could be achieved with other components of the same class. This particular system implements a simple office productivity environment for a university research laboratory.

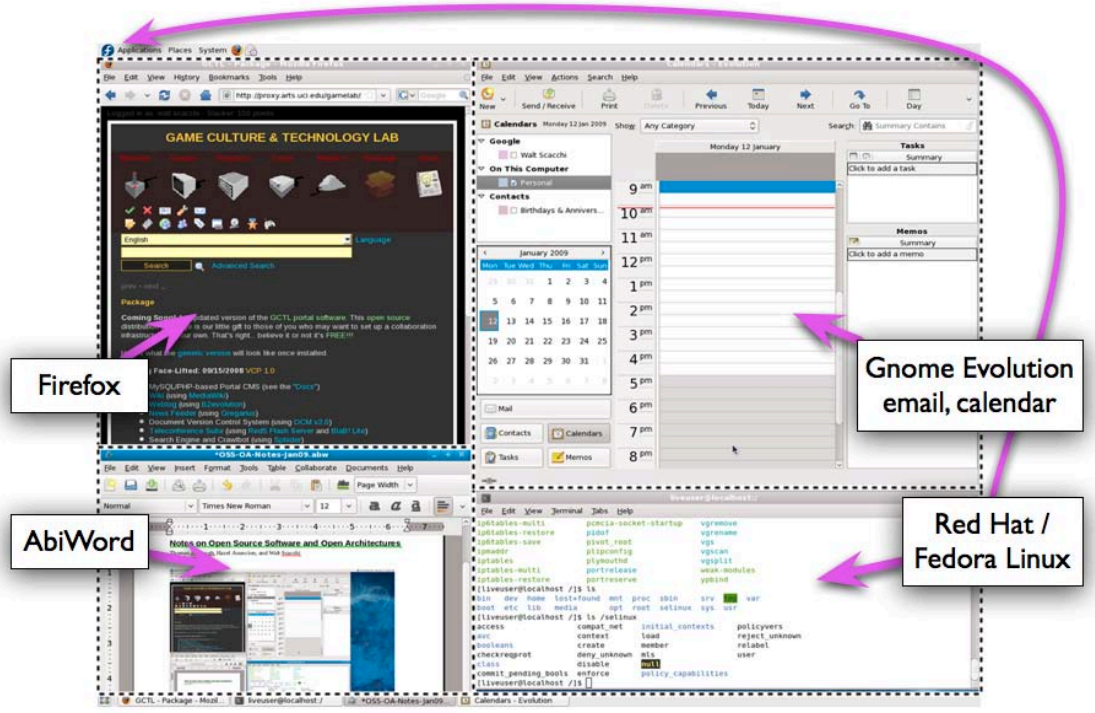


Figure 9. A heterogeneously-licensed composite system

The software architecture of the high level components and their interconnections in this system is shown in Figure 10. Components, shown in upright font, implement pieces of the system's functionality. Each component is associated with a license governing the terms of its use. Connectors, shown in italics, connect and translate if necessary between the interfaces of two components. Where the two interfaces are the same, the connector is identified simply by the API for the two interfaces, and has no license. If the two interfaces differ, then the connector is implemented by a small piece of software, a *shim*, that translates between them, and the shim has or could have a license. Arrows show communication between connectors and components. The components, the connectors, and the topology in which they are arranged constitute the architecture of the system. This architecture shows the system's high-level design.

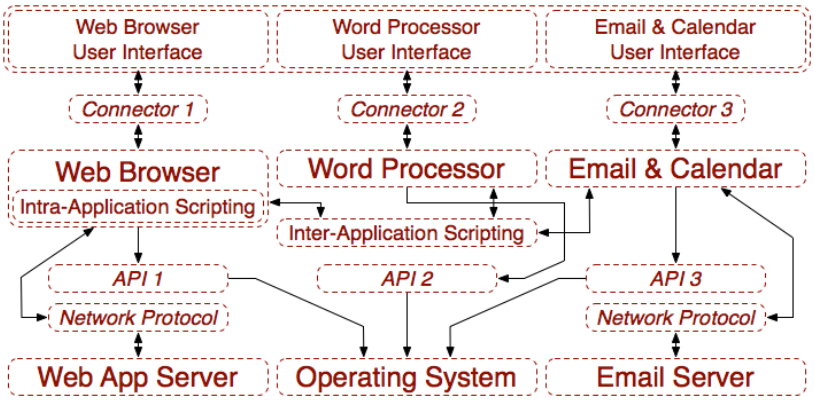


Figure 10. The design-time architecture describing the system

Figure 11 shows the architecture with the abstract components and connectors replaced by specific implementations. In this *instance architecture* describing a particular implementation of the system, the web browser is Firefox, the word processor is AbiWord, connectors 1 through 3 are XWindows calls, and so forth. All the components and connectors are FOSS, though not homogeneously licensed: for example, the Gnome Evolution and AbiWord components are distributed under GPL version 2.0, while the Apache web server component is distributed under the Apache License version 2.0, with which GPL 2.0 is not compatible. The network protocol HTTP, being a data transfer interface conforming to an open standard, has no license. Most of the components and connectors in the architecture are not visible to the user when the system is running, but four of them can be identified in Figure 9, and the user interface (UI) as a whole can be identified as running on a Red Hat Fedora system by the blue script "*f* " in the upper left corner of the screen.

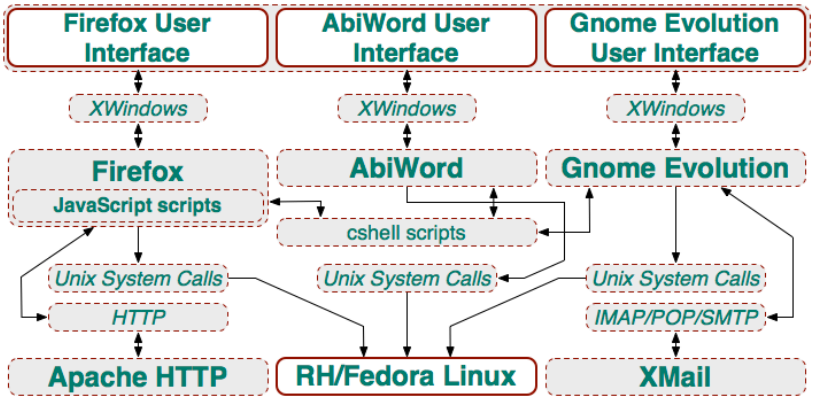


Figure 11. A build-time architecture describing the version running in Figure 9

Figure 12 shows a high-level view of the license architecture of the system of Figure 9, with the mappings to sources and license scopes omitted for clarity. Here each component is represented by its license, with the connectors and connections the same as in the design-time and build-time architectures. For ease of reference, we also provide Figure 13 in which the license architecture is additionally annotated with each component's name.

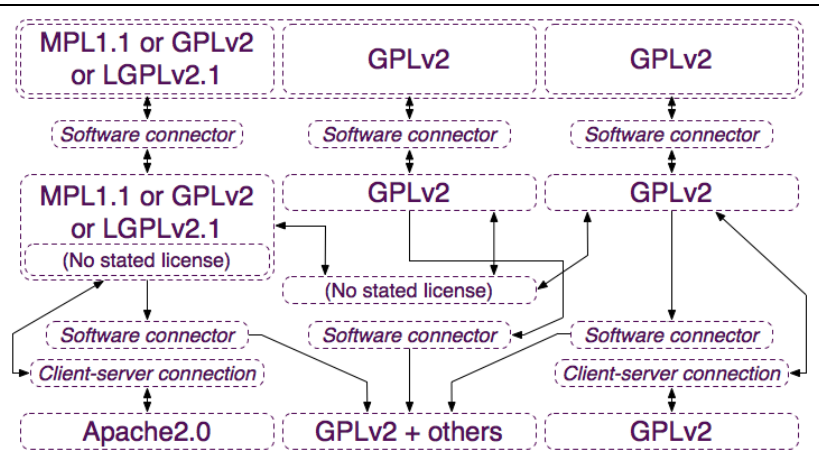


Figure 12. The license architecture of the instantiation of Figure 11

We noted above that GPLv2 and Apache2.0 are not compatible; a reverse-engineering and recovery approach for license analysis, as discussed above in "Related Work" (Gobeille, 2008; Tuunanen et al., 2009; Di Penta et al., 2010) could determine that the system in Figures 9 through 13 *might* have a

license conflict, but without the license architecture it could not determine whether the system *will* have a conflict. In fact, our analysis (confirmed by the software tool) correctly shows that no conflict exists, because the Apache2.0 component is licensed-firewalled from GPLv2 obligations by the client-server connection that blocks its only path to GPLv2 components. Even without this firewall, the conflict could be avoided by accepting Firefox under MPL1.1, since Firefox is distributed through the Mozilla *tri-license* under which recipients can choose any of the three possible licenses.

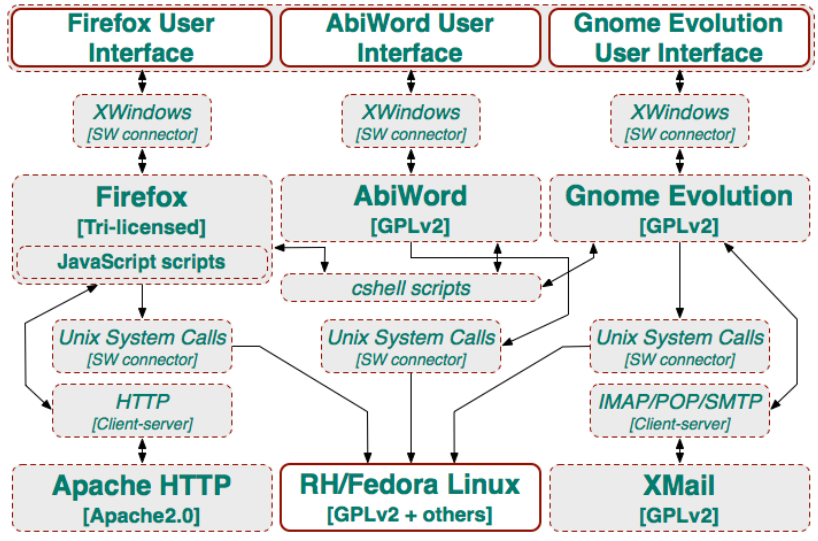


Figure 13. The license architecture, annotated with each component and connector

Since this system has an open architecture, it could be straightforwardly modified to use a proprietary word processor like Corel WordPerfect (or Microsoft Word) in place of AbiWord, for example to obtain different functionality or to be consistent with an enterprise policy for uniformity across systems. There would be no license conflict, since the word processor component in this architecture is firewalled off by client-server connections and software connectors. However, our analysis (confirmed by the tool) identifies that the system no longer has the same rights: anyone can use the system, but the right to distribute the system is no longer available, since the Corel Transactional License (CTL) forbids copying and redistribution. This is not a license conflict, but simply the lack of a potentially desired right. Note that the enterprise could choose to purchase a separate copy of WordPerfect for each system, in which case each individual instance of the system could be distributed (although still not copied); or it could distribute the system minus WordPerfect, with each recipient obtaining his/her own copy of WordPerfect and integrating it into the system. We do not maintain that any of these approaches (AbiWord, separate copies of WordPerfect, or user-integrated WordPerfect) is better or worse than another, merely that each has its own distinct functionality, available rights, and/or build process.

Finally, we outline the effect of a very different choice: replacing the word processor component with a web-based word processing service such as Google Docs. In this case the calculation is not dealing only with licenses, but also with a Terms of Service agreement whose provisions are different. Some rights are unchanged: anyone can use the system, and the system can be redistributed. But in order to achieve this, not only does the enterprise have to replace the word processing component and the inter-application scripts that shim its interfaces, it has to alter the system's architecture. Google Docs will provide word processing services but only through the web browser; the web browser will not be connected through system calls to the Linux component, or through XWindows to the display; and the scripts will be changed to connect to the word processor and browser through the same connection. Again, this choice is not necessarily better or worse than any of the previous three, it merely offers different costs and affordances.

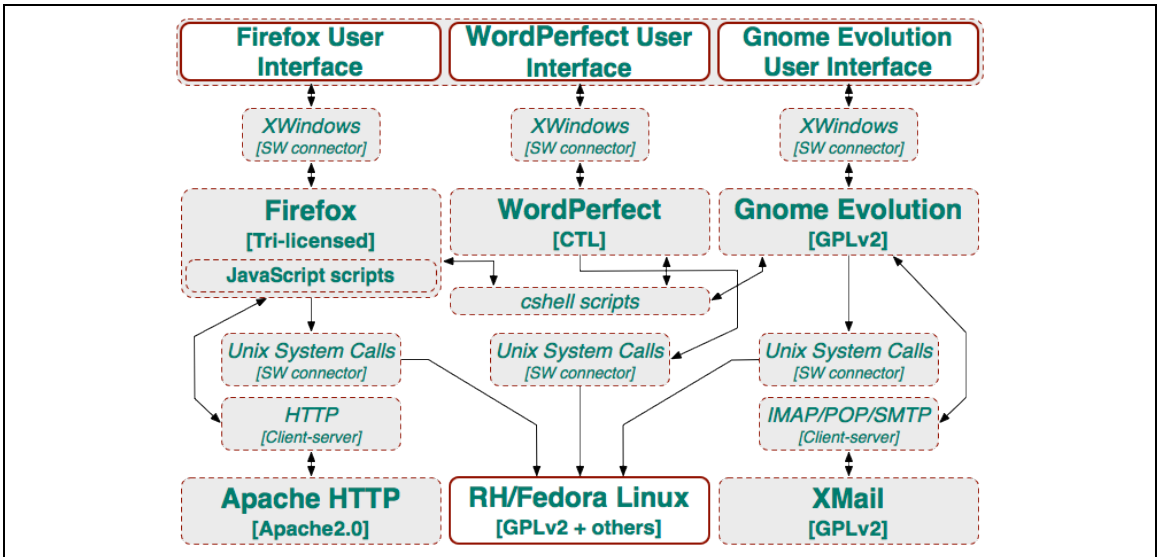


Figure 14. License architecture (annotated) using WordPerfect in place of AbiWord

We chose this simple but not trivial example so that the explanation would be of manageable length. Most enterprise HLSs differ from this one only in scope, not in the kinds of issues that arise. The issues would simply be more numerous, and the paths of license effects would be more complex, necessitating a theory-based automated tool not only for ease and accuracy but also to make the balancing and selection among design choices manageable. As we explored and analyzed OA systems, we realized this same example architecture describes a substantial number of e-business information systems. For example, our department's payroll system also has the architecture of Figure 10, although for that system of course the connectors and scripts are different in order to produce a different specific system function. Using and distributing this or any other HLS within or outside an enterprise has legal implications, due to the varied and possibly conflicting licenses found for the systems' components and the large number of distinct obligations that arise. Organizations with assets need to be concerned that the use or distribution of such HLSs can leave them vulnerable to liabilities due to violations (even if inadvertent) of license terms and conditions. Our work gives a basis from which organizations using and developing HLSs can assess their rights and obligations, and guide HLS acquisition and development in order to achieve not only the desired system functionality, quality, and evolvability, but also the rights that are needed in exchange for obligations that are acceptable.

9. Discussion

Throughout this paper, we have sought to understand and analyze composite software or information systems that constitute an HLS. Here, we discuss some of the consequences that follow from this understanding and analysis.

First, as we saw in related research, there has been a great deal of interest in determining what aspects or features of FOSS development projects are significant contributors to project success. The choices among available FOSS licenses, when applied to a single system as a whole, revealed that certain license choices are positively associated with project success. But, what are we to expect in trying to anticipate potential outcomes when system components may have different licenses? This is part of the challenge for understanding HLSs. Consequently, it is unclear what outcomes for HLS development projects should be anticipated based on prior studies of FOSS success factors, and thus this should be recognized as an outstanding problem for further study. However, it does appear that there is growing awareness that both public and private sector enterprises are moving to develop

composed (or otherwise integrated) systems of systems, and that such HLSs may become more commonplace.

Second, through modeling of FOSS licenses and analysis of HLSs rendered as an OA with a license attributed to each component and to the other components to which it is directly linked (or interconnected), it becomes possible to articulate which licenses or constraints match or conflict with one another. The case study in the previous section helps demonstrate this. Further, when license constraints are discovered, the nature of each conflict (e.g. "must not redistribute software" versus "must redistribute software if modified") then gives rise to possible ways in which that conflict can be mitigated or resolved. These include reconfiguring the system architecture, replacing conflicting components with compatible alternatives under non-conflicting licenses, or inserting license mediators. However, if an OA for the system is not available and cannot be derived from available information (as is the case for the Unity3D software package identified in the Introduction), then it may not be possible to analyze or reliably determine what license constraints apply, or whether any are in conflict; and this places system consumers or users in an uncertain situation. Thus, understanding and analyzing HLSs requires a model of the constraints found in the system component licenses, along with an OA rendering of the system and corresponding license constraints.

Lastly, once the analysis of an HLS has been conducted and identified license conflicts resolved or eliminated, then what remains is a set of license constraints that characterize the rights provided and the obligations to be fulfilled by the systems consumers or users. But this unified set of license constraints in general does not conform to an existing single license (since if it did, the system would not be considered an HLS), so then what is the license in effect? We believe one way to address this is to consider the resulting set of non-conflicting rights and obligations for an HLS to constitute a new kind of license which we can designate as a virtual license. However, we do not anticipate that such a virtual license will simply be treated as a new type of license (and thus submitted to a license review authority like the Open Source Initiative). Instead, a virtual license will simply represent a set of rights and obligations within corporate policy space or license regime that an enterprise finds acceptable, manageable, and traceable (cf. Dinkelacker et al., 2002). In this regard, we may expect that future study could determine whether enterprises can simply specify which resulting license rights or obligations they will accept and fulfill, and which they will not, independent of the name or pedigree of the FOSS license or project from which those rights and obligations may have originated.

10. Conclusion

In this paper, we introduce the problem of how to understand and analyze heterogeneously-licensed systems composed and built from FOSS and/or proprietary components, each of which may be subject to different licenses with different rights and obligations. We started by providing examples of HLSs that are beginning to appear, and that help articulate the challenge of understanding how license constraints may interact, match, or conflict within the system's architecture, whether at design time, build time, or run time. We found that such analysis requires an explicit semantic model of the license rights and obligations, along with an open architectural rendering that can be attributed with the corresponding license constraints. Without both of these, it may not be possible to systematically analyze or comprehensively understand what license constraints apply to the whole system (or system of systems). But with both, we demonstrate it is possible to conduct such an analysis, and to determine which license constraints match, subsume, or conflict with one another. It also becomes possible to explore and readily evaluate alternative system architectures that may resolve conflicting license constraints, particularly when analysis of HLSs can be supported with an automated environment, such as the one we have developed to this end.

The contribution presented in this paper constitutes identification of a new, emerging category of problem, which we identify as the challenge of heterogeneously-licensed systems, along with an empirically grounded approach and design theory for analyzing and modeling FOSS license rights and obligations, and to modeling and analyzing the architecture of an HLS, in order to determine the resulting set of license constraints that applies to the composed system. We described and

demonstrated our approach, along with the kinds of information we employ, which are empirically observable when FOSS components are employed, and when all participating software license rights and obligations can be systematically modeled. Whether our approach is relevant to legal interpretations or framings for the purpose of asserting defensible intellectual property rights is beyond the scope of our study (and expertise) and thus remains an open question for future study.

Acknowledgments

The authors extend their thanks to the anonymous reviewers of earlier versions of this paper for their thoughtful and insightful evaluations.

Support for this research is through grant #0808783 from the National Science Foundation, and also #N00244-10-1-0038 from the Acquisition Research Program at the Naval Postgraduate School. No review, approval, or endorsement implied.

References

Allen, L. E. and Saxon, C. S. (1995). Better Language, Better Thought, Better Communication: The A-Hohfeld Language for Legal Analysis. In *5th International Conference on Artificial Intelligence and Law (ICAIL'95)*:219–228.

American Law Institute (1981). *The Restatement (Second) of Contracts*.

Alspaugh, T. A., Asuncion, H.U., and Scacchi, W. (2009a). Intellectual Property Rights Requirements for Heterogeneously-Licensed Systems. In *17th IEEE International Requirements Engineering Conference (RE'09)*:24–33.

Alspaugh, T. A., Asuncion, H. U., and Scacchi, W. (2009b). The Role of Software Licenses in Open Architecture Ecosystems. In *First International Workshop on Software Ecosystems (IWSECO-2009)*: 4–18.

Apache Software Foundation (2009). Individual Contributor License Agreement version 2.0. http://www.apache.org/licenses/icla.txt

Asay, M. (2008). In Acquiring Merrill Lynch, must Bank of America Open Source Its Software? CNET News, http://news.cnet.com/8301-13505_3-10043029-16.html, 16 September 2008.

Balkin, J. M. (1991). The Promise of Legal Semiotics. University *of Texas Law Review,* 69:1831–1852.

Basili,V. R., Caldiera, G., and Rombach, H. D. (1994). The Goal Question Metric approach. In *Encyclopedia of Software Engineering*:528–532, John Wiley and Sons.

Black Duck Software (2009). Top 20 Most Commonly Used Licenses in Open Source Projects. http://www.blackducksoftware.com/oss/licenses

Breaux, T. D., Antón, A. I., and Doyle, J. (2008). Semantic Parameterization: A Process for Modeling Domain Descriptions. *ACM Transactions on Software Engineering and Methodology,* 18(2).

Brown, A.W., and Booch, G. (2002). Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors. In: *Software Reuse: Methods, Techniques, and Tools (ICSR-7)*.

Corbin, J. M. and Strauss, A. C. (2007). *Basics of Qualitative* Research*: Techniques and Procedures for Developing Grounded Theory.* SAGE Publications.

Corel Transactional License (2009). http://apps. corel.com/clp/terms.html.

Creswell, J. W. (2003). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* SAGE Publications, Thousand Oaks, CA, USA.

Daskalopulu, A. and Sergot, M. (1997). The Representation of Legal Contracts. *AI & Society,* 11(1-2):6–17.

de Laat, P.B. (2007). Governance of Open Source Software: State of the Art, *J. Management and Governance,* 11(2), 165–177.

Determann, L. (2006). Dangerous Liaisons – Software Combinations as Derivative Works? Distribution, Installation, and Execution of Linked Programs Under Copyright Law, Commercial Licenses, and the GPL. *Berkeley Technology Law Journal,* 21(4).

Di Penta, M., German, D. M., Gueheneuc, Y.-G., and Antoniol, G. (2010). An Exploratory Study of the Evolution of Software Licensing. To appear: *29th International Conference on Software Engineering (ICSE '10).*

Dinkelacker, J., Garg, P. K., Miller, R., and Nelson, D. (2002). Progressive Open Source, *Proc. 24th Intern. Conf. Software Engineering*, Orlando, FL, 177–184.

Elliott, M. S., and Scacchi, W. (2003). Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering Collaboration. *ACM International Conference on Supporting Group Work (GROUP'03)*:21–30.

Elliott, M. S., and Scacchi, W. (2008). Mobilization of Software Developers: The Free Software Movement. *Information Technology & People*, 21(1):4–33.

Feldt, K. (2007). *Programming Firefox: Building Rich Internet Applications with XUL.* O'Reilly Media.

Fontana, R., Kuhn, B.M., Moglen, E., Norwood, M., Ravicher, D.B., Sandler, K., Vasile, J., and Williamson, A. (2008). *A Legal Issues Primer for Open Source and Free* Software *Projects*, Software Freedom Law Center, Version 1.5.1. http://www.softwarefreedom.org/resources/2008/foss-primer.pdf

Free Software Foundation (1991). GNU General Public License Version 2. http://opensource.org/licenses/gpl-2.0.php

Free Software Foundation (1999). GNU Lesser General Public License Version

2.1. http://opensource.org/licenses/lgpl-2.1.php
Free Software Foundation (2007a). GNU Affero General Public License Version 3. http://opensource.org/licenses/agpl-v3.html
Free Software Foundation (2007b). GNU General Public License Version 3. http://opensource.org/licenses/gpl-3.0.html
Free Software Foundation (2007c). GNU Lesser General Public License Version 3. http://opensource.org/licenses/lgpl-3.0.html
German, D. M. and Hassan, A. E. (2009). License Integration Patterns: Addressing License Mismatches in Component-based Development. In *Proc. 31st International Conference on Software Engineering (ICSE '09),* Vancouver, BC, Canada, 188–198.
Glaser, B. G. and Strauss, A. L. (1967). *The* Discovery *of Grounded Theory: Strategies for Qualitative Research.* Aldine Publishing Company.
Gobeille, R. (2008). The FOSSology project. In *International Working Conference on Mining Software Repositories (MSR'08)*:47–50.
Gordon, W. J. (1989). An Inquiry into the Merits of Copyright: The Challenges of Consistency, Consent, and Encouragement Theory. *Stanford Law Review,* 41(6):1343–1469.
Guadamuz, A. (2009). The License/Contract Dichotomy in Open Licenses: A Comparative Analysis. *University of La Verne Law Review,* 30(2):101–116.
Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly,* 28(1):75–105.
Hillman, R. A. and O'Rourke, M. A. (2009). Rethinking Consideration in the Electronic Age. *Hastings Law Journal,* 61:311–336.
Hohfeld, W. N. (1913). Some Fundamental Legal Conceptions as Applied in Judicial Reasoning. *Yale Law Journal,* 23(1):16–59.
Huhns, M. N. and Singh, M. P. (1998). Agent Jurisprudence. *IEEE Internet Computing,* 2(2):90–91.
Humphris-Norman, D. O. *Justice, Rights, and Jural Relations: A Philosophy of Justice and Its Relationships.* University of Southampton. 2009.
Kemp, R. (2009). Current Developments in Open Source Software. *Computer Law and Security Review,* 25(6):569–582.
McCarty, L. T. (2002). Ownership: A Case Study in the Representation of Legal Concepts. *Artificial Intelligence and Law,* 10(1-3):135–161.
Meyers, B. C. and Oberndorf, P. (2001). *Managing Software Acquisition: Open Systems and COTS Products.* Addison-Wesley Professional.
Miles, M. B. and Michael Huberman, M. (1994). *Qualitative Data Analysis: An Expanded Sourcebook.* SAGE Publications.
Mozilla Foundation (2009). Mozilla Code Licensing. http://www.mozilla.org/MPL/.
MySQL (2006). MySQL Licensing Policy. http://www.mysql.com/about/legal/licensing/
Nelson, L. and Churchill, E. F. (2006). Repurposing: Techniques for Reuse and Integration of Interactive Systems. In *International Conference on Information Reuse and Integration (IRI-08)*:490.
OSI (2009). Open Source Initiative. http://www.opensource.org/.
Oreizy, P. (2000). *Open Architecture Software: A Flexible Approach to Decentralized Software Evolution*. PhD Thesis, Information and Computer Science, University of California. http://www.ics.uci.edu/~peymano/papers/thesis.pdf
Roberts, J. A., Hann, I.-H., and Slaughter, S. A. (2006). Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science,* 52(7):984–999.
Rosen, L. (2007). Comments on GPLv3. http://www.rosenlaw.com/GPLv3-Comments.htm
Rosen, L. (2005). *Open Source Licensing: Software Freedom and Intellectual Property Law.* Prentice Hall.
Scacchi, W. and Alspaugh, T.A. (2008). Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense, *Proc. 5th Annual Acquisition Research Symposium*, Vol. 1, 230-244, Naval Postgraduate School, Monterey, CA.
Sen, R. (2007). A Strategic Analysis of Competition between Open Source and Proprietary Software, *J. Management Information Systems*, 24(1), 233–257. Summer.
Sen, R., Subramaniam, C., and Nelson, M. (2008). Determinants of the Choice of Open Source

Software Licenses, *J. Management Information Systems*, 25(3), 207-240, Winter.

Shaw, M., DeLine, R., Klein, D. V., Ross, T. L., Young, D. M., and Zelesnik, G. (1995). Abstractions for Software Architecture and Tools to Support Them. *IEEE Transactions on Software Engineering,* 21(4):314–335.

Siena, A., Mylopoulos, J., Perini, A., and Susi, A. (2008). From Laws to Requirements. In *First International Workshop on Requirements Engineering and Law (RELAW'08)*:6–10.

Stewart, K.J., Ammeter, A.P., and Maruping, L.M. (2006). Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects, *Information Systems Research*, 17(2), 126–144.

Stoltz, M. L. (2005). The Penguin Paradox: How the Scope of Derivative Works in Copyright Affects the Effectiveness of the GNU GPL. *Boston University Law Review,* 85(5):1439–1477.

Subramaniam, C., Sen. R., and Nelson, M. (2009). Determinants of Open Source Software Project Success: A Longitudinal Study, *Decision Support Sys.*, 46(2), 576–585.

St. Laurent, A. M. (2004). *Understanding Open Source and Free Software Licensing.* O'Reilly Media, Inc., Sebastopol, CA.

Tuunanen, T., Koskinen, J., and Kärkkäinen T. (2009). Automated Software License Analysis. *Automated Software Engineering*, 16(3-4):455–490.

Unity Technologies (2009). Unity End User License Agreement. http://unity3d.com/unity/ unity-end-user-license-2.x.html.

University of California, Berkeley (1998). The BSD License. http://opensource.org/licenses/bsd-license.php

Ven, K. and Mannaert, H. (2008). Challenges and Strategies in the Use of Open Source Software by Independent Software Vendors. *Information and Software Technology,* 50(9-10), 991–1002.

About the Authors

Thomas A. Alspaugh is Visiting Assistant Researcher in the Institute for Software Research at the University of California, Irvine, and Adjunct Assistant Professor at Georgetown University. He resides in the Washington, DC area. After working in industry as a developer, project lead, and development manager (at IBM, Data General, and other companies), and as a computer scientist in the Naval Research Laboratory's Software Cost Reduction project, he received his PhD in Computer Science from North Carolina State University in 2002. From 2002 to 2008 he was Assistant Professor in Informatics at the University of California, Irvine. His research interests include software requirements, the interaction between software and law, and open-source licensing; see http://www.thomasalspaugh.org/.

Walt Scacchi is Senior Research Scientist and Research Faculty Member in the Institute for Software Research, and also Director of Research at the Center for Computer Games and Virtual Worlds, both at University of California, Irvine. He received a PhD in Information and Computer Science at UC Irvine in 1981. From 1981 to 1998, he was a professor at the University of Southern California. Dr. Scacchi returned to UC Irvine in 1999. His research interests include open source software development, computer game culture and technology, virtual worlds for modeling and simulating complex engineering and business processes, developing decentralized heterogeneous information systems, software acquisition, and organizational analysis of system development projects. Dr. Scacchi is an active researcher with more than 150 research publications, and has directed more than 50 externally funded research projects. He has also had numerous consulting and visiting scientist positions with more than 25 firms or institutes, including four start-up ventures. His recent activities and research publications can be found at http://www.ics.uci.edu/~wscacchi.

Hazeline Asuncion is an Assistant Professor at the University of Washington, Bothell. Dr. Asuncion received her PhD in computer science from the University of California, Irvine, in 2009. Previously, she was a Postdoctoral Researcher in the Institute for Software Research at the University of California, Irvine. She has also worked in industry in a variety of roles: as a software engineer at Unisys Corporation and as a traceability engineer at Wonderware Corporation where she designed a successful in-house traceability system. Her research emphasis is on traceability, the identification and visualization of related information units that are often distributed and represented heterogeneously.