NPS-AM-09-026

EXCERPT FROM THE

PROCEEDINGS

OF THE

SIXTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM

SOFTWARE LICENSES, OPEN SOURCE COMPONENTS, AND OPEN ARCHITECTURES

Published: 22 April 2009

by

Thomas Alspaugh, Hazel Asuncion and Walt Scacchi

6th Annual Acquisition Research Symposium of the Naval Postgraduate School:

Volume I: Defense Acquisition in Transition

May 13-14, 2009

Approved for public release, distribution is unlimited.

Prepared for: Naval Postgraduate School, Monterey, California 93943

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

The research presented at the symposium was supported by the Acquisition Chair of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

To request Defense Acquisition Research or to become a research sponsor, please contact:

NPS Acquisition Research Program
Attn: James B. Greene, RADM, USN, (Ret)
Acquisition Chair
Graduate School of Business and Public Policy
Naval Postgraduate School
555 Dyer Road, Room 332
Monterey, CA 93943-5103
Tel: (831) 656-2092
Fax: (831) 656-2253
E-mail: jbgreene@nps.edu

Copies of the Acquisition Sponsored Research Reports may be printed from our website www.acquisitionresearch.org

Conference Website:
www.researchsymposium.org

Proceedings of the Annual Acquisition Research Program

The following article is taken as an excerpt from the proceedings of the annual Acquisition Research Program. This annual event showcases the research projects funded through the Acquisition Research Program at the Graduate School of Business and Public Policy at the Naval Postgraduate School. Featuring keynote speakers, plenary panels, multiple panel sessions, a student research poster show and social events, the Annual Acquisition Research Symposium offers a candid environment where high-ranking Department of Defense (DoD) officials, industry officials, accomplished faculty and military students are encouraged to collaborate on finding applicable solutions to the challenges facing acquisition policies and processes within the DoD today. By jointly and publicly questioning the norms of industry and academia, the resulting research benefits from myriad perspectives and collaborations which can identify better solutions and practices in acquisition, contract, financial, logistics and program management.

For further information regarding the Acquisition Research Program, electronic copies of additional research, or to learn more about becoming a sponsor, please visit our program website at:

www.acquistionresearch.org

For further information on or to register for the next Acquisition Research Symposium during the third week of May, please visit our conference website at:

www.researchsymposium.org

THIS PAGE INTENTIONALLY LEFT BLANK

Software Licenses, Open Source Components, and Open Architectures

Presenter: Thomas Alspaugh is adjunct professor of Computer Science at Georgetown University, and visiting researcher at the Institute for Software Research at UC Irvine. He received his PhD in Computer Science from North Carolina State University in 2002. His research interests are in software engineering and focus on informal and narrative models of software at the requirements level. Before completing his PhD, he worked as a software developer, team lead, and manager at several companies—including IBM and Data General—and as a computer scientist at the Naval Research Laboratory on the Software Cost Reduction project, also known as the A-7E project.

Authors:

Hazel Asuncion is a PhD student in the Informatics Department in the Donald Bren School of Information and Computer Sciences at the University of California, Irvine, and also a graduate student researcher at the Institute for Software Research. Her research interests focus on traceability, process workflows, software system architectures, and their interrelationships.

Walt Scacchi is a senior research scientist and research faculty member at the Institute for Software Research, University of California, Irvine. He received a PhD in Information and Computer Science from UC Irvine in 1981. From 1981-1998, he was on the faculty at the University of Southern California. In 1999, he joined the Institute for Software Research at UC Irvine. He has published more than 150 research papers and has directed 45 externally funded research projects. In 2007, he served as General Chair of the 3rd IFIP International Conference on Open Source Systems (OSS2007), Limerick, IE.

Thomas A. Alspaugh, Hazeline U. Asuncion, and Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA
{alspaugh,hasuncion,wcacchi}@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 paper, we identify key properties of OSS licenses, present a license analysis scheme to identify license conflicts arising from composed software elements, and apply it to provide guidance for software architectural design choices whose goal is to enable specific licensed component configurations. Our scheme has been implemented in an operational environment and demonstrates a practical, automated solution to the problem of determining overall rights and obligations for alternative OAs.

1. Introduction

It has been common for OSS projects to require developers to contribute their work under conditions that ensure the project can license its products under a specific OSS license. For example, the Apache Contributor License Agreement grants enough rights to the Apache

Software Foundation for the foundation to license the resulting systems under the Apache License. This sort of license configuration, in which the rights to a system's components are homogenously granted and the system has a well-defined OSS license, was the norm and continues to this day.

However, we more and more commonly see a different license configuration in which the components of a system do not have the same license. The resulting system may not have any recognized OSS license at all—in fact, our research indicates this is the most likely outcome. Instead, if all goes well in its design, there will be enough rights available in the system so that it can be used and distributed—and perhaps modified by others and sublicensed, if the corresponding obligations are met. These obligations are likely to differ for components with different licenses; a BSD (Berkeley Software Distribution)-licensed component must preserve its copyright notices when made part of the system—for example, while the source code for a modified component covered by MPL (the Mozilla Public License) must be made public—and a component with a reciprocal license such as the Free Software Foundation's GPL (General Public License) might carry the obligation to distribute the source code of that component but also of other components that constitute "a whole which is a work based on" the GPL'd component. 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 which case, there may be no rights available for the system as a whole (not even the right of use), because the obligations of the licenses that would permit use of its components cannot simultaneously be met.

The central problem we examine and explain in this paper is to identify principles of software architecture and software licenses that facilitate or inhibit success of the OA strategy when OSS and other software components with open APIs are employed. This is the knowledge we seek to develop and deliver. Without such knowledge, it is unlikely that an OA that is clean, robust, transparent, and extensible can be readily produced. On a broader scale, this paper seeks to explore and answer the following kinds of research questions:

- What license applies to an OA system composed of components with different licenses?
- How do alternative OSS licenses facilitate or inhibit the development of OA systems?
- How should software license constraints be specified to make it possible to automatically determine the overall set of rights and obligations associated with a configured software system architecture?

This paper may help establish a foundation for how to analyze and evaluate dependencies that might arise when seeking to develop software systems that embody an OA when different types of software components or software licenses are being considered for integration into an overall system configuration.

In the remainder of this paper, we examine software licensing constraints. This is followed by an analysis of how these constraints can interact in order to determine the overall license constraints applicable to the configured system architecture. Next, we describe an operational environment that demonstrates automatic determination of license constraints associated with a configured system architecture, and thus offers a solution to the problem we face. We close with a discussion of the conclusions that follow.

2. Background

There is little explicit guidance or reliance on systematic empirical studies for how best to develop, deploy, and sustain complex software systems when different OA and OSS objectives are at hand. Instead, we find narratives that provide ample motivation and belief in the promise and potential of OA and OSS without consideration of what challenges may lie ahead in realizing OA and OSS strategies. Ven (2008) is a recent exception.

We believe that a primary challenge to be addressed is how to determine whether a system, composed of subsystems and components each with specific OSS or proprietary licenses and integrated into the system's planned configuration, is or is not open, and what license constraints apply to the configured system as a whole. This challenge comprises not only evaluating an existing system at run-time but also at design-time and build-time for a proposed system to ensure that the result is "open" under the desired definition and that only the acceptable licenses apply; another important aspect of this challenge is understanding which licenses are acceptable in this context. Because there is a range of types and variants of licenses (OSI, 2008), each of which may affect a system in different ways, and because there are a number of different kinds of OSS-related components and ways of combining them that affect the licensing issue, an essential first step is to understand the kinds of software elements that constitute a software architecture, and what kinds of licenses may encumber these elements or their overall configuration.

OA seems to simply 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 openness of an OA depends 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 copyright (Intellectual Property) licenses assigned to 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 (Alspaugh & Antón, 2007; Scacchi & Alspaugh, 2008). Subsequently, we believe this can lead to situations in which new software development or acquisition requirements stipulate a software system with an OA and OSS, but the resulting software system may or may not embody an OA. This can occur when the architectural design of a system constrains system requirements—raising the question of what requirements can be satisfied by a given system architecture when requirements stipulate specific types or instances of OSS (e.g., Web browsers and content management servers) to be employed (Scacchi, 2002), or what architecture style (Bass, Clements & Kazman, 2003) is implied by a given set of system requirements.

Thus, given the goal of realizing an OA and OSS strategy together with the use of OSS components and open APIs, it is unclear how to best align acquisition, system requirements, software architectures, and OSS elements across different software license regimes to achieve this goal (Scacchi & Alspaugh, 2008).

3. Understanding Open Architectures

The statement that a system is intended to embody an open architecture using open software technologies like OSS and APIs does not clearly indicate what possible mix of software elements may be configured into such a system. To help explain this, we first identify what kinds of software elements are included in common software architectures, whether they are open or closed (Bass et al., 2003).

- *Software source code components*—(a) stand-alone 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 (Feldt, 2007) or "mashups" (Nelson & Churchill, 2006)).
- *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. Executable binaries can be viewed as "derived works" (Rosen, 2005).
- *Application program interfaces/APIs*—The availability of externally visible and accessible APIs to which independently developed components can be connected is the minimum condition required to form an "open system" (Meyers & Obendorf, 2001).
- *Software connectors*—In addition to APIs, these may be software either from libraries, frameworks, or application script code, whose intended purpose is to provide a standard or reusable way of associating programs, data repositories, or remote services through common interfaces. The High Level Architecture (HLA) is an example of a software connector scheme (Kuhl, Weatherly & Damann, 2000), as are CORBA, Microsoft's .NET, Enterprise Java Beans, and LGPL libraries.
- *Configured system or sub-system architectures*—These are software systems that can be built to conform to an explicit architectural design. They include software source code components, executable components, APIs, and connectors that are organized in a way that may conform to a known "architectural style" such as the Representational State Transfer (Fielding & Taylor, 2002) for Web-based client-server applications, or may represent an original or ad hoc architectural pattern (Bass et al., 2003). Each of the software elements—and the pattern in which they are arranged and interlinked—can all be specified, analyzed, and documented using an Architecture Description Language and ADL-based support tools (Bass et al., 2003; Medvidovic, Rosenblum & Taylor, 1999).

Figure 1 provides an overall view of an archetypal software architecture for a configured system that includes and identifies each of the software elements above, as well as including free/open source software (e.g., Gnome Evolution) and closed source software (WordPerfect) components. In simple terms, the configured system consists of software components (grey boxes in the figure) that include a Mozilla Web browser, Gnome Evolution e-mail client, and WordPerfect word processor, all running on a Linux operating system that can access file, print, and other remote-networked servers (e.g., an Apache Web server). These components are interrelated through a set of software connectors (ellipses in the figure) that connect the interfaces of software components (small white boxes attached to a component) and link them together. Modern-day enterprise systems or command-and-control systems will generally have more complex architectures and a more diverse mix of software components than shown in the figure here. As we examine next, even this simple architecture raises a number of OSS licensing issues that constrain the extent of openness that may be realized in a configured OA.

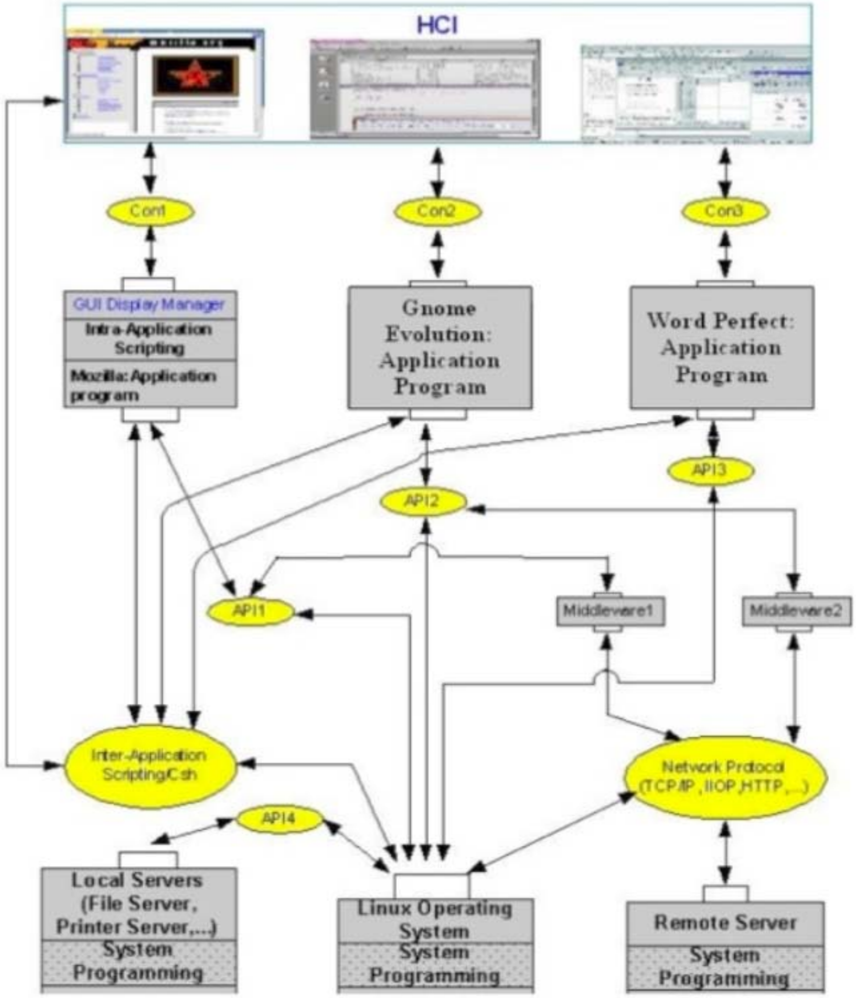

Figure 1. An Archetypal Software Architecture Depicting Components (grey boxes), Connectors (ellipses), Interfaces (small boxes on components), and Data/Control Links

4. Understanding Open Software Licenses

A particularly knotty challenge is the problem of licenses in OSS and OA. There are a number of different OSS licenses, and their number continues to grow. Each license stipulates different constraints attached to software components that bear it. External references are available which describe and explain many different licenses that are now in use with OSS (Fontana et al., 2008; OSI, 2008; Rosen, 2005; St. Laurent, 2004).

More and more software systems are designed, built, released, and distributed as OAs composed of components from different sources, some proprietary and others not. Systems include components that are 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. Software components in such systems evolve not only by ongoing maintenance but also by architectural refactoring, alternative component interconnections, and component replacement (via maintenance patches, installation of new versions, or migration to new technologies). Software components in such systems may be subject to different software licenses, and later versions of a component may be subject to different licenses (e.g., from CDDL—Sun's Common Development and Distribution License—to GPL, or from GPLv2 to GPLv3).

Software systems with open architectures are subject to different software licenses than may be common with traditional, proprietary, closed source systems from a single vendor. Software architects/developers must increasingly attend to how they design, develop, and deploy software systems that may be subject to multiple and possibly conflicting software licenses. We see architects, developers, software acquisition managers, and others concerned with OAs as falling into three groups. The first group pays little or no heed to license conflicts and obligations; they simply focus on the other goals of the system. Those in the second group have assets and resources, and, in order to protect these, they may have an army of lawyers to advise them on license issues and other potential vulnerabilities; or they may constrain the design of their systems so that only a small number of software licenses (possibly just one) are involved—excluding components with other licenses independent of whether such components represent a more effective or more efficient solution. The third group falls between these two extremes; members of this group want to design, develop, and distribute the best systems possible, while they respect the constraints associated with different software component licenses. Their goal is a configured OA system that meets all its goals and for which all the license obligations for the needed copyrights are satisfied. It is this third group that needs the guidance the present work seeks to provide.

There has been an explosion in the number, type, and variants of software licenses, especially with open source software (OSI, 2008). Software components are now available subject to licenses such as the General Public License (GPL), Mozilla Public License (MPL), Apache Public License, (APL), Academic licenses (e.g., BSD, MIT), Creative Commons, Artistic, and others as well as Public Domain (either via explicit declaration or by expiration of prior copyright license). Furthermore, licenses such as these can evolve, resulting in new license versions over time. But no matter their diversity, software licenses represent a legally enforceable contract that is recognized by government agencies, corporate enterprises, individuals, and judicial courts, and, as a result, they cannot be taken trivially. As a consequence, software licenses constrain open architectures and thus architectural design decisions.

So how might we support the diverse needs of different software developers with respect to their need to design, develop, and deploy configured software systems with different, possibly conflicting licenses for the software components they employ? Is it possible to provide automated means for helping software developers determine what constraints will result at design-time, build-time, or run-time when their configured system architectures employ diverse licensed components? These are the kind of questions we address in this paper.

4.1. Software Licenses: Rights and Obligations

Copyright, the common basis for software licenses, gives the original author of a work certain exclusive rights, which for software include the right to use, copy, modify, merge, publish, distribute, sub-license, and sell copies. These rights may be licensed to others, including individuals or groups, and they may be licensed either exclusively so that no one else can exercise them or (more commonly) non-exclusively. After a period of years, the rights enter the public domain, but, until then, the only way for anyone other than the author to have access to the copyright is to license it.

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 buy a legal copy to use and not distribute copies (proprietary licenses), the obligation to preserve copyright and license notices (academic licenses), the obligation to publish at no cost source code that has been modified (MPL), or the reciprocal obligation to publish all source code included at build-time or statically linked (GPL).

Licenses may provide for the creation of derivative works (e.g., a transformation or adaptation of existing software) or collective works (e.g., a Linux distribution that combines software from many independent sources) from the original work by granting those rights, possibly with corresponding obligations.

In addition, the author of an original work can make it available under more than one license, enabling the work's distribution to different audiences with different needs. For example, one licensee might be happy to pay a license fee in order to be able to distribute the work as part of a proprietary product whose source code is not published, while another might need to license the work under MPL rather than GPL in order to have consistent licensing across a system. The result is the distribution of software under any one of several licenses, with the licensee choosing from two ("dual license") or three (Mozilla's "tri-license") licenses.

The basic relationship between software license rights and obligations can be summarized as follows: if you meet the specified obligations, then you get the specified rights. In other words, for the academic licenses, if you retain the copyright notice, list of license conditions, and disclaimer, then you have the right to use, modify, merge, sub-license, etc. For MPL, if you publish modified source code and sub-licensed derived works under MPL, then you get all the MPL rights. These same relationships apply for other types of licenses. However, one thing we have learned from our efforts to carefully analyze and lay out the obligations and rights pertaining to each license is that license details are difficult to comprehend and track—it is easy to get confused or make mistakes. Some of the OSS licenses were written by developers, and often these turn out to be incomplete and legally ambiguous; others, usually more recent, were written by lawyers and 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 (Fontana et al., 2008; Rosen, 2005). Therefore, one of our goals is to make it possible to architect software systems of heterogeneously licensed components without necessarily consulting legal counsel. Similarly, such a goal is best realized with automated support that can help architects understand design choices across components with different licenses and that can provide support for testing build-time releases and run-time distributions to make sure they achieve the specified rights by satisfying the corresponding obligations.

4.2. Expressing Software Licenses

Historically, most software systems, including OSS systems, were entirely under a single software license. However, we now see more and more software systems being proposed, built, or distributed with components that are under various licenses. Such systems may no longer be covered by a single license, unless such a licensing constraint is stipulated at design-time and enforced at build-time and run-time. But when components with different licenses are to be included at build-time, their respective licenses might either be consistent or conflict. Further, if designed systems include components with conflicting licenses, then one or more of the conflicting components must be excluded in the build-time release or must be abstracted behind an open API or middleware, with users required to download and install to enable the intended operation. (This is common in Linux distributions subject to GPL, where, for example, users may choose to acquire and install proprietary run-time components, like proprietary media players.) As a result, a component license conflict need not be a show-stopper if identified at design time. However, developers have to be able to determine which components' licenses conflict and take appropriate steps at design-time, build-time, and run-time that are consistent with the different concerns and requirements that apply at each phase (Scacchi & Alspaugh, 2008).

In order to fulfill our goals, we need 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 components' licenses. We considered relatively complex structures—such as Hohfeld's eight fundamental jural relations (Hohfeld, 1913)—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; following Hohfeld, the lack of a right (which would be "may not") correlates with a duty not to exercise the right ("must not"), and, whenever lack of a right seemed significant in a license, we expressed it as a negative obligation with "must not." 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 in order to minimize the set of actions. A license then may be expressed as a set of rights, with each right associated (in that license) with zero or more obligations that must be fulfilled in order to enjoy that right. Figure 2 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.

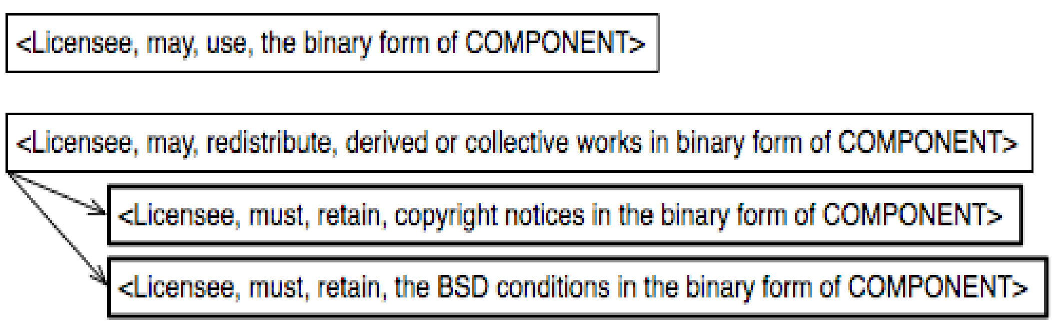

Figure 2. A Portion of the BSD License Tuples

We now turn to examine how OA software systems that include components with different licenses can be designed and analyzed while effectively tracking their rights and obligations.

When designing an OA software system, there are heuristics one can employ to enable architectural design choices that might otherwise be excluded due to license conflicts. First, it is possible to employ a "license firewall," which 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 becomes too cumbersome. Thus, automated support needs to be provided to help overcome and manage the multi-component, multi-license complexity.

5. Automating Analysis of Software License Rights and Obligations

We find that if we start from a formal specification of a software system's architecture, then 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. Accordingly, we employ an architectural description language specified in xADL (2005) to describe OAs that can be designed and analyzed with a software architecture design environment (Medvidovic et al., 1999) such as ArchStudio4 (2006). We have taken this environment and extended it with a Software Architecture License Traceability Analysis module (Asuncion, 2008). This allows for the specification of licenses as a list of attributes (license tuples) using a form-based user interface, similar to those already used and known for ArchStudio4 and xADL (ArchStudio, 2006; Medvidovic et al., 1999).

Figure 3 shows a screenshot of an ArchStudio4 session in which we have modeled the OA seen in Figure 1. OA software components, each of which has an associated license, are indicated by darker-shaded boxes. Light-shaded boxes indicate connectors. Architectural connectors may or may not have associated license information; those with licenses (such as architectural connectors that represent functional code) are treated as components during license traceability analysis. A directed line segment indicates a link. Links connect interfaces between the components and connectors. Furthermore, the Mozilla component, as shown here, contains a hypothetical subarchitecture for modeling the role of intra-application scripting—as might be useful in specifying license constraints for Rich Internet Applications. This subarchitecture is specified in the same manner as the overall system architecture and is visible in Figure 5. The automated environment allows for tracing and analysis of license attributes and conflicts.

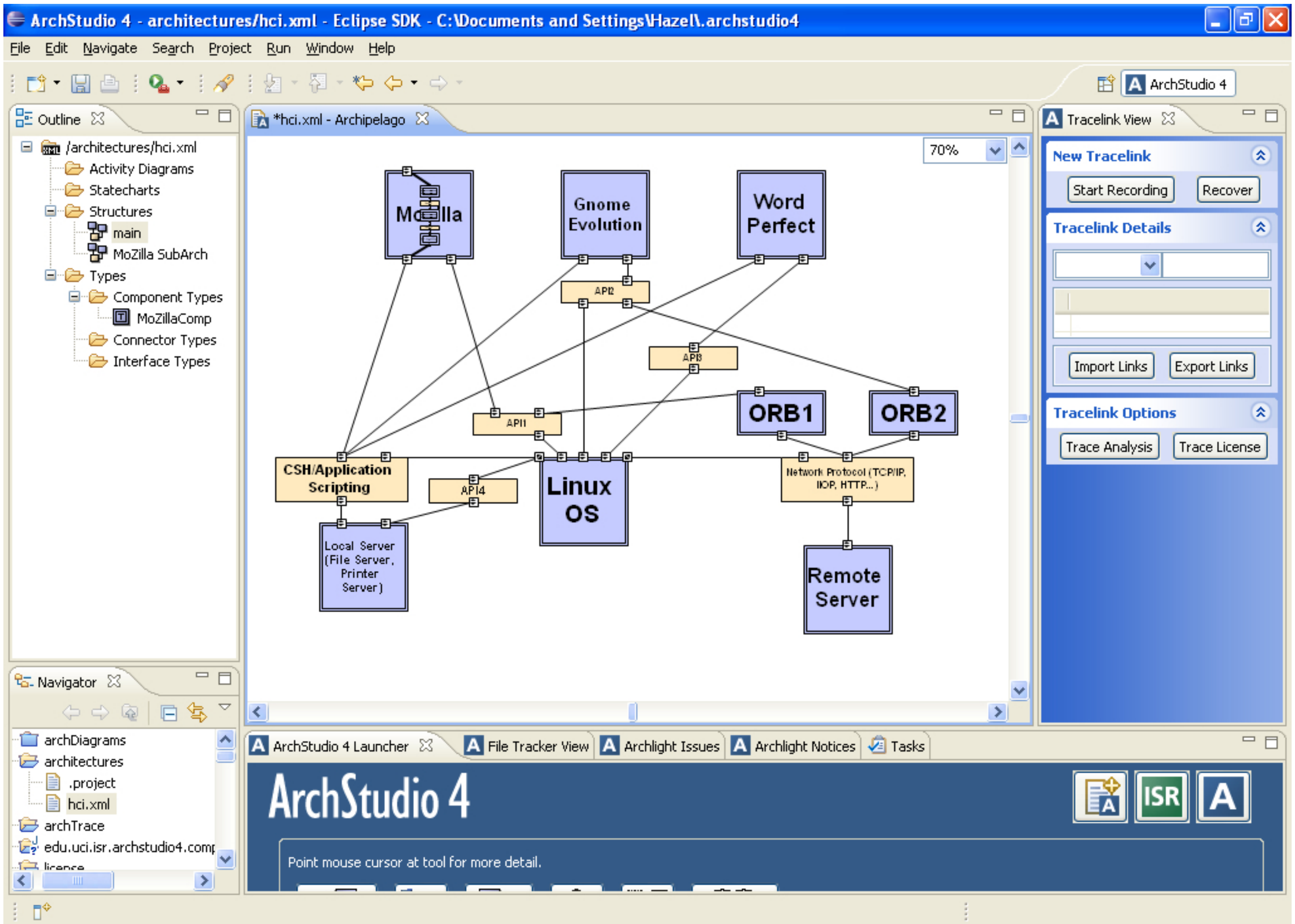

Figure 3. An ArchStudio 4 Model of the Open Software Architecture of Figure 1

Figure 4 shows a view of the internal XML representation of a software license. Analysis and calculations of rights, obligations, and conflicts for the OA are done in this form. This schematic representation is similar in spirit to that used for specifying and analyzing privacy and security regulations associated with certain software systems (Breaux & Anton, 2008).

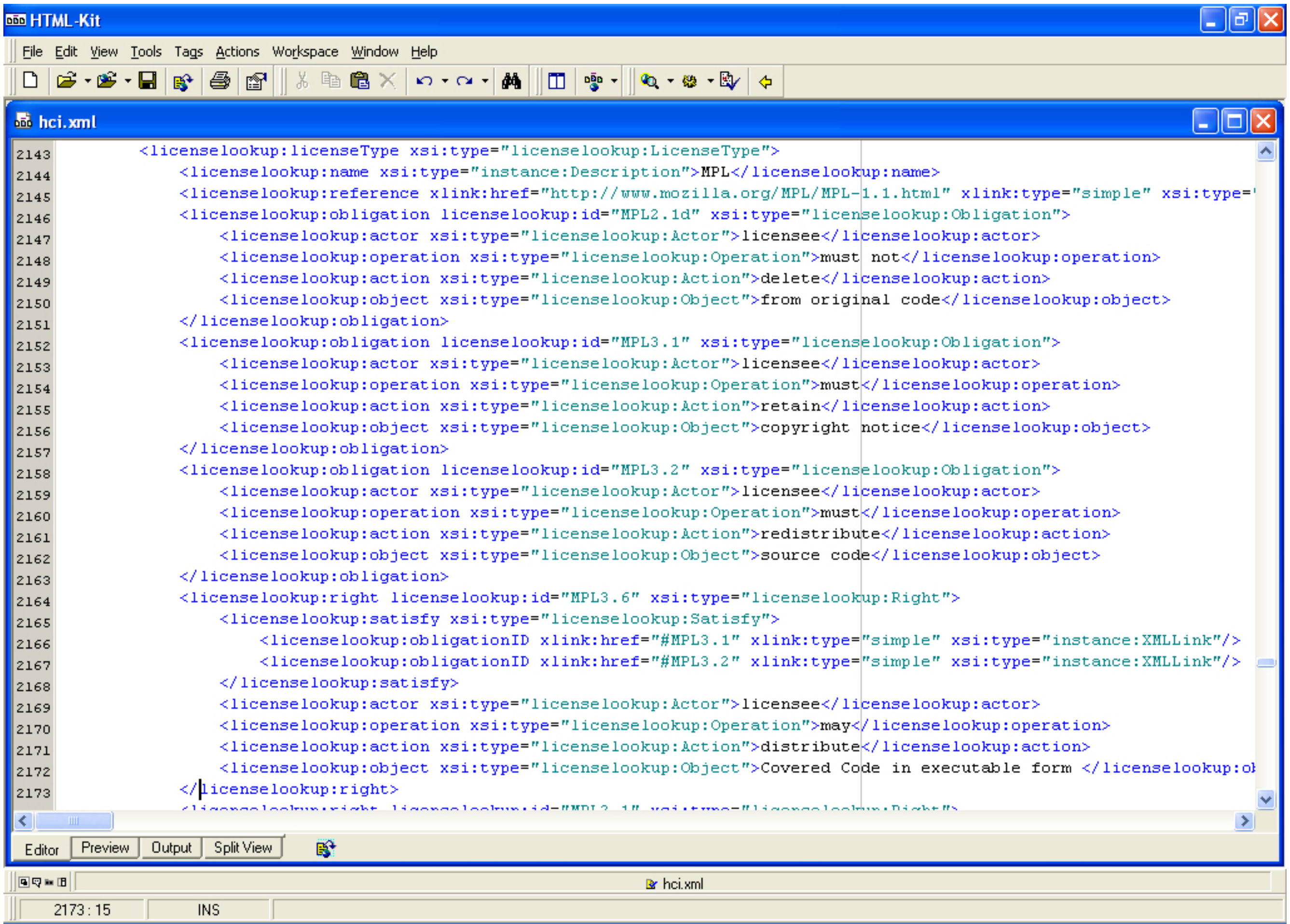

Figure 4. A View of the Internal Schematic Representation of the Mozilla Public License

With this basis to build on, it is now possible to analyze the alignment of rights and obligations for the overall system:

- **Propagation of reciprocal obligations**

Reciprocal obligations are imposed by the license of a GPL'd component on any other component that is part of the same "work based on the Program" (i.e., on the first component), as defined in GPL. 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 contrary to it, or with the set of available rights, by requiring a copyright right that has not been granted. For instance, the Corel proprietary license for the WordPerfect component, CTL (Corel Transactional License), may be taken to entail that a licensee must not redistribute source code. However, an OSS license, GPL, may state that a licensee must redistribute source code. Thus, the conflict appears in the modality of the two otherwise identical obligations, “must not” in CTL and “must” in GPL. A conflict on the same point could also occur between GPL and a component whose license fails to grant the right to distribute its source code.

This phase of the analysis is affected by the overall set of rights that are required. If conflicts arise involving the union of all obligations in all components’ licenses, it may be possible to eliminate some conflicts by selecting a smaller set of rights—in which case, only the obligations for those rights need be considered.

Figure 5 shows a screenshot in which the License Traceability Analysis module has identified obligation conflicts between the licenses of two pairs of components (“WordPerfect” and “Linux OS,” and “GUIDisplayManager” and “GUIScriptInterpreter”).

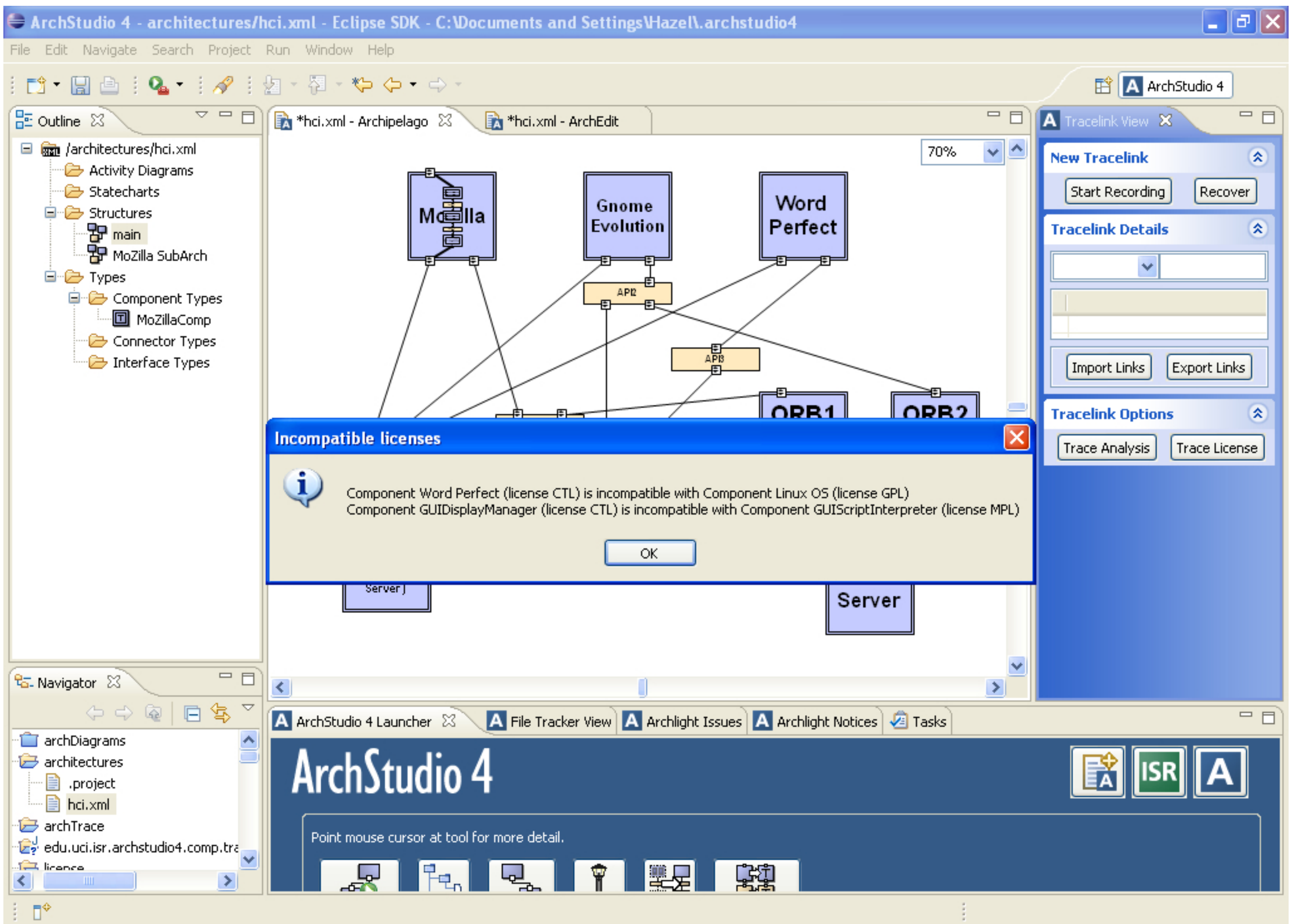

Figure 5. License Conflicts Identified between Two Pairs of Components

- **Rights and obligations calculations**

The rights available for the entire system (use, copy, modify, etc.) then are calculated as the intersection of the sets of rights available for each component of the system.

The obligations required for the whole system then are the union of the specific obligations for each component that are associated with those rights. Examples of specific obligations are “Licensee must retain copyright notices in the binary form of module.c” or “Licensee must publish the source code of component.java version 1.2.3.”

Figure 6 shows a report of the calculations for the hypothetical subarchitecture of the Mozilla component in our archetypal architecture—exhibiting an obligation conflict and the single copyright right (to run the system) that the prototype tool shows would be available for the subarchitecture as a whole if the conflict is resolved; a production tool would also list the rights (none) currently available.

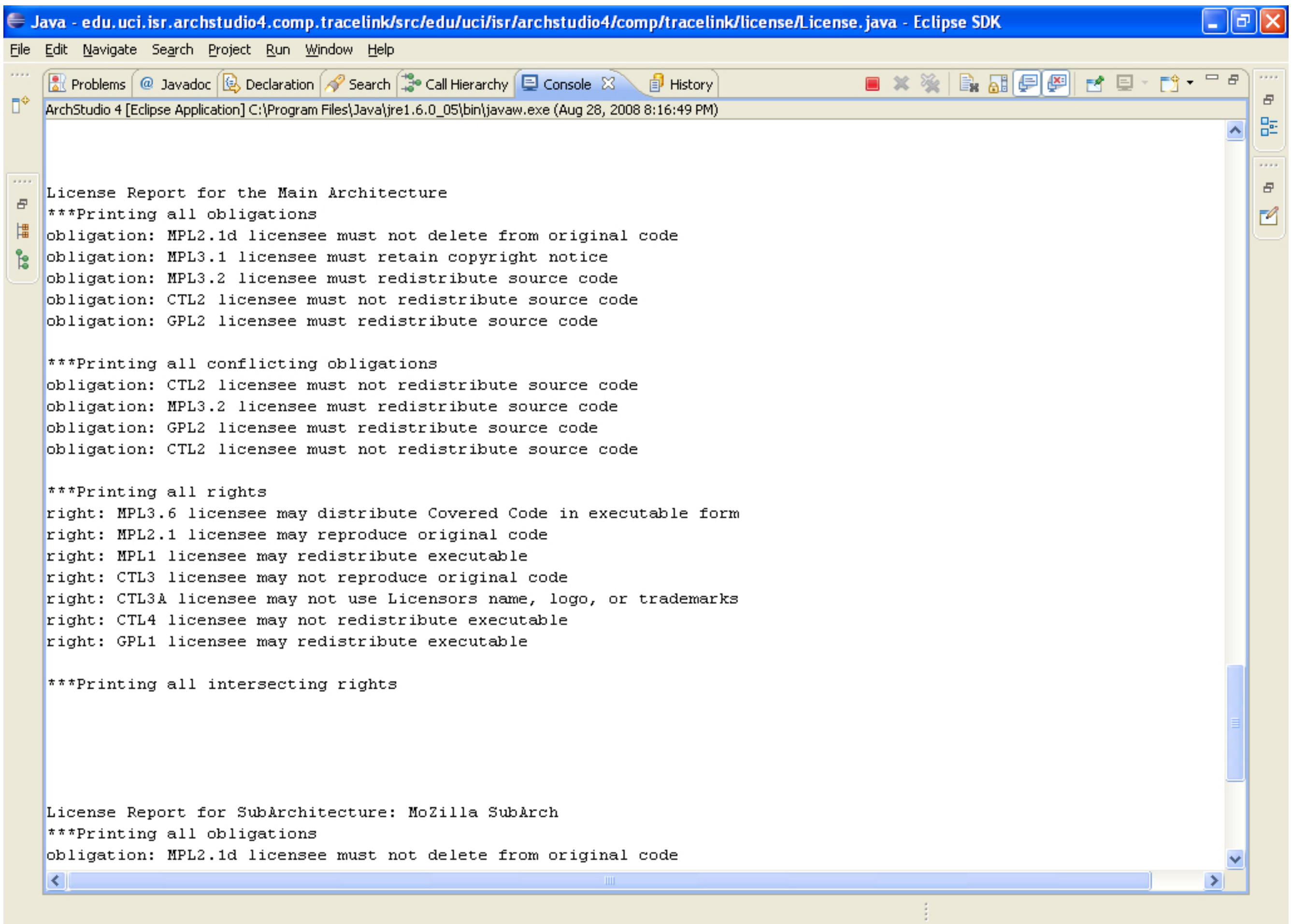

Figure 6. A Report Identifying the Obligations, Conflicts, and Rights for the Architectural Model

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—employing one or more connectors along the paths between the components that act as a license firewall, thereby mitigating or neutralizing the component-component license conflict. 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.

At build-time (and later at run-time), many of the obligations can be tested and verified, for example, that the binaries contain the appropriate notices for their licenses and that the source files are present in the correct version on the Web. These tests can be generated from the internal list of obligations and run automatically. If the system's interface were extended to add a control for it, the tests could be run by a deployed system.

The prototype License Traceability Analysis module provides a proof-of-concept for this approach. We encoded the core provisions of four licenses in XML for the tool—GPL, MPL, CTL, and AFL (Academic Free License)—to examine the effectiveness of the license tuple encoding and the calculations based upon it. While it is clear that we could use a more complex and expressive structure for encoding licenses, in encoding the license provisions to date, we found that the tuple representation was more expressive than needed; for example, the actor was always "licensee" and seemed likely to remain so, and we found use for only three operations or modalities. At this writing, the module shows proof of concept for calculating with reciprocal obligations by propagating them to adjacent, statically linked modules; the extension to all paths not blocked by license firewalls is straightforward and is independent of the scheme and calculations described here. Reciprocal obligations are identified in the tool by lookup in a table, and the meaning and scope of reciprocality is hard-coded; this is not ideal, but we considered it acceptable since the legal definition in terms of the reciprocal licenses will not change frequently. We also focused on the design-time analysis and calculation (rather than on build- or run-time), as it involves the widest range of issues—including representations, calculation of rights and obligations, and design guidance derived from them.

Based on our analytical approach, it appears that the questions of what license (if any) covers a specific configured system, and what rights are available for the overall system (and what obligations are needed for them) are difficult to answer without automated license-architecture analysis. This is especially true if the system or sub-system is already in operational run-time form (Kazman & Carrière, 1999). It might make distribution of a composite OA system somewhat problematic if people cannot understand what rights or obligations are associated with it. We offer the following considerations to help make this clear. For example, a Mozilla/Firefox Web browser covered by the MPL (or GPL or LGPL, in accordance with the Mozilla Tri-License) may download and run intra-application script code that is covered by a different license. If this script code is only invoked via dynamic run-time linkage, or via a client-server transaction protocol, then there is no propagation of license rights or obligations. However, if the script code is integrated into the source code of the Web browser as a persistent part of an application (e.g., as a plug-in), then it could be viewed as a configured sub-system that may need to be accessed for license transfer or conflict implications. A different kind of example can be anticipated with application programs (like Web browsers, e-mail clients, and word processors) that employ Rich Internet Applications or mashups entailing the use of content (e.g., textual character fonts or geographic maps) that is subject to copyright protection—if the content is embedded in and bundled with the scripted application sub-system. In such a case, the licenses involved may not be limited to OSS or proprietary software licenses.

In the end, it becomes clear that it is possible to automatically determine what rights or obligations are associated with a given system architecture at design-time and whether it contains any license conflicts that might prevent proper access or use at build-time or run-time, given an approach such as ours.

6. Discussion

Software system configurations in OAs are intended to be adapted to incorporate new innovative software technologies that are not yet available. These system configurations will evolve and be refactored over time at ever-increasing rates (Scacchi, 2007); components will be patched and upgraded (perhaps with new license constraints), and inter-component connections will be rewired or remediated with new connector types. As such, sustaining the openness of a configured software system will become part of ongoing system support, analysis, and validation. This, in turn, may require ADLs to include OSS licensing properties on components, connectors, and overall system configuration, as well as in appropriate analysis tools (Bass et al. 2003; Medvidovic et al., 1999).

Constructing these descriptions is an incremental addition to the development of the architectural design or alternative architectural designs. But it is still time-consuming and may present a somewhat daunting challenge for large, pre-existing systems that were not originally modeled in our environment.

Advances in the identification and extraction of configured software elements at build-time and their restructuring into architectural descriptions is becoming an evermore automatable endeavor (Choi & Scacchi, 1990; Kazman & Carrière, 1999; Jansen, Bosch & Avgeriou, 2008). Further advances in such efforts have the potential to automatically produce architectural descriptions that can either be manually or semi-automatically annotated with their license constraints, and thus enable automated construction and assessment of build-time software system architectures.

The list of recognized OSS licenses is long and ever-growing, and, as existing licenses are tested in the courts, we can expect their interpretations to be clarified and perhaps altered; the GPL definition of "work based on the Program," for example, may eventually be clarified in this way, possibly refining the scope of reciprocal obligations. Our expressions of license rights and obligations are for the most part compared for identical actors, actions, and objects, then by looking for "must not" in one and either "must" or "may" in the other, so that new licenses may be added by keeping equivalent rights or obligations expressed equivalently. Reciprocal obligations, however, are handled specially by hard-coded algorithms to traverse the scope of that obligation so that addition of obligations with different scope, or the revision of the understanding of the scope of an existing obligation, requires development work. Possibly these issues will be clarified as we add more licenses to the tool and experiment with their application in OA contexts.

Lastly, our scheme for specifying software licenses offers the potential for the creation of shared repositories where these licenses can be accessed, studied, compared, modified, and redistributed.

7. Conclusion

The relationship between open architecture, open source software, and multiple software licenses is poorly understood. OSS is often viewed as primarily a source for low-cost/free software systems or software components. Thus, given the goal of realizing an OA strategy together with the use of OSS components and open APIs, it has been unclear how to best align software architecture, OSS, and software license regimes to achieve this goal. Subsequently, the central problem we examined in this paper was to identify principles of software architecture and software copyright licenses that facilitate or inhibit how best to ensure the success of an OA strategy when OSS and open APIs are required or otherwise employed. In turn, we presented an analysis scheme and operational environment that demonstrates that an automated solution to this problem exists.

We have developed and demonstrated an operational environment that can automatically determine the overall license rights, obligations, and constraints associated with a configured system architecture whose components may have different software licenses. Such an environment requires the annotation of the participating software elements with their corresponding licenses. These annotated software architectural descriptions can be prescriptively analyzed at design-time, as we have shown, or descriptively analyzed at build-time or run-time. Such a solution offers the potential for practical support in design-time, build-time, and run-time license conformance checking and the evermore complex problem of developing large software systems from configurations of software elements that can evolve over time.

Acknowledgements

The research described in this report has been supported by grants #0534771 and # 0808783 from the US National Science Foundation and the Acquisition Research Program at the Naval Postgraduate School. No endorsement implied.

List of References

Alspaugh, T.A., & Antón, A.I. (2007). Scenario support for effective requirements. *Information and Software Technology*, *50*(3), 198-220.

ArchStudio. (2006). *ArchStudio 4 software and systems architecture development environment.* Retrieved from Institute for Software Research, University of California, Irvine website,

http://www.isr.uci.edu/projects/archstudio/

Asuncion, H. (2008). Towards practical software traceability. In *Companion of the 30th International Conference on Software Engineering* (pp. 1023-1026). Leipzig, Germany: ICSE.

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice* (2nd ed.). New York: Addison-Wesley Professional.

Breaux, T.D., & Anton, A.I. (2008). Analyzing regulatory rules for privacy and security requirements. *IEEE Transactions on Software Engineering*, *34*(1), 5-20.

Choi, S., & Scacchi, W. (1990). Extracting and restructuring the design of large systems. *IEEE Software*, *7*(1), 66-71.

Feldt, K. (2007). *Programming Firefox: Building rich internet applications with XUL.* Sebastopol, CA: O'Reilly Press.

Fontana, R., Kuhn, B.M., Molgen, E., Norwood, M., Ravicher, D.B., Sandler, K., et al. (2008). *A legal issues primer for open source and free software projects*. Software Freedom Law Center. (Vers. 1.5.1). Retrieved from http://www.softwarefreedom.org/resources/2008/foss-primer.pdf

Fielding, R., & Taylor, R.N. (2002). Principled design of the modern web architecture. *ACM Transactions Internet Technology*, *2*(2), 115-150.

Hohfeld, W.N. (1913). Some fundamental legal conceptions as applied in judicial reasoning. *Yale Law Journal*, *23*(1), 16-59.

Jansen, A., Bosch, J., & Avgeriou, P. (2008). Documenting after the fact: Recovering architectural design decisions. *Journal of Systems and Software*, *81*(4), 536-557.

Kazman, R., & Carrière, J. (1999). Playing detective: Reconstructing software architecture from available evidence. *Journal of Automated Software Engineering*, *6*(2), 107-138.

Kuhl, F., Weatherly, R., & Dahmann, J. (2000). *Creating computer simulation systems: An introduction to the high level architecture*. Upper Saddle River, NJ: Prentice-Hall PTR..

Medvidovic, N., Rosenblum, D.S., & Taylor, R.N. (1999). A language and environment for architecture-based software development and evolution. In *Proceedings of the 21st International Conference on Software Engineering (ICSE '99*) (pp. 44-53). Los Angeles, CA: IEEE Computer Society..

Meyers, B.C., & Obendorf, P. (2001). *Managing software acquisition: Open systems and COTS products*. New York: Addison-Wesley..

Nelson, L., & Churchill, E.F. (2006, September). Repurposing: Techniques for reuse and integration of interactive services. In *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration*. Waikoloa, HI: IEEE.

Open Source Initiative (OSI). (2008). *The open source initiative*. Retrieved from http://www.opensource.org/

Rosen, L. (2005). *Open source licensing: Software freedom and intellectual property law*. Upper Saddle River, NJ: Prentice-Hall PTR. Retrieved from http://www.rosenlaw.com/oslbook.htm

Scacchi, W. (2002, February). Understanding the requirements for developing open source software systems. *IEE Proceedings—Software*, *149*(1), 24-39.

Scacchi, W. (2007). Free/open source software development: Recent research results and emerging opportunities. In *Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering* (pp. 459-468). Dubrovnik, Croatia: ESEC/FSE.

Scacchi, W., & Alspaugh, T.A. (2008). *Emerging issues in the acquisition of open source software within the U.S. Department of Defense*. In *Proceedings of the 5th Annual Acquisition Research Symposium* (pp. 230-244). Monterey, CA: Naval Postgraduate School.

St. Laurent, A.M. (2004). *Understanding open source and free software licensing*. Sebastopol, CA: O'Reilly Press.

Ven, K., & Mannaert, H. (2008). Challenges and strategies in the use of open source software by independent software vendors. *Information and Software Technology*, *50*, 991-1002.

Wheeler, D.A. (2007, June). Open source software (OSS) in U.S. government acquisitions. *The DoD Software Tech News*, *10*(2), 7-13.

xADL. (2005). *xADL 2.0: Highly-extensible architecture description language for software and systems*. Retrieved from Institute for Software Research, University of California, Irvine website, http://www.isr.uci.edu/projects/xarchuci/

THIS PAGE INTENTIONALLY LEFT BLANK

2003 - 2009 Sponsored Research Topics

Acquisition Management

- Acquiring Combat Capability via Public-Private Partnerships (PPPs)
- BCA: Contractor vs. Organic Growth
- Defense Industry Consolidation
- EU-US Defense Industrial Relationships
- Knowledge Value Added (KVA) + Real Options (RO) Applied to Shipyard Planning Processes
- Managing Services Supply Chain
- MOSA Contracting Implications
- Portfolio Optimization via KVA + RO
- Private Military Sector
- Software Requirements for OA
- Spiral Development
- Strategy for Defense Acquisition Research
- The Software, Hardware Asset Reuse Enterprise (SHARE) repository

Contract Management

- Commodity Sourcing Strategies
- Contracting Government Procurement Functions
- Contractors in 21st Century Combat Zone
- Joint Contingency Contracting
- Model for Optimizing Contingency Contracting Planning and Execution
- Navy Contract Writing Guide
- Past Performance in Source Selection
- Strategic Contingency Contracting
- Transforming DoD Contract Closeout
- USAF Energy Savings Performance Contracts
- USAF IT Commodity Council
- USMC Contingency Contracting

Financial Management

- Acquisitions via leasing: MPS case
- Budget Scoring
- Budgeting for Capabilities-based Planning
- Capital Budgeting for DoD

- Energy Saving Contracts/DoD Mobile Assets
- Financing DoD Budget via PPPs
- Lessons from Private Sector Capital Budgeting for DoD Acquisition Budgeting Reform
- PPPs and Government Financing
- ROI of Information Warfare Systems
- Special Termination Liability in MDAPs
- Strategic Sourcing
- Transaction Cost Economics (TCE) to Improve Cost Estimates

Human Resources

- Indefinite Reenlistment
- Individual Augmentation
- Learning Management Systems
- Moral Conduct Waivers and First-tem Attrition
- Retention
- The Navy's Selective Reenlistment Bonus (SRB) Management System
- Tuition Assistance

Logistics Management

- Analysis of LAV Depot Maintenance
- Army LOG MOD
- ASDS Product Support Analysis
- Cold-chain Logistics
- Contractors Supporting Military Operations
- Diffusion/Variability on Vendor Performance Evaluation
- Evolutionary Acquisition
- Lean Six Sigma to Reduce Costs and Improve Readiness
- Naval Aviation Maintenance and Process Improvement (2)
- Optimizing CIWS Lifecycle Support (LCS)
- Outsourcing the Pearl Harbor MK-48 Intermediate Maintenance Activity
- Pallet Management System
- PBL (4)
- Privatization-NOSL/NAWCI
- RFID (6)
- Risk Analysis for Performance-based Logistics
- R-TOC Aegis Microwave Power Tubes

- Sense-and-Respond Logistics Network
- Strategic Sourcing

Program Management

- Building Collaborative Capacity
- Business Process Reengineering (BPR) for LCS Mission Module Acquisition
- Collaborative IT Tools Leveraging Competence
- Contractor vs. Organic Support
- Knowledge, Responsibilities and Decision Rights in MDAPs
- KVA Applied to Aegis and SSDS
- Managing the Service Supply Chain
- Measuring Uncertainty in Earned Value
- Organizational Modeling and Simulation
- Public-Private Partnership
- Terminating Your Own Program
- Utilizing Collaborative and Three-dimensional Imaging Technology

A complete listing and electronic copies of published research are available on our website: www.acquisitionresearch.org

THIS PAGE INTENTIONALLY LEFT BLANK

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL
555 DYER ROAD, INGERSOLL HALL
MONTEREY, CALIFORNIA 93943

www.acquisitionresearch.org