Paper for the 5th Annual Acquisition Research Symposium, 14-15 May 2008

Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense

Walt Scacchi and Thomas A. Alspaugh

Institute for Software Research

University of California, Irvine

Irvine, CA 92697-3455 USA

{wscacchi, alspaugh}@ics.uci.edu

1-949-824-4130, 1-949-824-1715 (fax)

April 2008

Abstract

In the past five or so years, it has become clear that the U.S. Air Force, Army, and Navy have all committed to a strategy of acquiring software-intensive systems that require or utilize an "open architecture" (OA) and "open technology" (OT) which may incorporate OSS technology or OSS development processes. There are many perceived benefits and anticipated cost savings associated with an OA strategy. However, the challenge for acquisition program managers is how to realize the savings and benefits through requirements that can be brought into system development practice. As such, the central problem we examine in this paper is to identify principles of software architecture and OSS copyright licenses that facilitate or inhibit the success of an OA strategy when OSS and open APIs are required or otherwise employed. By examining and analyzing this problem we can begin to identify what additional requirements may be needed to fulfill an OA strategy during program acquisition.

Biographies

Walt Scacchi is a senior research scientist and research faculty member at the Institute for Software Research, University of California, Irvine. He received a Ph.D. 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 Alspaugh is an Assistant Professor of Informatics in the Donald Bren School of Information and Computer Sciences, University of California, Irvine. He received his Ph.D. 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 Ph.D., 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.

Introduction

Interest within the U.S. Department of Defense (DoD) and military services in free and open source software (OSS) first appeared in the past five or so years [cf. Bollinger 2003]. More recently, it has become clear that the U.S. Air Force, Army, and Navy have all committed to a strategy of acquiring software-intensive systems across the board that require or utilize an "open architecture" (OA) and "open technology" (OT) which may incorporate OSS technology or OSS development processes [Herz and Scott 2007]. Why?

According to Riechers [2007], the Air Force sees that with its software-intensive systems, there is increasing complexity of the software (code) itself, they may be "held hostage" to proprietary legacy components, they seek more timely delivery of new solutions, and that acquisitions and requirements take too long. So the Air Force is moving towards an OT development approach that embraces open standards, open data, open program interfaces, best-of-breed OSS, and OSS development practices.

According to Brig. Gen. Justice [2007a, 2007b], the Army seeks to move away from closed source software, expensive software upgrades, vendor lock-in, and broadly exploited security weaknesses. Subsequently, the Army seeks to adopt OSS because it may realize direct cost savings (compared to proprietary closed source software), gain access to source code to better develop domain and IT expertise, enable the transition to Web 2.0 technologies, and to enable rapid injection of innovative concepts from diverse R&D/IT communities into systems for tactical command and control (C3T), future combat systems, enterprise information systems, and others [Starett 2007].

Last, according to Guertin [2007], the Navy seeks to mitigate the spiraling costs of weapon systems through adoption of OA [Navy 2006], as well as the adoption of open business models for the acquisition and spiral development of new systems. This may therefore necessitate better alignment of the system requirements and program acquisition communities, as well as to better alignment of industry and academic partners who engage in software-focused research and development activities with DoD support.

The central problem we examine and explain in this paper is to identify principles of software architecture and OSS copyright licenses that facilitate or inhibit how best to insure the success of the OA strategy when OSS and open APIs are required or otherwise employed. This is the knowledge we seek to develop and deliver. Without such knowledge, program acquisition managers and Program Executive Offices are unlikely to acquire software-intensive systems that will result in an OA that is clean, robust, transparent and extensible. This may frustrate the ability of program managers or program offices to realize faster, better, and cheaper software system acquisition, development, and post-deployment support.

On a broader scale, this paper seeks to explore and answer the following kinds of research questions: How does the use of OSS components and open APIs (a) facilitate, or (b) inhibit the ability to develop and deliver an OA software system? How do the requirements for OA affect system acquisition? How do alternative OSS licenses facilitate or inhibit the development of OA systems? How does the use of OSS components and open APIs manifest requirements that (a) facilitate, or (b) inhibit program acquisition?

Last, this paper may help establish a foundation for how to analyze and evaluate dependencies that might arise when seeking to develop software systems that should embody an OA when different types of OSS components or OSS component licenses are being considered for integration. Finally, we believe there are new ways for determining requirements for how best to develop software systems with OSS [Scacchi 2002] can interact with acquisition processes [Choi and Scacchi 2001] in ways that are not apparent within current public perspectives for OA based on OSS [cf. Guertin 2007, Justice 2007a, 2007b, Riechers 2007].

In the remainder of this paper, we examine what makes achieving OA and OT difficult from a technical and program management/acquisition perspective, with respect to understanding what OA incorporating modern OSS entail from a software architecture standpoint, software licensing regimes, and how/where they interact. We start by providing some additional background on "openness" , and then follow we an description and analysis of open software architecture concepts, and of open source software licenses. This gives rise to a discussion that identifies some new requirements that must be addressed by program managers in acquisitions that are intended to realize an OA software system. We then close with a review of the conclusions that follow.

Background

Across the three military services within the DoD, OA means different things and is seen as the basis for realizing different kinds of outcomes. Thus, it is unclear whether the acquisition of a software system that is required to incorporate an OA as well as utilize OSS technology and development processes [cf. Wheeler 2007] for one military service will realize the same kinds of benefits anticipated for OA-based systems by another service. Somehow, DoD acquisition program managers must make sense or reconcile such differences in expectations and outcomes from OA strategies in each service or across DoD. Yet there is little explicit guidance or reliance on systematic empirical studies for how best to develop, deploy, and sustain complex software-intensive military systems in the different OA and OSS presentations and documents that have so far been disseminated [cf. Weathersby 2007]. Instead, what mostly exists are narratives that serve to provide ample motivation and belief into the promise and potential of OA and OSS without consideration of what socio-technical challenges may lie ahead in realizing OT, OA, and OSS strategies.

In characterizing the challenges facing acquisition of OA and OSS systems, we have found it helpful to compare the new property of "Openness" with the familiar property "Correctness": we summarize this with the maxim "*open* is the new *correct*".

Acquisition officers are familiar with the challenges of acquiring systems that meet the necessary requirements with regard to correct behavior: the correctness of the overall system depends on the correctness of its components and how they are interconnected; correctness is a relative quality, in that a system may meet its behavioral requirements to a greater or lesser degree, but almost by definition a system is never completely correct, and its degree of correctness cannot be definitely established in a finite time; a lack of correctness has an effect when that part of the system is executing; and the correctness of a system in meeting its requirements is determined, by engineers and the system's users, through testing it and using it. Openness is both similar to and different from correctness, however. We argue that the openness of a system depends, like correctness, on the system's components, how they are interconnected, and how they are configured into an overall software system architecture. Unlike for correctness, however, a system may be completely open, or may fail to be open in various ways; and because the software elements that define a system are finite and enumerable, its openness can in principle be determined. Also unlike correctness, a system is either open or not open even when it is not operating, and DoD may pay the consequences of a lack of openness (in the form of license fees) before the system is ever used, or even if it is never used. Finally, unlike for correctness, openness may ultimately by the province of lawyers and policy makers, not of engineers or users.

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 in the system's planned configuration, is or is not open, and what license(s) apply to the configured system as a whole. This challenge comprises not only evaluating an existing system, but planning for a proposed system to ensure that the result is "open" under the desired definition, and that only the acceptable licenses apply; and also understanding which licenses are acceptable in this context. Because there are a range of kinds of licenses, 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, a first necessary step is to understand kinds of software elements that constitute a software architecture, and what kinds of licenses may encumber these elements or their overall configuration.

OA seem to simply suggest 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 OA, since OA depend on: (a) how/why OSS and open APIs are located within the system architecture, (b) how OSS and open APIs are implemented, embedded, or interconnected, (c) whether the 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) many alternative architectural configurations and APIs that may or may not produce an OA [cf. Antón and Alspaugh 2007, Diallo, *et al.*, 2007, Scacchi 2007]. Subsequently, we believe this can lead to situations in which if program acquisition stipulates a software-intensive system with an OA and OSS, then the resulting software system may or may not embody an OA. This can occur when the architectural design of a system constrains system requirements—that is, what requirements can be satisfied by a given system architecture, when requirements stipulate specific types or instances of OSS (e.g., Web browsers, content management servers) to be employed, or what architecture style [Bass, Clements, and Kazman 2003] is implied by given system requirements.

Thus, given the goal of realizing an OA and open technology strategy [cf. Herz and Scott 2007] together with the use of OSS components and open APIs, it is unclear how to best align program acquisition, system requirements, software architectures, and OSS license regimes to achieve this goal.

Understanding open software architecture concepts

A system 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 [cf. Bass, Clements, Kazman 2003].

- *Software source code components* – these include the computer programs that direct the intended computation, calculation, control flow, and data manipulation. These are programs for which the source code is open for access, review, modification, and possible redistribution by their developers. However, there are at least four different forms of computer programs these days.
    - *standalone programs* – these are the computer programs that we have long understood, often as isolated systems or monolithic applications that accept data inputs, manipulate and transform this data, and produce outputs (calculated results, information displays, emit control signals to devices, etc.) under user or system administered control.
    - *libraries, frameworks, or middleware* – these are collections of software functions no one of which is typically a standalone program. Such software is often expected to be routinely reused in many different systems or applications. This software may also be used to provide a layer of abstraction that hides source code implementation details so as to improve subsequent software portability, or to hide alternative software implementations.
    - *inter-application script code* – this software is used to combine independent programs together by associating their respective inputs, outputs, and control variables. This software is sometimes called, "glue code" to suggest its primary use is to connect programs together through the use of "pipes" and/or "filters" which control or modulate the directed flow of information between the associated programs. Such scripts may be as short a a single line of code, but on the other hand, they can be as large as thousands (even hundreds of thousands) source lines of code.
    - *intra-application script code* – this software is similar in spirit to inter-application script code, except the focus is on organizing, controlling, and manipulating input and output data/presentations from remote Web services/repositories for view and end-user interaction at the human-computer interface. Popular Web application systems like the Firefox Web browser may be scripted to provide animated user interfaces coded in languages like Javascript, ActionScript, or PhP to create Rich Internet Applications [Feldt 2007] or "mashups" [Nelson and Churchill 2006]. Such scripts may be as short as a single line of code, but on the other hand, they can be as large as thousands (even tens of thousands) source lines of code. However, custom intra-application software languages may also be designed to create domain-specific languages (e.g., XUL for Firefox Web browser [Feldt 2007]) for rapid construction of persistent/disposable software functions (or macros), which enable increased software development productivity or end-user programming.
- *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 are rarely treated as open since they may also be viewed as "derived works" [Rosen 2005] that result from the compilation or interpretation of software source code which may not be available, or may be proprietary. Executable components are widespread and common in every computing system, even in OSS systems. However, executable components may also only become part of a system during its execution through dynamic (or run-time) linking. Finally, though their binary form makes them available for execution through external linkage to some other program, such form also makes figuring out what they do very difficult, if they have little/no documentation available.

- *Application program interfaces/APIs* – These software interfaces are generally not programs that can be executed, but they enable software system developers to access their functionality without direct access to their source code. The availability of externally visible and accessible APIs to which independently developed components can be connected to is the minimum required to form an "open system" [Meyers and Obendorf 2001]. Oftentimes the APIs are treated as if they enable direct access to the otherwise hidden software, but a closed software system may employ a layer of abstract APIs as "shims" that better align multiple program interfaces or security barriers that seek to protect disclosure of private or proprietary information. Such information may include the details of actual software function interfaces (which may be designated as "trade secrets"), or hidden software functions that may only be known to software developers with secure, restricted code access.
- *Software connectors* – 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. These may include software technologies that constitute a "software bus" for plugging in independent software modules (programs or functions), network protocols that enable and control the flow of data between remote programs across a LAN or Internet, or even a database management system (DBMS) that is used to enable data sharing and storage among programs connected to the DBMS. The High Level Architecture (HLA) is an example of a software connector scheme [Kuhl, Weatherly, Damann 2000], as are CORBA, Microsoft's .NET, and Enterprise Java Beans.
- *Configured system or sub-system* – These are software systems built to conform to an explicit architectural specification. They include software source code/binary 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 and Taylor 2002] for Web-based client-server applications, or may represent an original or ad hoc architectural pattern [Bass, Clements, Kazman 2003]. All of the software elements, and how they are arranged and interlinked, can all be specified, analyzed, and documented using an Architecture Description Language [Bass, Clements, and Kazman 2003] and ADL-based support tools. Beyond this, any or all of the software elements in a configured system or sub-system may be OSS or not. In contrast to a derived work, a configured system or sub-system is considered as a "collective work" and as such is subject to its own copyright and license protection as intellectual property, whether open or closed [Rosen 2005, St. Laurent 2004]. However, such intellectual property declaration cannot employ a license regime on the overall system that supercedes or controverts the license protections/obligations of the individual software elements that

constitute the configured system or sub-system.

Figure 1 provides an overall view of a hypothetical software architecture for a configured system that includes and identifies each of the software elements above, as well as including open source (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 email client, and WordPerfect word processor that run on a Linux operating system that can access file, print, and other remote networked servers (e.g., 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) that are linked 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, this simple architecture raises a number of OSS licensing issues that mitigate the extent of openness that is realized in a configured OA.

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, each with different rights and obligations attached to software components that bear it. External sources are available which describe and explain the many different licenses that are now in use with OSS [OSI 2008, Rosen 2005, St. Laurent 2004]. As such, we will not delve into the details or variations among the many licenses, except to note a few key properties that should be recognized as potentially impacting the openness of a configured software system, and therefore whether it can realize an OA.

The GNU General Public License (GPL), the most widely used OSS license, implements a strong copyleft, requiring that the software source code be distributed and that any modified versions also be licensed under GPL [Rosen 2005, St. Laurent 2004]. The GPL along with some other OSS licenses like the Mozilla Public License (MPL), and others (CPL, OSL [OSI 2008, Rosen 2005]), are identified as "reciprocal" licenses that in some way transfer license obligations to derivative software systems. A software system component or connector based on existing OSS inherits the obligations or restrictions of the originating OSS. In contrast, an academic freedom license such as the BSD, MIT, or Apache license permits derivative software works to be incorporated into a proprietary, closed-source product [Rosen 2005, St. Laurent 2004]. Academic licenses are identified as "unrestrictive" so that software components or connectors derived from OSS covered by an academic freedom license need not adhere to the obligations of the originating OSS.

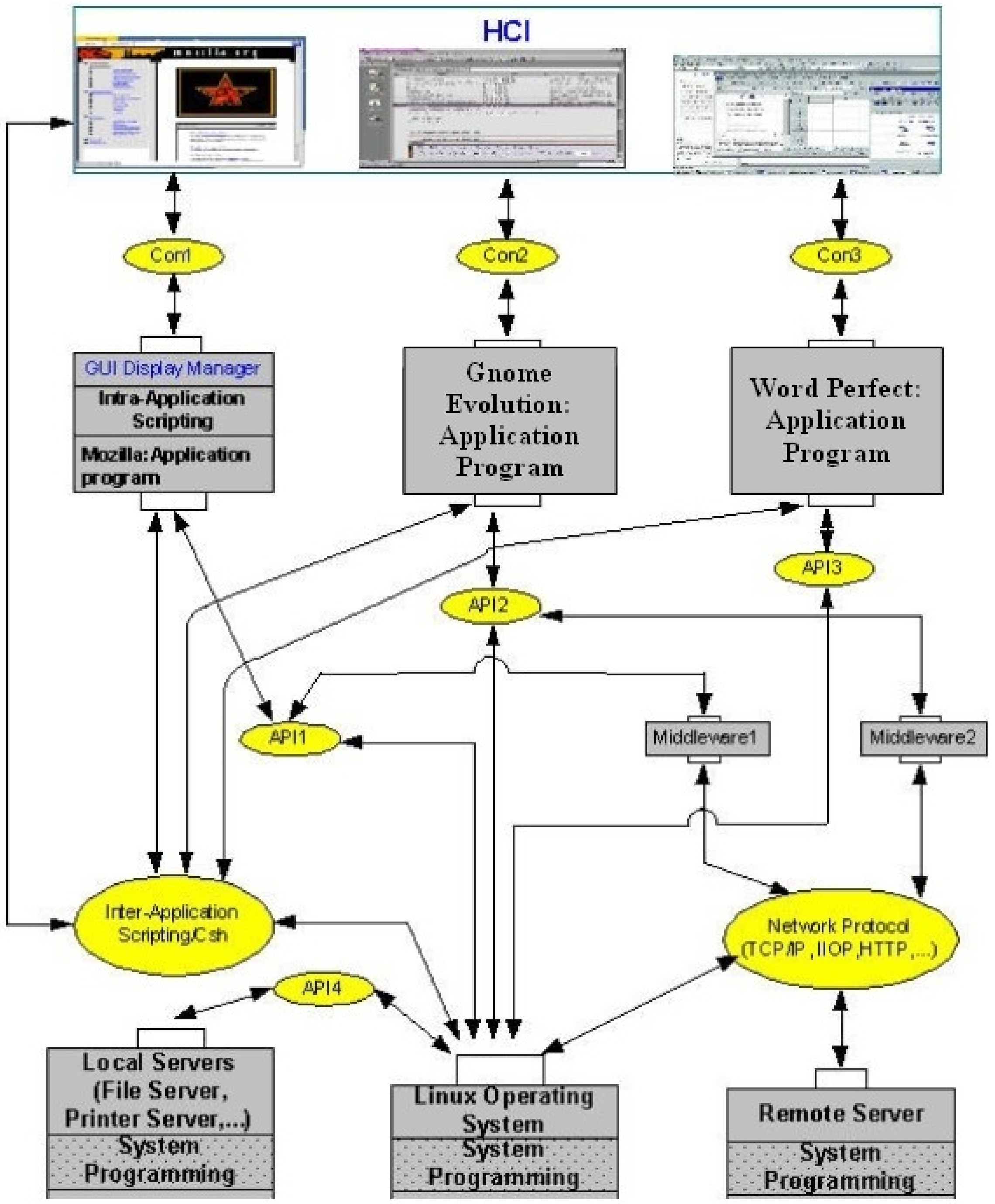
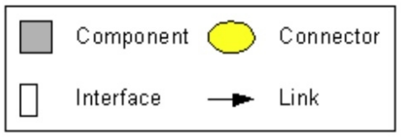

HCI
Con1
Con2
Con3
GUI Display Manager
Intra-Application Scripting
Mozilla: Application program
Gnome Evolution: Application Program
Word Perfect: Application Program
API1
API2
API3
API4
Middleware1
Middleware2
Inter-Application Scripting/Csh
Network Protocol (TCP/IP, IIOP, HTTP,...)
Local Servers (File Server, Printer Server,...)
System Programming
Linux Operating System
System Programming
Remote Server
System Programming

Figure 1. Software components, connectors, interfaces arranged in an overall software system configuration. Components, connectors, and overall system configuration may be subject to different software licenses.

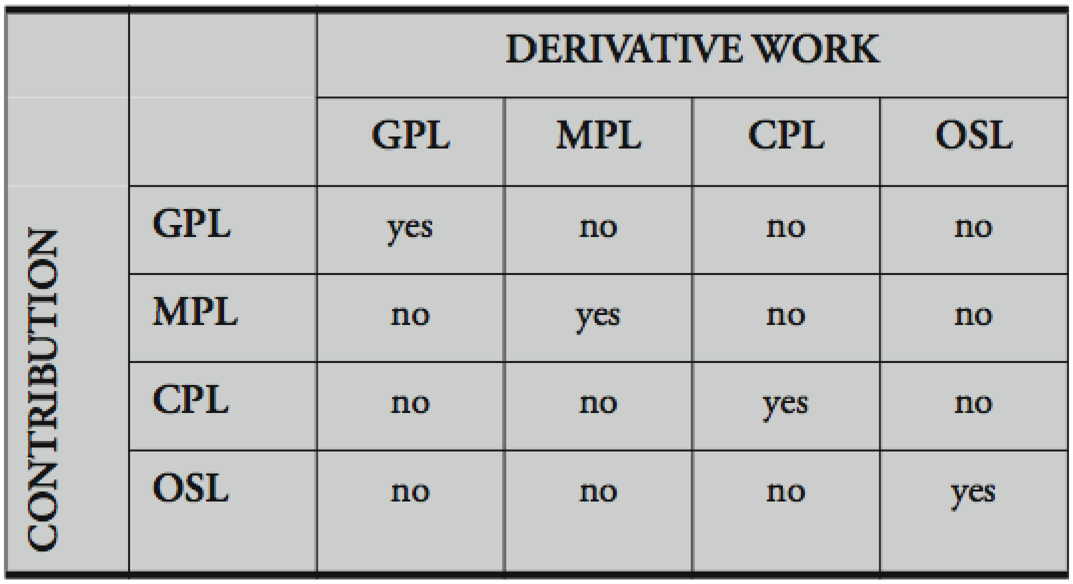


What license applies to an OA system containing some GPL components with a reciprocal license and some BSD components with unrestrictive license, or perhaps even some proprietary software license? In Figure 1, we see at least three software components that have different software licenses: the Mozilla Web browser (subject to the MPL), Gnome Evolution email client (subject to the GPL), and WordPerfect word processor (subject to a proprietary software license). The license problem is further complicated by components designed to operate on license requirements. For example, a software shim may be a library function, abstract interface, or script code designed to serve as a connector between two applications that have different licenses, so that neither application's license is violated and neither application is "infected" by the restrictions or obligations of the other's license. In this regard, a software connector is a configured system (or OA) element specifically designed to modulate the license requirements imposed on the components it connects. In Figure 1, follows the links between the Mozilla Web browser, Gnome Evolution, and WordPerfect. The requirements imposed by a component's license is thus affected by the architectural structure of the system containing it, and vice versa. Figures 2a and 2b provide suggested mappings of license obligations that can constrain a configured software system derived from OSS components and connectors covered by a specific OSS license.

The question of what license covers a specific configured system is difficult to answer, especially if the system or sub-system is already in operation [cf. Kazman and Carrière 1999]. We offer the following considerations to help make this clear. For example, a Mozilla/Firefox Web browser covered by the MPL 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 linking (or invocation), then there is no transfer of license restrictions or obligations. However, if the script code is integrated into the source code of the Web browser as persistent part of an application, then it could be viewed as a configured sub-system that may need to be accessed for license transfer implications. Another different kind of example can be anticipated with application programs (like Web browsers, email clients, and word processors) that employ Rich Internet Applications or mashups that entail 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.

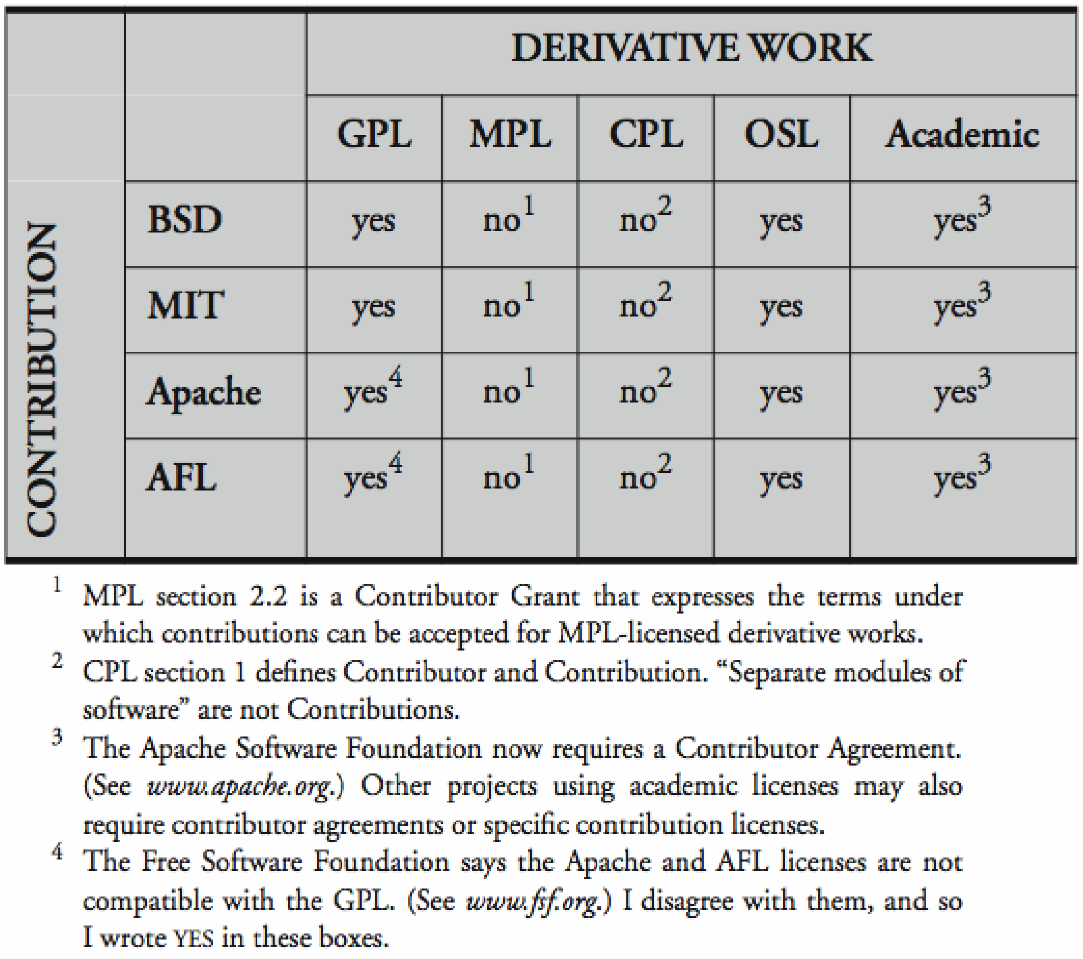
Next, as software system configuration or OA are intended to be adapted to incorporate new innovative software technologies that are not yet at hand, then we recognize that these OSS-based system configurations will evolve over time at ever increasing rates [Scacchi 2007], components will be replaced, 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 [cf. Bass, Clements, and Kazman 2003].

Figure 2a. Mapping *Reciprocal* OSS licenses to derivative works [Rosen 2005]



		DERIVATIVE WORK			
		GPL	MPL	CPL	OSL
CONTRIBUTION	GPL	yes	no	no	no
	MPL	no	yes	no	no
	CPL	no	no	yes	no
	OSL	no	no	no	yes

Figure 2b. Mapping *Unrestrictive* Academic to Reciprocal OSS licenses [Rosen 2005]



		DERIVATIVE WORK				
		GPL	MPL	CPL	OSL	Academic
CONTRIBUTION	BSD	yes	no[1]	no[2]	yes	yes[3]
	MIT	yes	no[1]	no[2]	yes	yes[3]
	Apache	yes[4]	no[1]	no[2]	yes	yes[3]
	AFL	yes[4]	no[1]	no[2]	yes	yes[3]

1 MPL section 2.2 is a Contributor Grant that expresses the terms under which contributions can be accepted for MPL-licensed derivative works.

2 CPL section 1 defines Contributor and Contribution. "Separate modules of software" are not Contributions.

3 The Apache Software Foundation now requires a Contributor Agreement. (See *www.apache.org*.) Other projects using academic licenses may also require contributor agreements or specific contribution licenses.

4 The Free Software Foundation says the Apache and AFL licenses are not compatible with the GPL. (See *www.fsf.org*.) I disagree with them, and so I wrote YES in these boxes.

Original footnotes from [Rosen 2005, p. 251].

Moving forward, analyses of OSS licenses by intellectual property lawyers may suggest a way out of the OSS licensing/relicensing mess at hand. Note, we are not lawyers, so we are not offering any legal advice. Feel free to consult legal counsel if or when appropriate for guidance on license interpretation or enforcement conditions. However, we find some encouraging words are offered for consideration here. Rosen [2005, p. 252] observes OSS license incompatibilities can prevent OSS from being freely used and combined. The multiplicity of such licenses only makes the problem worse (review the tables in Figure 2a and 2b). Copyright law and contract law which cover the interpretation and enforcement of OSS licenses is such that OSS developers or distributors (e.g., Defense contractors) cannot simply relicense copyright protected OSS unless they have permission to do so. This in turn may mitigate some requirements shaping the development and deployment of military software applications that are suppose to embody an OA.

Terms and conditions for reciprocity obligations in licenses like the GPL and others apply to OSS that is modified and redistributed, and not to software that may be modified but not distributed to others outside of the organization. Also, this raises the questions for what constitutes “distribution” or “redistribution” for a government organization that acquires access rights to all software and data developed under contract. Similarly, for government employees whose work is not protected by copyright (and thus may enter into the public domain), then this may pose new opportunities for adhering to or working around OSS license restrictions or obligations.

Finally, as Rosen [2005, p. 253] observes, by merely aggregating (or configuring) software from different sources, and treating such software as black boxes (e.g., no intra-application scripting allowed and/or employ dynamic run-time linkage), it is possible to technically avoid creation of derivative works that inherit the license restrictions or obligations of the software elements involved. Subsequently, Rosen finds that OSS license incompatibilities are inconveniences rather than barriers, and ultimately, one can get around almost all licensing restrictions by being sufficiently creative and inventive. Thus, there is a need to providing guidance to both program acquisition officers, Program Executive Offices, and Defense contractors for how to specify requirements for military software applications that best achieve a cost effective level of openness that can enable the maximum possible benefits anticipated. But without explicit guidance or guidelines, then we cannot assume that OA will just happen because of the use of OSS elements and open systems APIs.

With this in mind, we turn to put forward some initial guidelines for such requirements.

Discussion

The relationship between open technology, open architecture, and open source software requirements, and program acquisition is poorly understood. We can call such a view of OSS *product oriented*. Alternatively, we can view OSS as (b) primarily a set of development processes, work practices, project community activities (code sharing, review, modification, redistribution), and multi-project software ecosystem that produce OSS systems and components. This view of OSS as an integrated web of people, processes, and organizations (including project teams operating as virtual organizations [Noll and Scacchi 1999, Crowston and Scozzi 2002]) is *production oriented* (including production processes, production organizations, production people, and governance over software production [Scacchi 2007, Scacchi, Feller, *et al.* 2006, Scacchi and Jensen 2008]). The requirements for (a) are not the same as for (b), and thus program acquisition targeting (a) may fail to realize the benefits, capabilities, or constraints engendered by (b), and vice versa. As such, there is need to understand how to identify an optimal mix of OSS within OA as both products, and production processes, practices, community activities, and multi-project (or multi-organization) software ecosystem.

The success of DoD's OA and OSS programs in achieving the positive qualities associated with OSS depend on the socio-technical context in which a system is developed and used. The stakeholders and users of an OSS system typically include the developers of that system; they know its goals and requirements implicitly, and can adapt and evolve the system to follow their understanding of the context in which it is used. If DoD is to achieve quick response, rapid adaptation, and context-appropriate use of OSS, it may be necessary to have a representative group of the personnel that are to use and adapt it to the needs they see around them, be OSS developers for that system.

Following from our analysis above, it appears there are a new set of requirements that are emerging that will need to be addressed in any acquisition of a software-intensive system that is stipulated to employ an OA that accommodates OSS components or connectors. Identifying specific requirements for a given program acquisition or system development contract can benefit from consideration of the the following guidelines for how best to realize an OA:

- Determining how much openness is required or desired.
- Identifying guidelines and incentives for software development contractors that encourage them to develop, provide, and distribute/deploy OA systems with OSS components, connectors, and configuration that minimize conflicting OSS license obligations.
- Determining the restrictions, if any, that the OSS licenses used by different software system components, connectors, or configurations within a OA system.
- Identifying alternative OSS component, connector, or configuration candidates that may satisfy a specified overall system architecture.
- Determining scenarios that help reveal whether there are OSS licensing conflicts for a given set of OSS components, connectors, or configuration.
- Identifying and analyzing any OSS licensing obligations that must be satisfied for the resulting system to be available for redistribution.
- Identifying and validating OSS license conformance criteria for configured systems

intended for redistribution.

Further elaboration on these guidelines is subject to additional research, application, and refinement. However, they do provide a useful starting point for discussion, debate, and action in program acquisition.

Conclusions

The relationship between open technology, open architecture, and open source software requirements, and program acquisition is poorly understood. In recent OA presentations, OSS is viewed as primarily a source for low-cost/free software systems or software components. Thus, given the goal of realizing an OA and open technology strategy [cf. Herz and Scott 2007] together with the use of OSS components and open APIs, it is unclear how to best align program acquisition, system requirements, software architectures, and OSS license regimes to achieve this goal. Subsequently, the central problem we examined in this paper was to identify principles of software architecture and OSS copyright licenses that facilitate or inhibit how best to insure the success of an OA strategy when OSS and open APIs are required or otherwise employed.

Consideration of emerging issues in the acquisition of OSS within the U.S. Department of Defense is an important problem for acquisition research at this time. The goal of this paper is to help establish a foundation for how to analyze and evaluate dependencies that might arise when seeking to develop software systems that should embody an OA when different types of OSS components or OSS component licenses are being considered for integration.

ACKNOWLEDGMENTS

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

References

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

Bass, L., Clements, P., and Kazman, R., (2003). *Software Architecture in Practice*, 2nd Edition, Addison-Wesley Professional, New York.

Bollinger, T., (2003). *Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense*, The MITRE Corporation, 2 January. Available at http://www.terrybollinger.com/dodfoss/dodfoss_html/index.html

Choi, J.S. And Scacchi, W., (2001). Modeling and Simulating Software Acquisition Process Architectures, *J. Systems and Software*, 59(3), 343-354, 15 December.

Crowston, K. and Scozzi, B., (2002). Open Source Software Projects as Virtual Organizations. *IEE Proceedings--Software*, 149, 1, 3-17.

Diallo, M., Sim, S.E., and Alspaugh, T.A., (2007). The Mythical Requirements-Architecture Gap, submitted to *European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering* (ESEC/FSE 2007).

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

Fielding, R. and Taylor, R.N., (2002). Principled Design of the Modern Web Architecture, *ACM Transactions Internet Technology*, 2(2), 115-150.

Guertin N., (2007). (Director, Open Architecture, Program Executive Office IWS 7B). *Naval Open Architecture: Open Architecture and Open Source in DOD*, Presentation at "Open Source - Open Standards - Open Architecture," Association for Enterprise Integration Symposium, Arlington VA, 14 March 2007.

Herz, J.C. And Scott, J., (2007). COTR Warriors: Open Technologies and the Business of War, *The DoD Software Tech News*, 10(2), 3-6, June. https://www.softwaretechnews.com/stn_view.php?stn_id=42

Justice, Brig. General Nick (2007a). (Program Executive Office C3T), *Open Source Software Challenge: Delivering Warfighter Value*, Presentation at "Open Source - Open Standards - Open Architecture," Association for Enterprise Integration Symposium, Arlington VA, 14 March.

Justice, Brig. General Nick (207b). (Program Executive Office C3T), *Deploying Open Technologies and Architectures within Military Systems*, Presentation at **3rd** DoD Open Conference, Deployment of Open Technologies and Architectures within Military Systems, Association for Enterprise Integration Symposium, Arlington VA, 12 December.

Kazman, R. and Carrière, J. (1999). Playing Detective: Reconstructing Software Architecture from Available Evidence. *J. Automated Software Engineering,* 6(2), 107-138.

Kuhl, F., Weatherly, R., and Dahmann, J. (2000). *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Prentice-Hall PTR, Upper Saddle River, New Jersey.

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

Navy, (2006). *Naval OA Strategy*. https://acc.dau.mil/oa

Nelson L. and Churchill, E.F., (2006). Repurposing: Techniques for Reuse and Integration of Interactive Services, *Proc. 2006 IEEE Intern. Conf. Information Reuse and Integration*, September.

Noll, J. and Scacchi, W., (1999). Supporting Software Development in Virtual Enterprises, *Jour. Digital Information*, 1(4), February.

OSI (2008). The Open Source Initiative, http://www.opensource.org/

Riechers, C., (2007). (Principal Deputy, Asst. Sect. of the Air Force, Acquisition). *The Role of Open Technology in Improving USAF Software Acquisition*, Presentation at “Open Source - Open Standards - Open Architecture,” Association for Enterprise Integration Symposium, Arlington VA, 14 March.

Rosen, L. (2005). *Open Source Licensing: Software Freedom and Intellectual Property Law*, Prentice-Hall PTR, Upper Saddle River, New Jersey. http://www.rosenlaw.com/oslbook.htm

Scacchi, W., (2002). Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(1), 24-39, February.

Scacchi, W., (2007). Free/Open Source Software Development: Recent Research Results and Methods, in M. Zelkowitz (Ed.), *Advances in Computers*, 69, 243-295.

Scacchi, W., Feller, J., B. Fitzgerald, Hissam, S.and Lakhani, K., (2006). Understanding Free/Open Source Software Development Processes, *Software Process--Improvement and Practice*, 11(2), 95-105, March/April.

Scacchi, W. and Jensen, C., (2008). Governance in Open Source Software Development Projects: Towards a Model for Network-Centric Edge Organizations, *Proc. 13th. Intern. Command and Control Research and Technology Symp.*, Bellevue, WA, (to appear, June).

Starrett, E., (2007). Software Acquisition in the Army, *Crosstalk: The Journal of Defense Software Engineering*, 4-8, May, http://stsc.hill.af.mil/crosstalk.

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

Weathersby, J.M., (2007). Open Source Software and the Long Road to Sustainability within the U.S. DoD IT System, *The DoD Software Tech News*, 10(2), 20-23, June. https://www.softwaretechnews.com/stn_view.php?stn_id=42

Wheeler, D.A., (2007). Open Source Software (OSS) in U.S. Government Acquisitions, *The DoD Software Tech News*, 10(2), 7-13, June. https://www.softwaretechnews.com/stn_view.php?stn_id=42