NPS-AM-08-036

EXCERPT FROM THE

PROCEEDINGS

OF THE

FIFTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM

EMERGING ISSUES IN THE ACQUISITION OF OPEN SOURCE SOFTWARE WITHIN THE US DEPARTMENT OF DEFENSE

Published: 23 April 2008

by

Dr. Walt Scacchi and Dr. Thomas Alspaugh

5th Annual Acquisition Research Symposium of the Naval Postgraduate School:

Acquisition Research: Creating Synergy for Informed Change

May 14-15, 2008

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

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

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

Emerging Issues in the Acquisition of Open Source Software within the US Department of Defense

Presenter: 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.

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-34 55 USA
Phone: 1-949-824-4130
Fax: 1-949-824-1715
E-mail: wscacchi@ics.uci.edu

Author: 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 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.

Thomas A. Alspaugh
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-34 55 USA
Phone: 1-949-824-4130
Fax: 1-949-824-1715
E-mail: alspaugh@ics.uci.edu

Abstract

In the past five or so years, it has become clear that the US 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) that 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 additional requirements that may be needed to fulfill an OA strategy during program acquisition.

Introduction

Interest within the US Department of Defense (DoD) and military services in free and open source software (OSS) first appeared in the past five or so years (Bollinger, 2003). More recently, it has become clear that the US Air Force, Army, and Navy have 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 & Scott, 2007). Why?

According to Riechers (2007), the Air Force sees several factors within its software-intensive systems: there is increasing complexity of the software (code) itself; the Air Force may be "held hostage" by proprietary legacy components; it seeks more timely delivery of new solutions, and it is aware 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 (2007, March; 2007, December), 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 in order to better develop domain and IT expertise, enable the transition to Web 2.0 technologies, and 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 (US 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 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 the identification of principles of software architecture and OSS copyright licenses that facilitate or inhibit 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 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 less expensive 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 PMs seek to develop software systems that should embody an OA—especially 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) that can interact with acquisition processes (Choi & Scacchi, 2001) in ways that are not apparent within current public perspectives for OA, based on OSS (Guertin, 2007; Justice, 2007, March; 2007, December; 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 entails from a software architecture standpoint, software licensing regimes, and how/where they interact. We start by providing additional background on "openness." We then add a description and analysis of open software architecture concepts and of open source software licenses. This gives rise to a discussion that identifies 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 for one military service will realize the same kinds of benefits anticipated for OA-based systems by another service (Wheeler, 2007). Somehow, DoD acquisition program managers must make sense of or reconcile such differences in expectations and outcomes from OA strategies across the 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 been disseminated (Weathersby, 2007). Instead, what mostly exists are narratives that serve to provide and promise the 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 executed (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 correctness, however, a system may be completely open, or may fail to be open in various ways. 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 correctness, openness may—ultimately—be the province of lawyers and policy makers, not engineers or users.

We believe that a primary challenge is how to determine whether a system, composed of sub-systems 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 also planning for a proposed system to ensure that the result is "open" under the desired definition, with only the acceptable licenses applying. It is also important to understand which licenses are acceptable in this context. Because there are a range of licenses (each of which may affect a system in a different way), and due to the number of various kinds of OSS-related components and ways of combining them (which have an effect on the licensing issue), the first step in this process is to understand types of software elements that constitute a software architecture, and the types of licenses that may encumber these elements or their overall configuration.

OA seems 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 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) whether many alternative architectural configurations and APIs may or may not produce an OA (Alspaugh & Antón, 2007; Diallo, Sim, & Alspaugh, 2007; Scacchi, 2007). Subsequently, we believe this can lead to complex situations: 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, which 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, or what architecture style (Bass, Clements & Kazman, 2003) is implied by given system requirements.

Thus, given the goal of realizing an OA and open technology strategy (Herz & 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.

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 which possible mix of software elements may be configured into it. To help explain this, we first identify the types of software elements included in common software architectures, whether they are open or closed (Bass et al., 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 currently at least four forms of computer programs.

- *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 by associating their respective inputs, outputs, and control variables. This software is sometimes called "glue code," which suggests its primary use is to connect programs through the use of "pipes" and/or "filters" that control or modulate the directed flow of information between the associated programs. 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 hundreds of thousands) of 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 & 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) of 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 that 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 is the minimum required to form an “open system” (Meyers & Obendorf, 2001). Often, 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.

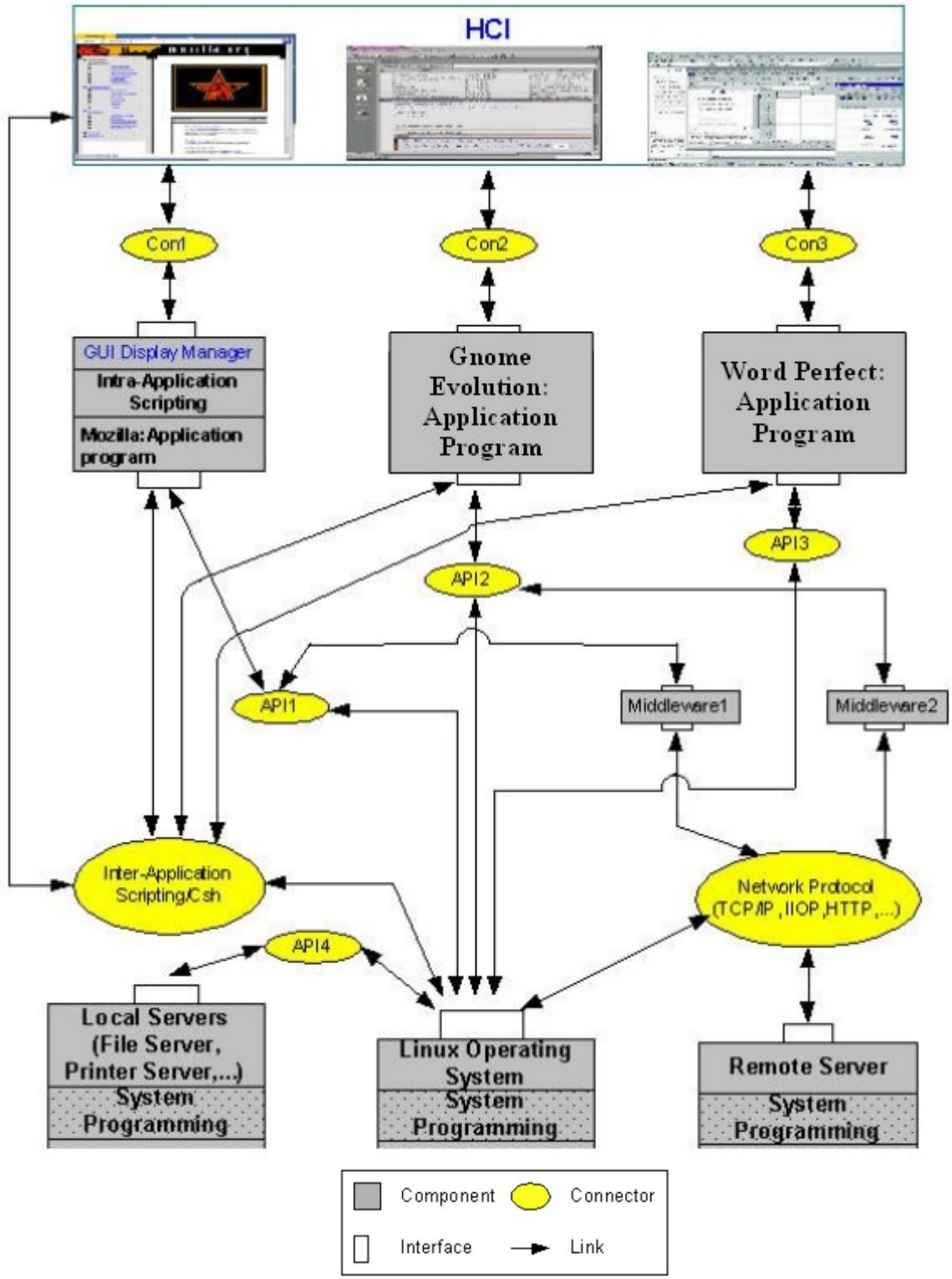
- *Software connectors*—These may be software either from libraries, frameworks, or application script code whose intended purpose is to provide a standard or reusable way of associating programs, data repositories, or remote services through common interfaces. These may include software technologies 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 & Dahmann, 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 & Taylor, 2002) for Web-based client-server applications or may represent an original or ad hoc architectural pattern (Bass et al., 2003). All the software elements, and how they are arranged and interlinked, can all be specified, analyzed, and documented using an Architecture Description Language (Bass et al., 2003) and ADL-based support tools. Beyond this, any or all of the software elements in a configured system or sub-system may or may not be OSS. 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. It also includes 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 e-mail 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 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 that describe and explain the many different licenses now in use with OSS (OSI, 2008; Rosen, 2005; St. Laurent, 2004). Thus, 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.



Note: Components, connectors, and overall system configuration may be subject to different software licenses.

Figure 1. Software Components, Connectors, Interfaces Arranged in an Overall Software System Configuration

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 e-mail 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. Figure 1 follows the links between the Mozilla Web browser, Gnome Evolution, and WordPerfect. The requirements imposed by a component's license are 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 (Kazman & Carrière, 1999). We offer the following considerations to clarify this. 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 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 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) is intended to be adapted to incorporate new innovative software technologies that are not yet at hand, 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, the 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).

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

Figure 2a. Mapping *Reciprocal* OSS Licenses to Derivative Works
(Rosen, 2005)

CONTRIBUTION	DERIVATIVE WORK				
	GPL	MPL	CPL	OSL	Academic
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.

Note: Footnotes in original (Rosen, 2005, p. 251).

Figure 2b. Mapping *Unrestrictive* Academic to Reciprocal OSS Licenses

Moving forward, analyses of OSS licenses by intellectual property lawyers may suggest a way out of the current OSS licensing/relicensing mess. 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 offer some encouraging words. 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 are modified and redistributed and not to software that may be modified but not distributed outside of the organization. Also, this raises the questions of 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), 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 involved software elements. 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 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, which can enable the maximum possible benefits anticipated. But, without explicit guidance or guidelines, we cannot assume that OA will just happen because of the use of OSS elements and open systems APIs.

With this in mind, we outline some initial guidelines for such requirements.

Discussion

The relationships among open technology, open architecture, open source software requirements, and program acquisition is poorly understood. We can call such a view of OSS: (a) *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 & Scacchi, 1999; Crowston & 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 & Jensen, 2008)). The requirements for (a) are not the same as for (b), and 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 ecosystems.

The success of the DoD's OA and OSS programs in achieving the positive qualities associated with OSS depends 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 the DoD is to achieve quick response, rapid adaptation, and context-appropriate use of OSS, it may require a representative group of the personnel who use and adapt it to their needs be OSS developers for that system.

Following our analysis above, it appears there are a new set of requirements 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. PMs that identify specific requirements for a given program acquisition or system development contract can benefit from consideration of 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 apply to the OSS licenses used by different software system components, connectors, or configurations within an 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 relationships among open technology, open architecture, 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 (Herz & 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. Subsequently, the central problem we examined in this paper was how 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.

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

List of References

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

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

Bollinger, T. (2003, January 2). *Use of free and open-source software (FOSS) in the US Department of Defense.* The MITRE Corporation. Retrieved March 2008, from http://www.terrybollinger.com/dodfoss/dodfoss_html/index.html

Choi, J.S., & Scacchi, W. (2001, December 15). Modeling and simulating software acquisition process architectures. *Journal of Systems and Software*, *59*(3), 343-354.

Crowston, K., & Scozzi, B. (2002). Open source software projects as virtual organizations. *IEEE Proceedings—Software*, *149*(1), 3-17.

Diallo, M., Sim, S.E., & Alspaugh, T.A. (2007). The mythical requirements-architecture gap. Submitted to the 2007 European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE).

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

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

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

Herz, J.C., & Scott, J. (2007, June). COTR warriors: Open technologies and the business of war. *The DoD Software Tech News*, *10*(2), 3-6. Retrieved March 2008, from https://www.softwaretechnews.com/stn_view.php?stn_id=42

Justice, N., Brig. Gen., Program Executive Office C3T. (2007, March 14). *Open source software challenge: Delivering warfighter value*. Remarks delivered at "Open Source—Open Standards—Open Architecture," Association for Enterprise Integration Symposium, Arlington, VA.

Justice, N., Brig. Gen., Program Executive Office C3T. (2007, December 12). *Deploying open technologies and architectures within military systems*. Remarks delivered at 3rd DoD Open Conference, Deployment of Open Technologies and Architectures within Military Systems, Association for Enterprise Integration Symposium, Arlington, VA.

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.

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 of Information Reuse and Integration.* Los Alamitos, CA: IEEE.

Noll, J., & Scacchi, W. (1999, February). Supporting software development in virtual enterprises. *Digital Information*, *1*(4).

OSI. (2008). *The open source initiative*. Retrieved March 2008, from http://www.opensource.org/

Riechers, C., Principal Deputy, Asst. Sect. of the Air Force, Acquisition. (2007, March 14). *The role of open technology in improving USAF software acquisition*. Remarks delivered at "Open Source—Open Standards—Open Architecture," Association for Enterprise Integration Symposium, Arlington, VA.

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 methods. In M. Zelkowitz (Ed.), *Advances in Computers*, *69*, 243-295.

Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., & Lakhani, K. (2006, March/April). Understanding free/open source software development processes. *Software Process—Improvement and Practice*, *11*(2), 95-105.

Scacchi, W., & Jensen, C. (2008). *Governance in open source software development projects: Towards a model for network-centric edge organizations.* Remarks delivered at the 13th International Command and Control Research and Technology Symposium, Bellevue, WA. To appear in June.

Starrett, E. (2007, May). Software acquisition in the army. *Crosstalk: The Journal of Defense Software Engineering*, 4-8. Retrieved March 2008, from http://stsc.hill.af.mil/crosstalk

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

US Navy. (2006). *Naval OA strategy*. Retrieved March 2008, from https://acc.dau.mil/oa

Weathersby, J.M. (2007, June). Open source software and the long road to sustainability within the US DoD IT system. *The DoD Software Tech News*, *10*(2), 20-23. Retrieved March 2008, from https://www.softwaretechnews.com/stn_view.php?stn_id=42

Wheeler, D.A. (2007, June). Open source software (OSS) in US government acquisitions. *The DoD Software Tech News*, *10*(2), 7-13. Retrieved March 2008, from https://www.softwaretechnews.com/stn_view.php?stn_id=42

Acknowledgments

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

THIS PAGE INTENTIONALLY LEFT BLANK

2003 - 2008 Sponsored Research Topics

Acquisition Management

- Software Requirements for OA
- Managing Services Supply Chain
- Acquiring Combat Capability via Public-Private Partnerships (PPPs)
- Knowledge Value Added (KVA) + Real Options (RO) Applied to Shipyard Planning Processes
- Portfolio Optimization via KVA + RO
- MOSA Contracting Implications
- Strategy for Defense Acquisition Research
- Spiral Development
- BCA: Contractor vs. Organic Growth

Contract Management

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

Financial Management

- PPPs and Government Financing
- Energy Saving Contracts/DoD Mobile Assets
- Capital Budgeting for DoD
- Financing DoD Budget via PPPs
- ROI of Information Warfare Systems
- Acquisitions via leasing: MPS case
- Special Termination Liability in MDAPs

Human Resources

- Learning Management Systems
- Tuition Assistance
- Retention
- Indefinite Reenlistment
- Individual Augmentation

Logistics Management

- R-TOC Aegis Microwave Power Tubes
- Privatization-NOSL/NAWCI
- Army LOG MOD
- PBL (4)
- Contractors Supporting Military Operations
- RFID (4)
- Strategic Sourcing
- ASDS Product Support Analysis
- Analysis of LAV Depot Maintenance
- Diffusion/Variability on Vendor Performance Evaluation
- Optimizing CIWS Lifecycle Support (LCS)

Program Management

- Building Collaborative Capacity
- Knowledge, Responsibilities and Decision Rights in MDAPs
- KVA Applied to Aegis and SSDS
- Business Process Reengineering (BPR) for LCS Mission Module Acquisition
- Terminating Your Own Program
- Collaborative IT Tools Leveraging Competence

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

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

www.acquisitionresearch.org