SYM-AM-16-033

PROCEEDINGS
OF THE
THIRTEENTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM

WEDNESDAY SESSIONS
VOLUME I

Achieving Better Buying Power for Mobile Open Architecture Software Systems Through Diverse Acquisition Scenarios

Walt Scacchi, Senior Research Scientist, Institute for Software Research, UC Irvine
Thomas Alspaugh, Project Scientist, Institute for Software Research, UC Irvine

Published April 30, 2016

Prepared for the Naval Postgraduate School, Monterey, CA 93943.

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

The research presented in this report was supported by the Acquisition Research Program of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

To request defense acquisition research, to become a research sponsor, or to print additional copies of reports, please contact any of the staff listed on the Acquisition Research Program website (www.acquisitionresearch.net).

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

Panel 6. Considerations in Software Modeling and Design

Wednesday, May 4, 2016	
1:45 p.m. – 3:15 p.m.	**Chair: John Zangardi,** Deputy Assistant Secretary of the Navy for Command, Control, Communications, Computers, Intelligence, Information Operations, and Space ***Achieving Better Buying Power for Mobile Open Architecture Software Systems Through Diverse Acquisition Scenarios*** Walt Scacchi, Senior Research Scientist, Institute for Software Research, UC Irvine Thomas Alspaugh, Project Scientist, Institute for Software Research, UC Irvine ***Architecting Out Software Intellectual Property Lock-In: A Method to Advance the Efficacy of BBP*** Maj Chris Berardi, USAF; Bruce Cameron, Lecturer, MIT; Daniel Sturtevant, CEO, Silverthread, Inc.; Carliss Baldwin, Professor, Harvard Business School; and Edward Crawley, Professor, MIT ***Navy Mobile Apps Acquisition: Doing It in Weeks, Not Months or Years*** Jacob Aplanalp, Assistant Program Manager, My Navy Portal, PEO EIS; Dave Driegert, Senior Technical Advisor, PEO EIS; Kevin Burnett, Technical Manager, PEO EIS; and Kenneth Johnson, Technical Director, PEO EIS

Achieving Better Buying Power for Mobile Open Architecture Software Systems Through Diverse Acquisition Scenarios

Walt Scacchi—is 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 170 research papers, and has directed more than 65 externally funded research projects. In 2011, he served as co-chair for the 33rd International Conference on Software Engineering—Practice Track, and in 2012, he served as general co-chair of the 8th IFIP International Conference on Open Source Systems. [wscacchi@ics.uci.edu]

Thomas Alspaugh—is a Project Scientist at the Institute for Software Research, University of California, Irvine. His research interests are in software engineering, requirements, and licensing. Before completing his PhD, he worked as a software developer, team lead, and manager in industry, and as a computer scientist at the Naval Research Laboratory on the Software Cost Reduction, or A-7, project. [thomas.alspaugh@acm.org]

Abstract

The U.S. Defense community denotes an ecosystem of system or software component producers, system integrators, and customer organizations. For a variety of reasons this community now embraces the need to utilize open source software (OSS) and proprietary closed source software (CSS) in the system capabilities or software components it acquires, design, develops, deploys, and sustains. But the long-term transition to agile and adaptive capabilities that integrate bespoke or legacy, OSS and CSS components, has surfaced a number of issues that require acquisition-research-led approaches and solutions. In this paper, we identify and describe six key issues now found in the Defense software ecosystem: (1) unknown or unclear software architectural representations; (2) how to best deal with diverse, heterogeneous software IP licenses; (3) how to address cybersecurity requirements; (4) challenges arising in software integration and release pipelines; (5) how OSS evolution patterns transform software IP and cybersecurity requirements; and (6) the emergence of new business models for software distribution, cost accounting, and software distribution. We use the domain of command and control systems under different acquisition scenarios as our focus to help illuminate these issues along the way. We close with suggestions for how to resolve them.

Introduction

The U.S. Defense community, which includes the military services and civilian-staffed agencies, is among the world's largest acquirers of commodity and bespoke (custom) software systems. The Defense community further extends its reach and influence on a global basis through national treaties and international alliances through enterprises like NATO. The Department of Defense (DoD), other government agencies, and most large-scale business enterprises continually seek new ways to improve the functional capabilities of their software-intensive systems while lowering acquisition costs. The acquisition of open architecture (OA) systems that can adapt and evolve through replacement of functionally similar software components is an innovation that can lead to lower cost systems with more powerful functional capabilities. OA system acquisition, development, and deployment are thus seen as an approach to realizing Better Buying Power (BPP) goals for lowering system costs, achieving technical excellence, enabling innovation, and advancing the acquisition workforce (Kendall, 2015).

Bespoke software systems are produced and integrated within the Defense community. In addition Defense system acquisition or procurement enterprises also obtain wares from most non-Defense industry providers of software systems, applications, or services (i.e., the mainstream software products or services industry). The acquisitions often entail software procurement or development contracts valued in the millions to hundreds-of-millions of dollars (Myers & Obendorf, 2001). At this scale of endeavor and economic value, certain kinds of software engineering (SE) research problems arise that are not visible or are insignificant in smaller scale SE R&D efforts.

In this paper, we focus attention to the slice of this world that focuses on the development and deployment of *software-intensive command, control, communication, cyber and business systems* (hereafter, C3CB). We further limit our focus to the most general software elements found in C3CB system capabilities; for example, software infrastructure components, common development technologies supporting app/widget development, and mission-specific apps/widgets, in particular widgets produced with the Ozone Widget Framework (Conley et al., 2014). OWF (now called the Ozone Platform or OZP) was initially developed by the NSA, though is now identified as Government OSS (GOSS) and supported by a third-party contractor. It is widely used within the Defense and Intelligence community. The growing importance of OZP within the Defense community has directed focus to the production and integration of C3CB system capabilities to be assembled using it. This focus drives open discussion of and broad exposure to emerging research issues that arise from the production and integration (or software engineering—SE) of software components, and these in turn raise challenges for acquisition management and personnel. Specifically, we draw attention to issues surrounding the development, integration, and deployment of *multi-version* and *multi-variant software systems* composed from various open source software (OSS) and proprietary (CSS) software elements or remote services (Scacchi, 2002, 2010), eventually including recent efforts to support Web-compatible services and/or mobile devices in C3CB. This focus also provides exposure to future C3CB system capabilities composed from apps acquired through various acquisition regimes, including apps downloaded from different Defense community app stores (George, Morris, O'Neil, et al., 2013; George et al., 2014).

Recent Scenarios for Acquisition of OA Software Capabilities

Interest in open source software (OSS) within the U.S. Department of Defense (DoD) and military services first appeared more than 10 years ago (Bollinger, 2003; Scacchi & Alspaugh, 2008). More recently, it has become clear that the U.S. Defense community has committed to a strategy of acquiring software-intensive systems across the board that require or utilize an "open architecture" (OA) which may incorporate OSS technology or OSS development processes that can help Defense customer organizations to achieve better buying power (Kendall, 2015). Why? Among the reasons identified is the desire to realize more choices among software component producers or integrators, as producers and integrators often act in ways that lock their customer organizations into overly costly and sometimes underperforming and difficult to sustain systems. One approach being explored focuses attention to agile and adaptive OA software components that are acquired and assembled (integrated) as C3CB system capabilities (assembled capabilities or AC) that are acquired and shared by multiple parties via independent "lines of efforts" acting within an ecosystem of producers, integrators, and consumer organizations (Reed et al., 2014; Scacchi & Alspaugh, 2015). The goals of the AC approach include a shorter delivery and update cycle for mission components and an improved cybersecurity posture. We explain this approach as follows.

The AC approach contemplates independent acquisition lines of effort for different types of OA software components that can be acquired from independent providers:

- *Mission Components* enable C3CB processes and present common operating picture data to end-users. Mission components may be realized as apps/widgets that may be deployed on mission-specific platforms, including those operating on secured Web/mobile devices.
- *Common Development Technology* provides AC development tools and common run-time applications servers that support the mission components. The servers are bundled with Shared Infrastructure, as follows.
- *Shared Infrastructure Components* combine local/remote application servers and data repositories with networking services and platforms.

Assembled capabilities therefore represent alternative configurations of mission-specific components that are produced with common development technology for deployment on shared infrastructure technology platforms.

Independent Lines of Effort (LOEs) by single or multi-party acquisition for mission components, common development technologies, or shared infrastructure components, are expected to greatly accelerate development and fielded deployment. This acceleration entails tradeoffs in increased dependency and risk management. Independent LOEs enable at least three alternative scenarios for acquiring OA C3CB system capabilities.

1. *Use current strategy and acquisition capabilities*. Here there is no focus on AC that utilizes mission components, common development technologies, or shared infrastructure components.
2. *Augment deployed systems with mission components and common technologies*. Augmentation is either for (a) new mission functionality; (b) modernization "in place" so that part of the original system is deprecated as the new mission components are delivered; or (c) infrastructure replacement over parts of original system that may be combined with modernization efforts.
3. *Focus efforts on production, integration, security assurance, and deployment of mission components that use common technologies and shared infrastructure, and that can be assembled into different ACs*. This can entail production, integration, and delivery of all mission components in one contract vehicle; or alternatively, the delivery of mission components partitioned across multiple acquisition contract vehicles, so as to spread and manage risk, while insuring multi-party buy-in commitment.

The following efforts provide examples where these alternative C3CB acquisition scenarios can be considered. First, the Air Force's Theater Battle Management Core System–Force Level (TBMCS-FL), which manages air tasking orders and airspace management, among other things, is being harvested for current operational capabilities. These capabilities can then be encapsulated and delivered as mission components for other C3CB systems, using OZP widgets and supporting common technologies. The C2AOS C2IS acquisition scenario also intends to deliver harvested functionality as mission components. Air Force AOC (Air Operations Center) is planning to include C2AOS C2IS as the replacement for TBMCS-FL, and will use the Navy ACS (hence indicating the need for multi-party acquisition agreements). This in turn implies the need for Joint C2, and needs to be copied to all Services. It represents an opportunity to reduce duplicate activities for producing equivalent C3CB system capabilities. Second, the Army's Distributed Common Ground System (DCGS-A) currently uses mission components for visualization (over 300

widgets available). DCGS-A will incorporate metadata mission components that utilize the DCGS Integration Backbone (DIB). Third and last, the Navy is deploying CANES and ACS (Agile Core Services) shared infrastructure to its fleet as a modernization effort (Guertin, Sweeney, & Schmidt, 2015).

There are now a number of policy directives within the Defense community that formally recognize that OSS system elements can be treated as commercial-off-the-shelf (COTS) components, and that bespoke software system development projects will utilize an OA, unless otherwise justified and approved. Thus, developing contemporary C3CB that incorporate both OSS and new/legacy CSS elements are "business as usual." However, many legacy Defense community system capability producers are hesitant about how best to engineer such OA/OSS systems. For example, does an OA system imply/require that its software architecture be explicitly modeled, be accessible for sharing/reuse (e.g., as a Reference Model), and be modeled in a form/notation that is amenable to architectural analysis and computational processing ("Software Architecture," 2016)? Therefore, we can begin to identify what kinds of SE research issues can be observed and investigated within the Defense community associated with its transition to OA systems and OSS software elements, specifically for Web and Mobile devices within the realm of C3CB.

OA, Open APIs, OSS, and CSS

OA C3CB system capabilities are assembled with mission components, common development technologies, and infrastructure. *Infrastructure components* are broadly construed to include non-mission specific software functionality or operations. Such components can include computer operating systems, Web servers, database management systems, cloud services, mobile device management middleware, and others, along with desktop, mobile, or smartphone-based Web browsers, word processors, email and calendaring, text/voice chat, and end-user media players. Example infrastructure components include the U.S. Army's Common Operating Environment (COE), the Navy's Consolidated Afloat Networks and Enterprises Services (CANES) Afloat Core Services (ACS) (Guertin, Sweeney, & Schmidt, 2015), and similar elements in the Joint Intelligence Environment.

Common development technologies are common software development tools, libraries, or frameworks used to implement the necessary software functionality so that new or legacy mission components can be integrated into mission-specific software capabilities. Software technology frameworks (or common implementation libraries) like Oracle Java 8, Ozone Platform, OpenJDK (OSS Java Development Kernel for Android app development), and the NASA World Wind Java SDK; programming languages like Java or C++; and scripting languages like Javascript may be utilized as common development technologies for developing mission components. Other software production capabilities like the Navy Tactical Cloud and CANES integrate both infrastructure and common development tools like Hadoop, MapReduce, and other mission data analysis tools for the Tactical Cloud, and the Agile Core Services and Java for CANES.

Mission components represent a hybrid assortment of (a) *simple widgets*—small, thin apps similar in spirit to those acquired and downloaded from online app stores (like a clock, calculator, dictionary, sticky note, or unit converter); (b) *singular widgets*—more substantial functional components either created new (bespoke) or extracted from legacy systems that must run on a specific local computing platform (e.g., shipboard fire control system); or (c) *compound widgets*—hosted in a cloud and run as a remote cloud service over a single/multi-tiered client-server software architecture (e.g., Google Maps, NASA World

Wind), and thus potentially accessible and usable on a Web/mobile computing platform (Google Chrome Web browser running on a secure Android mobile device).

OA seems to simply suggest software system architectures incorporating OSS/CSS infrastructure, common development technologies, and mission components that all utilize open application program interfaces (APIs). But not all software system architectures incorporating OSS/CSS components and open APIs will produce an OA, since whether an architecture is an OA depends on (a) how/why OSS/CSS and open APIs are located within it; (b) how OSS/CSS and open APIs are implemented, embedded, or interconnected within it; (c) whether the copyright (Intellectual Property) licenses assigned to different OSS/CSS components encumber all/part of the architecture into which they are integrated; and (d) choices among alternative architectural configurations and APIs that may or may not produce an OA (cf. Scacchi & Alspaugh, 2008). This can lead to situations in which acquisition contracts stipulate a software-intensive system with an OA and OSS/CSS components, but the resulting software system may or may not embody an OA. This can occur when the architectural design of a system constrains the system requirements: if not all requirements can be satisfied by a given system architecture, if requirements stipulate specific types or instances of OSS/CSS (e.g., Web browsers, content management servers), if an architecture style (Bass, Clements, & Kazman, 2003) is implied by given system requirements, or if requirements are implied by the choice to incorporate legacy software capabilities with one architectural style that are to be wrapped within mission-specific widgets with a different architectural style.

Application domain of interest: C3CB with Web/Mobile Devices Utilizing Widgets C3CB are common information system applications that support modern military operations at a regional, national, or global level. These applications may be focused to address common military mission planning, mapping, resource status tracking and scheduling, mission performance, and monitoring activities through application sub-systems. However, closely related C3CB systems applications are also in common use within civilian/public safety agencies, public infrastructure/utility operations, live television and sports event broadcasting, massively multi-player online game operations centers, and even in international motorsports racing competition events like Formula 1. So the study of software production and system integration issues arising in the Defense community can inform awareness of similar issues in other non-Defense software system domains, and vice versa.

Modern C3CB applications are increasingly expected/planned to be composed from best-available software components, whether OSS or CSS, utilizing bespoke or legacy software capabilities. Furthermore, as smartphones, tablets and laptop computers are being brought into the workplace, so too is interest increasing within the Defense community in supporting the acquisition and development of Web-compatible widgets and mobile apps, provided through an emerging ecosystem of component producers and system integrators, for configuration into secure OA C3CB software system capabilities (George et al., 2014; Reed et al., 2012; Reed et al., 2014; Scacchi & Alspaugh, 2013a; Scacchi &Alspaugh, 2015). Common software elements for such systems include Web browsers open to extensions like custom mission-specific Map widgets, and remote content servers, email and calendaring, word processing, local/networked file servers, and operating systems. The data processed by the software may be of high-relevance to military missions/operations, or may just be the daily grind of data manipulated by “productivity” applications which most of us use routinely to perform/enact our work assignments. Security has been mostly addressed through system isolation or “air gaps” to the outside world due, for example, to airborne or afloat capability deployments. But this is no longer common practice, and cybersecurity concerns have risen to the top of functional and non-functional requirements for all such

C3CB applications. New OA systems are now required to be secure by design, by implementation, and through release, deployment and evolution, as well as subject to independent testing and certification. Secure OA designs can then entail different schemes for encapsulating different (sets of) components, use of virtualization schemes, shims and wrappers, encrypting data transfers and storage, and configuring multi-level system access capabilities. But we have found examples in which different OA system designs and configurations propagate security obligations, and privacy protections and access rights are either mediated or nullified by different software component IP licenses or system updates.

OA Ecosystems Within the Defense Community

In our view, a software ecosystem is a network of software component producers, system integrators, and customer organizations. In the Defense community, producers and integrators are commonly industrial entities (defense contractors), while customer organization are military program offices. Figure 1 presents an abstract view of a software ecosystem that associates software components or apps with their producers, system architectures with system integrators, and delivered component or integrated application systems with their customers. We also add annotations to indicate that each component or app has its own software IP license, and that integrated systems delivered to customers come with some composition of IP license obligations and rights propagated through the system's OA.

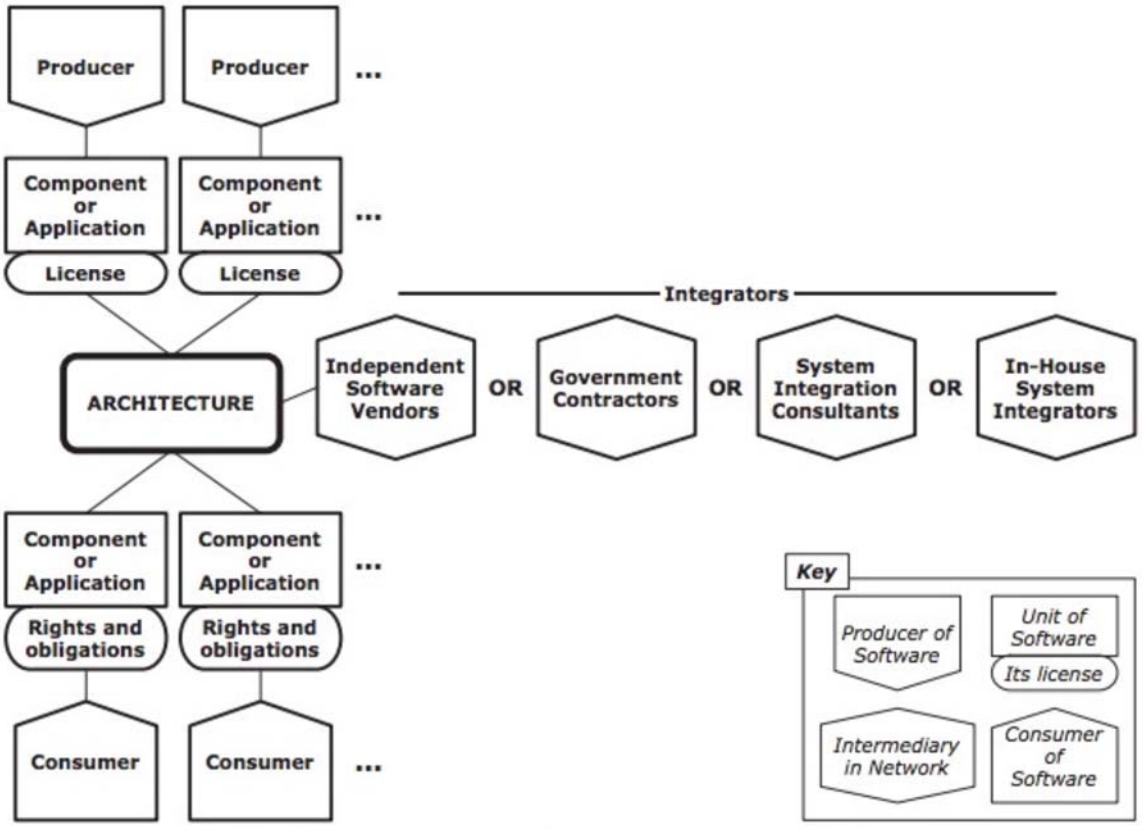


Figure 1. An Abstract Software Ecosystem Rendered as a Network of Software Component Producers, Integrators of Systems/AC, and End-User Consumer Organizations

There is growing interest within the Defense community in transitioning to acquiring complex software system capabilities via an agile and adaptive ecosystem (Reed et al., 2012; Reed et al., 2014; Scacchi & Alspaugh, 2015), where components may be sourced from alternative producers or integrators, allowing for more competition, and ideally lowering costs and improving the quality of software elements that arise from a competitive marketplace (Kendall, 2015). But this adaptive agility to mix, match, reuse, mashup, swap, or reconfigure integrated systems, or to accommodate end-user architecting (Garlan et al., 2012) as in-house integrations of mission components, requires that systems be compatible with or designed to utilize an OA. Consequently, we can identify six kinds of emerging research challenges or issues for software capability acquisition that we have observed within the U.S. Defense community as they move to produce, integrate, deploy and evolve OA systems for C3CB system capabilities that utilize contemporary OSS and bespoke/legacy CSS components. These issues center around (1) unclear representations of OA software system capabilities, (2) how best to accommodate diverse intellectual property licenses when combining bespoke/legacy OSS/CSS mission components, (3) how to accommodate diverse and complicated cybersecurity requirements, (4) technical challenges arising from alternative ways to integrate and deploy diverse software components, (5) how to accommodate many different paths within the Defense community that drive software component evolution, and (6) how to estimate and manage the costs of acquiring, deploying, and sustaining diverse software-based mission components and C3CB system capabilities. These are examined in the next section.

With this background and sets of concepts for understanding a simplified view of the world of C3CB software systems, we now turn to identify and examine a set of issues that are now recurring in the acquisition, design, development, and deployment of such systems.

Emerging Issues in Developing and Deploying OA C3CB Systems Within Different Acquisition Scenarios

There are at least six kinds of emerging research challenges or issues for software capability acquisition that we have observed within the U.S. Defense community as it moves to OA systems for C3CB system capabilities.

Unknown or Unclear OA Solutions

An OA entails a documented representation of software capability described in an architectural description language that specifies component types, component interconnections and connector types, open APIs, and their properties and interrelationships. The common core of a C3CB system OA resembles most enterprise business systems, as C3CB are a kind of management information system for navigating, mapping, tracking resources; scheduling people and other resources; producing plans and documentation; and supporting online email, voice, or video communications. Figure 2 depicts an OA representation that can also serve as a “reference model” for a C3CB software product line (Womble et al., 2011). Figure 3 further expands the sub-architecture of software components that denote configurations of mission-specific components as widgets. Thus, C3CB system capabilities can compose or reuse multiple or nested OA reference models.

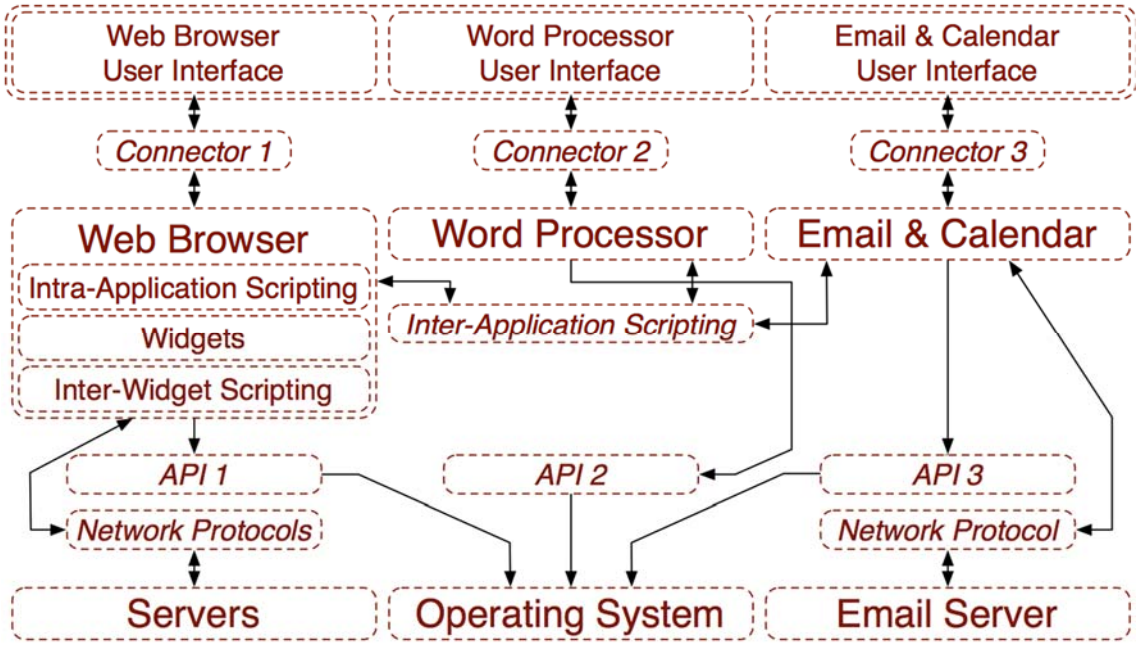


Figure 2. OA Reference Model for Common Software Component Types

Note. This is an OA reference model for common software component types including widgets interconnected within integrated C3CB system capability. Components come from producers that are assembled into OA C3CB capabilities by system integrators.

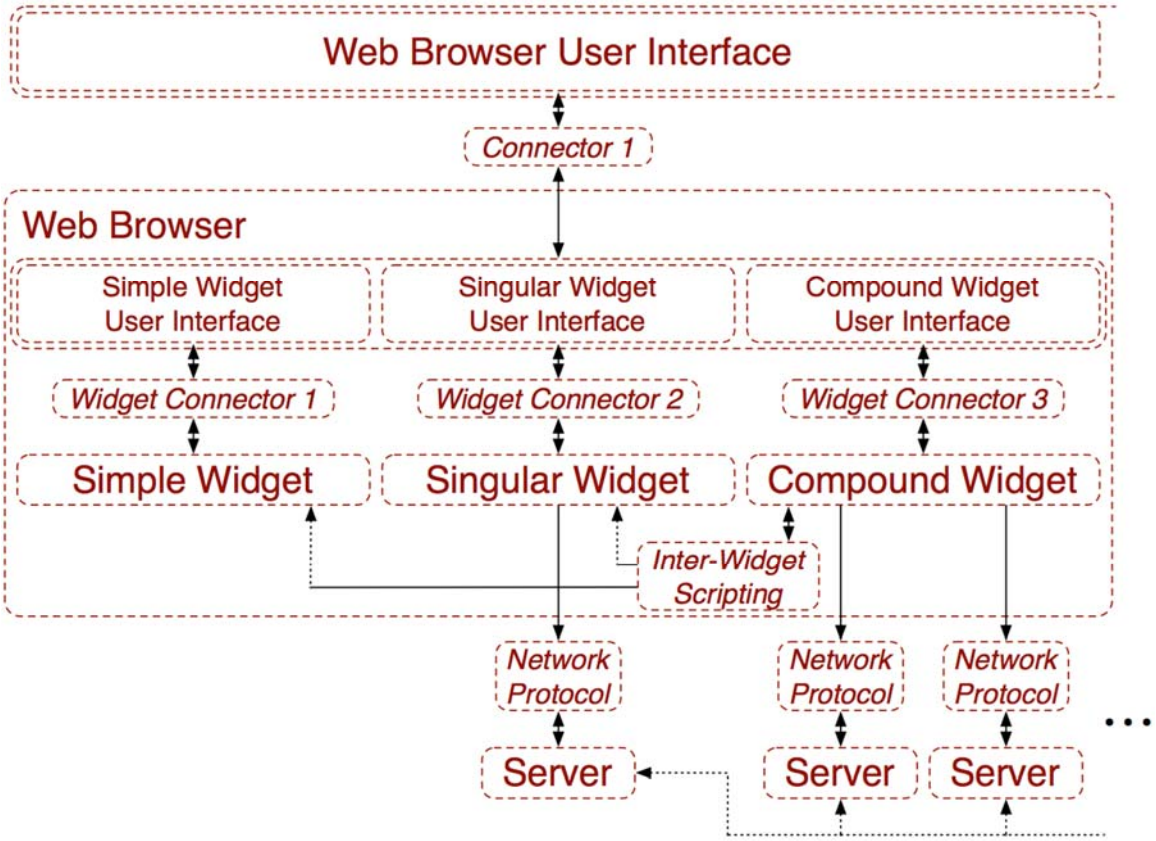


Figure 3. OA Reference Model for Common Types of Software Widget Components

Note. This figure is an OA reference model for common types of software widget components that can be connected and integrated to realize mission-specific C3CB system capabilities, within the overall OA shown on the left-side in Figure 2. Servers may be secured Web content servers, app servers, databases, or file system servers/repositories.

The next piece of the OA challenge we are studying is the envisioned transition with the Defense community to C3CB system capabilities being composed by end-user system integration architects (Garlan et al., 2012) working within/for customer organizations, or potentially extended by end-users deployed in the field. This is the concept that surrounds the transition to discovering software components, apps, or widgets in Defense customer organization app stores (George et al., 2013; George et al., 2014). These app stores are modeled after those used in distributing and acquiring software apps for Web-based or mobile devices, operated by Apple, Google, Microsoft, and others. How the availability of such Defense mission capability app stores will transform the way C3CB systems are produced, or even if legacy Defense industry contractors will produce them, remains to be seen. Said differently, how app stores transform OA software ecosystem networks, business models, and cybersecurity practices is an emerging challenge for acquisition and SE research in the Defense community.

Another kind of challenge arises when acquiring new or retrofitting legacy C2 software system applications that lack an open or explicit architectural representation identifying major components, interfaces, interconnections and remote services (if any). Though OA reference models and architectural description languages are in use within the SE research community, contemporary C3CB generally lack such descriptions or

representations that are open, sharable, or reusable. This may be the result of legacy business practices in the Defense community that see d*etailed software architecture representations as proprietary IP* rather than as open, sharable technical data, even when OSS components are included or when applications sub-systems are entirely made of OSS code. An alternative explanation reveals that complex software systems like common Web browsers (Mozilla Firefox, Google Chrome, Apple Safari, Microsoft Internet Explorer) have complex architectures that integrate millions of SLOC that are not well understood, and that entail dozens of independently-developed software elements with complex APIs and IP licenses that shift across versions (Scacchi & Alspaugh, 2012). For such systems the effort to produce an explicit OA reference model is itself a daunting architectural discovery, component/sub-system extraction, restructuring/refactoring, and continuous software evolution task (Choi & Scacchi, 1990; Kazman & Carriere, 1998). Thus, new ways and means for extracting software components interconnections and interfaces and transforming them into higher-level architectural representations of mission-specific apps/widget configurations are needed.

Harvesting legacy source/executable binary code entails many software engineering challenges that constrain acquisition efforts. First, legacy code provides too much technical detail and comparatively little abstraction of overall system configuration, composition, components and interconnection/dependencies. Second, incongruent computational system models (e.g., legacy data-flow versus publish-subscribe widgets) or hybrid OA AC arise when transitioning legacy system software elements into new widget-based mission components. Third, there is a general inability to visualize or analyze (test, selectively execute, translate into another programming language, etc.) overall system configurations, interconnections, or interfaces. Fourth, lacking these three, the potential for general software reuse is limited to executable code reuse, which is the lowest common denominator for reuse. Such reuse results in substantial blocks of unused code that cannot be easily removed due to indiscernible interdependencies. Last, when configuring mission components that entail legacy C2 software applications wrapped for integration as widgets, different architectural styles can inadvertently be mixed (e.g., dataflow architecture for legacy C2 software, and publish-subscribe architecture for configured mission widgets), which in turns raises the potential for architectural mismatches (Velasco-Elizondo et al., 2013) that may be difficult to determine or detect during system integration, especially when such integration activities are performed by end-user/consumer organizations.

Heterogeneously Licensed OA Software Capabilities

OSS components are subject to widely varying copyright, end-user license agreements, digital civil rights, or other IP protections. The Open Source Institute recognizes dozens of OSS licenses are in use, though the top 10 represents more than 90% of the open source ecosystem (Scacchi & Alspaugh, 2012). This is especially true for OSS components or application systems that incorporate source code from multiple, independent OSS development projects, such as found in contemporary Web browsers like Firefox and Chrome which incorporate components from dozens of OSS projects, most with diverse licenses (Scacchi & Alspaugh, 2012). This means that C3CB system capabilities that entail configuration of OSS/CSS components are subject to complex software IP obligations and rights that may defy tracking, or entail contradictory legal obligations or rights (Alspaugh, Scacchi & Asuncion, 2010). Determining overall IP obligations for such systems is generally beyond the scope of expertise for software developers, as well as most corporate lawyers. Furthermore, we have observed many ways in which IP licenses interact within an OA software system, such that different architectural design choices that configure the same set of software components result in different overall system obligations and rights. Understanding multiple license interaction and IP mismatches is far too confusing for most

acquisition professionals and Program Office decision-makers and a source of legal expense, or alternatively expensive indemnification insurance policies by the software producers or system integrators.

One complication that can be anticipated here arises when component types are replaced with versioned component instance alternatives (Scacchi & Alspaugh 2012). Consider the situation where a Web Browser (e.g., *Firefox 40.0.3* or *Chrome 47.0.2526.111 (64-bit);* etc.) component has a specific IP license (e.g., *Mozilla Public License 2.0* or *GPL 3.0*) associated with the versioned instance, which in turn may be viewed by system integrators as enabling/limiting an integrated system's architectural design, depending on how different components are interconnected in ways that may or may not propagate (un-) desirable IP obligations and rights—a concern that arises frequently when using components subject to the GPL (Scacchi & Alspaugh, 2008). As we have learned in practice, corporate lawyers employed by Defense contractors or in government agencies do not have solutions for how to resolve such complexities, except via costly overall liability indemnification schemes, and efforts to distribute integrated systems with many IP obligations and few rights that effectively make an integrated open source system closed. This in turn can defeat the potential opportunities and benefits for commitment to OA systems that integrate OSS components.

Bespoke/legacy software components for OA AC design, integration and delivery within widgets will be subject to their bespoke/legacy IP obligations. This may include limits on the right to extract, restructure, or reengineer their architecture (cf. Choi & Scacchi, 1990; Kazman & Carriere, 1998) into open source formats. Similarly, IP licenses associated with OSS or new CSS components may impinge on their integration with these legacy components, or may limit disclosure of their interfaces that would allow more open integration of alternative software AC configurations developed by different Defense community component producers (Scacchi & Alspaugh, 2012).

Nonetheless, in our view, OA software ecosystems are defined, delimited, and populated with niches that locate specific integrated system solutions (Scacchi & Alspaugh, 2012). Furthermore, we see that these niches effectively have *virtual IP licenses* that must be calculated via the obligations and rights that propagated across integrated system component licenses via union, intersection, and subsumption relations among them (Alspaugh & Scacchi, 2012). Such calculation may appear to be daunting, and thus begs for a simpler, tractable, and computationally enforced scheme that can scale to large systems composed from many components, as well as be practically usable by C3CB system capability producers, integrators, and acquisition professionals. In such a scheme, OSS/CSS licenses could formalize IP obligations as operational requirements (i.e., computationally enforceable, at the integrated system level) instantiated by system integration architects (Alspaugh, Scacchi, & Asuncion, 2010; Alspaugh & Scacchi, 2013). Similarly, customer/user rights are then non-functional requirements that can be realized and validated as access/update capabilities propagated across the integrated system (Alspaugh & Scacchi, 2013).

Cybersecurity for OA Software Capabilities

Cybersecurity is a high priority requirement in all C3CB systems, applications, AC, and platforms (Scacchi & Alspaugh, 2013c; Scacchi & Alspaugh, 2013d). No longer is cybersecurity something to be addressed after C3CB systems are developed and deployed—cybersecurity must be included throughout the design, development, deployment, and evolution of C3CB. However, the best ways and means for addressing cybersecurity requirements are unclear, and oftentimes somewhat at odds with one another depending on whether cybersecurity capability designs are specific to a C3CB platform

(e.g., operating system or processor virtualization; utilization of low-level operating system access control or capability mechanisms); component producer (secure programming practices and verification testing); system integrator (e.g., via use secure data communications protocols and data encryption); customer deployment setting (mobile: airborne or afloat; fixed: offices, briefing rooms, operations centers); end-user authentication mechanisms; or acquisition policy (e.g., reliance on third-party audit, certification, and assurance of system cybersecurity). However, in reviewing these different arenas for cybersecurity, we have found that the cybersecurity requirements or capabilities can be expressed in much the same way as IP licenses: using concise, testable formal expressions of obligations and rights. Some examples follow (capital letters are placeholders that denote specified system, service, or component contexts):

- The obligation that a user must verify his/her authority by password or other specified authentication process.
- The obligation that all components connected to specified component C must grant it the capability to read and update data in compartment T.
- The obligation to reconfigure a system in response to detected threats, when given the right to select and include different component versions, or executable component variants.
- The right that a user or software component may read and update data in compartment T using the licensed component.
- The right that may allow replacement of a specified component C with some other vetted component.

These examples show how cybersecurity requirements can be expressed or paraphrased in restricted natural language (e.g., using a domain-specific language) into composite specifications that denote “security licenses” (Alspaugh, Scacchi & Asuncion, 2010; Alspaugh & Scacchi, 2012). In this way, it should be possible to develop new software analysis tools whose purpose is to interpret cybersecurity obligations as operational constraints (executable) or provided capabilities (access control or update privileges), through mechanisms analogous to those used for analyzing software licenses (Alspaugh, Scacchi & Asuncion, 2010; Alspaugh & Scacchi, 2012), and show how component or sub-system-specific obligations and rights can be propagated across a system’s architecture.

We similarly envision the ability for OA system capabilities to be produced and integrated according to different cybersecurity requirements, depending on where and how they are deployed (Scacchi & Alspaugh, 2013d). For example, in Figure 4 we show one possible layout of software components that confines different sub-configurations within different virtual machines. These virtual machines may also be hierarchically nested, as is the case when mission-specific widgets that entail legacy C2 applications must be securely confined at run time in order to access remote servers, in contrast to a secured Web browser running on a secured mobile device.

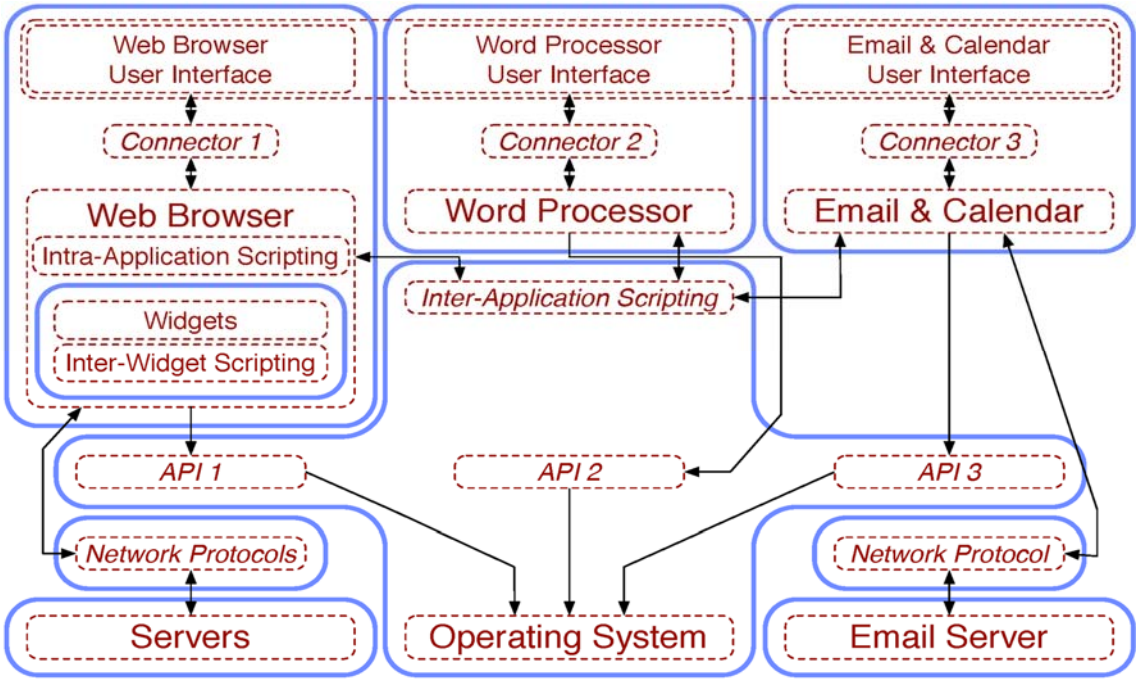


Figure 4. A Configuration of Security Confinement Vessels that Encapsulate Infrastructure Software Components and Mission-Specific Widgets for the OA Shown in Figures 2 and 3

Last, the inclusion of OSS or new CSS components within future OA C3CB software systems or AC will be amenable to current approaches to cybersecurity assurance, as we have outlined before (Scacchi & Alspaugh, 2013d). Mission components can be assessed for cybersecurity characteristics, and assembled, without triggering reaccreditation. Similarly, evolutionary support for field-deployed AC can allow rapid substitution of mission components that enable rapid, agile response to cybersecurity issues in mission components. However, legacy CSS components which were developed and deployed before current cybersecurity assurance challenges will need to rely on "air-gap" interfaces at deployment time that may be vulnerable to aggressive exploits delivered through mobile devices.

Consequently, we believe that cybersecurity can be addressed in the future using explicit, computational OA representations that are attributed with both IP and cybersecurity obligations and rights.

Software Component Build, Release, Deployment (BRD) Processes

C3CB applications represent complex software systems that are often challenging to produce, especially when conceived as bespoke systems. To no surprise, acquisition of these systems often requires a development life cycle approach, though some system elements may be fully-formed components that are operational as packaged software (e.g., commercial database management systems, Web browsers, Web servers, user interface development kits/frameworks). C3CB development is rarely clean-sheet and less likely to be so in the future. As a result, component-based system development approaches are expected to dominate, thus relegating system integrators (or even end-users) to perform any residual source code development, inter-app integration scripting, or intra-app extension

script development. But software process challenges arise along the way (Scacchi & Alspaugh, 2013b).

First is again the issue noted earlier of whether there is an explicit, open-source OA design representation, preferably one that is not just a diagram, but instead is expressed in an architectural design language. With only a diagram or less, then is little or no guidance for how to determine whether a resulting software implementation is verifiable or compliant with its OA requirements or acquisition policies, such as provision or utilization of standardized, open APIs to allow increased software reuse, selection of components from alternative producers, or post-deployment extensions (Kendall, 2015).

Second is the issue arising from system development practices based on utilization of software components, integrated sub-systems, or turnkey application packages. These software elements come with their own, possibly unknown requirements that are nonetheless believed to exist and be knowable with additional effort (Alspaugh & Scacchi, 2013). They also come with either OSS code or CSS executables, along with their respective APIs. These components must be configured to align with the OA specification. Consequently, software tool chains or workflow automation pipelines are utilized to build and package internal/external executable, version-controlled software releases. We have found many diverse automated software process pipelines are used across and sometimes within software integration activities (Scacchi & Alspaugh, 2013b). These pipelines take in OSS code files, dependent libraries, or repositories (e.g., GitHub) and build executable version instances that are then subjected to automated testing regimes that include simple “smoke tests” and extensive regression testing. Successful builds eventually turn into packaged releases that may or not be externally distributed and deployed as ready-to-install executables. While this all seems modest and tractable, when one sees the dozens of different OSS tools used in different combinations across different target platforms it becomes clear that what is simple when small becomes a complex SE activity when the scale of deployment increases.

Another complication, which is now beginning to be recognized within and across BRD processes and process automation pipelines, arises in determining when and how different BRD tool chain versions/configurations can mediate cybersecurity requirements in the target system being built. We have seen cases in which software builds and deployed releases are assumed to integrate to functionally equivalent CSS components, but which are then not included in releases due to IP restrictions. We have also observed and reported how functionally equivalent variants as well as functionally similar versions may or may not be produced by BRD tool chains, either by choice or by unintentional consequence. This, in our opinion, gives rise to the need for explicit open-source models of BRD process automation pipelines that can be analyzed, tested, reused, and shared to determine whether release versions/variants can be verified and/or validated to produce equivalent/similar releases that preserve prior cybersecurity obligations and usage rights.

Last, mixing new OSS and CSS components with legacy apps wrapped within widgets will complicate build and release processes and obscure deployment processes. Legacy apps encapsulated within mission-specific widgets will commonly need to dynamically link executable binary components, which in turn increases the challenges in their testing and cybersecurity assurance, both during development and during field deployment. In order to mitigate these technical challenges while enabling more agile software component system integration, multi-component OA configurations should explicitly declare pre/post conditions on acceptable input/output parameter values, along with exceptional values, that in turn can be independently verified or validated.

Software Component Evolution Practices Transmitted Across the OA Ecosystem

Software evolution is among the most-studied of SE processes. While formerly labeled as "software maintenance," a profitable activity mediated through maintenance contracts from software producers to customers, the experience of OSS development projects and practices suggest a transition to a world of continuous software development—one that foreshadows the emergence of continuous SE processes, or software life cycles that just keep cycling until interest falters or spins off into other projects. OSS development projects rely on OSS tools that themselves are subject to ongoing development, improvement, and extension, as are the software platforms, libraries, code-sharing repositories, and end-user applications utilized by OSS developers to support their development work. Developers entering, progressing, or migrating within/across OSS projects further diversify the continuous development of the most successful and widely used OSS components/apps. This dynamism in turn produces many ways for OSS systems or OA systems that incorporate OSS components to evolve.

Figure 5 portrays different software evolution patterns, paths, and practices we have observed arising with new C3CB applications (Scacchi and Alspaugh 2012). Here we see paths from a currently deployed, executable system release, to a new deployed release—something most of us now accept as routine as software updates are propagated across the Internet from producers, through integrators, to customers and end-users.

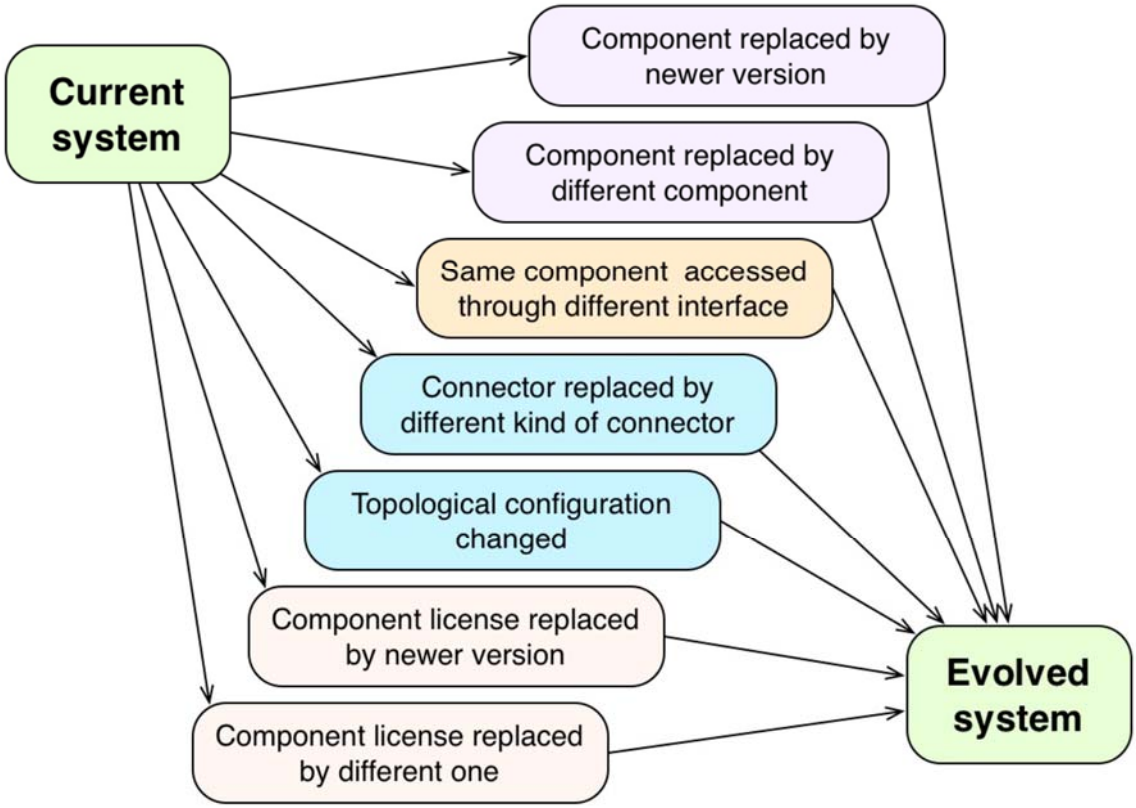


Figure 5. Different Paths and Mechanisms Through Which OA Software Systems Can Evolve
(Scacchi & Alspaugh, 2012)

Integrated OA systems can evolve through upgrades of functionally equivalent component variants (patches) as well as through substitution of functionally similar software components sourced from other producers or integrators. In Figure 6, we show a generic situation that entails identifying how an OA consistent with that depicted in Figure 2 may accommodate the substitution and replacement of a locally installed word processor application with a remote Web-based word processing software services (for example, Google Docs or Microsoft Office 365). This capability is a result of utilizing an OA that constitutes a reference model aligned with a vendor-neutral software product line. This is also a capability sought by customer organizations, and sometimes encouraged by software producers to accommodate their evolving business models (discussed below). While the OA remains constant, the location of the component has moved from local to remote/virtual, as has its evolutionary path. Similarly, the cybersecurity of the local versus remote component has changed in ways that are unclear, and entail a different, evolved assurance scheme.

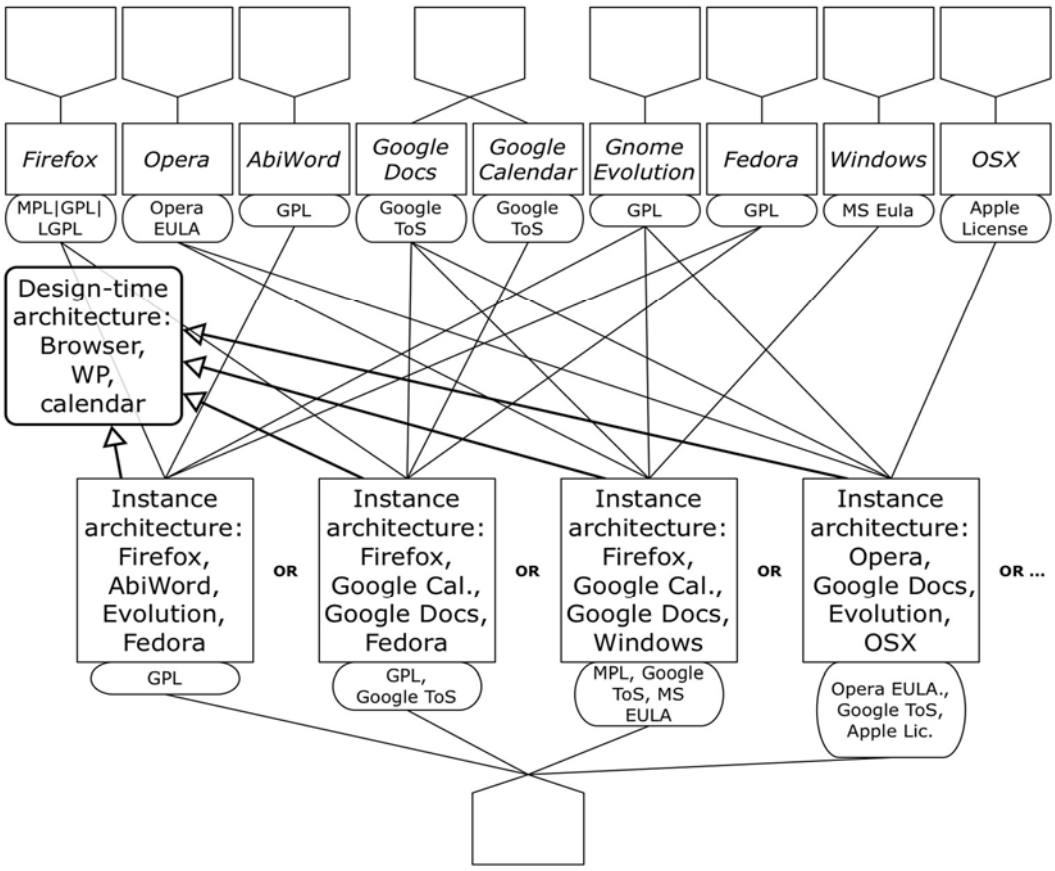


Figure 6. Alternative Configurations of Integrated Instance Releases of Components Consistent With the OA in Figure 2 That Are Treated as Functionally Equivalent by Customer Organizations
(Scacchi & Alspaugh, 2012)

Next, any common development technology used to support production or integration of mission components with shared infrastructure components must recognize that these technologies and components are all subject to independent, mostly autonomous evolution practices within the Defense community. For example, OZP is currently undergoing evolution, including its migration to Java 8 sourced by Oracle, and this move will

may disrupt the correct operation of widgets already produced using Java 7 common development technologies. Similarly, new OSS and CSS components will evolve due to practices arising in the competitive marketplace, while legacy mission components wrapped within widgets will have obscure or opaque evolution practices that are locked into legacy Defense community component providers. Legacy components will also limit how their encapsulating widgets evolve, potentially due to architectural mismatches or dependencies to legacy systems that are no longer supported, operational, or compatible with current platform technologies (Velasco-Elizondo et al., 2013).

Overall, the evolution of software components, component licenses, component interconnects and interconnections, and interconnected component or AC configurations are now issues that call for research efforts to help make such patterns, paths, and practices more transparent, tractable, manageable, and scalable within an OA software ecosystem, as well as customers seeking the benefits of openness, sharing, and reuse.

New Business Models for Acquisition of Software Components and Widgets

The last issue we address is the newest in this set of six for consideration for new acquisition research. While the field of acquisition research and practice has long paid attention to software economics, the challenges of software cost estimation are evolving in light of new business models being put into practice by software producers and system integrators. In the past, software development projects were often managed by a single contractor responsible for both software production and system integration. Costs could be assessed through augmentation to internal business accounting practices (e.g., budgeting, staffing workloads, time-sheet reports, project schedules, etc.). But a move to OA ecosystems means that multiple producers can participate, and OA schemes accommodate switching among providers while a system is being integrated, deployed, or evolved in the field. This in turn coincides with new ways and means to electronically distribute software updates, components, or applications, as well as new ways to charge for software. OSS components may be acquired and distributed at “no cost,” but their integration and evolution are charged as service subscription, or as time-effort billings.

We have already seen other alternatives for costing or charging for software that include franchising; enterprise licensing; metered usage; advertising supported; subscription; free component, paid service/support fees; federated reciprocity for shared development; collaborative buying; donation; sponsorship; free/open source software (e.g., Government OSS—GOSS); and others. So how are customer organizations, especially in the Defense community where software cost estimation practices are routine, supposed to estimate the development or sustaining costs of the software components or integrated systems they acquire and evolve, especially when an OA system allows for producers whose components come with different costing/billing schemes? This is an open problem for both acquisition research and software engineering practice.

Overall, new OSS and CSS components are experiencing a rapid diversification of acquisition cost models and practices, while legacy components are generally tied to single-source contractors as a result of utilizing legacy components as a cost-avoidance practice. All of the preceding five factors further obfuscate how to estimate or measure software component/AC development costs, schedules, or time to delivery/usage. So acquisition costs of systems that mix and match new OSS and bespoke CSS components, together with legacy CSS components, will be difficult to cost-estimate or cost-manage. This in turn will limit the efficacy of BBP 3.0 practices for such systems.

Discussion and Conclusions

Our study reported in this paper identifies a set of technical issues and risks that can dilute the cost-effectiveness of Better Buying Power efforts. It similarly suggests that current acquisition practices aligned with BBP can also give rise to acquisition management activities that can dominate and overwhelm the costs of OA system development. This adverse condition can arise through app/widget vetting, new software business models, opaque and/or underspecified acquisition management processes, and the evolving interactions of new software development and deployment techniques. Unless proactive investment in acquisition research and development can give rise to worked examples, open-source models, and new acquisition management system technologies, the likelihood of acquisition management dominating agile development and adaptive deployment of component-based OA C2 system capabilities is unsettling.

Our research identified and analyzed how new software component technologies like OSS infrastructure components, common development technology components, and mission-specific widgets for Web-based and/or mobile devices, along with their intellectual property (IP) license and cybersecurity requirements, engineering and evolution processes, and cost estimating practices interact to drive down (or drive up) total system costs across the system acquisition life cycle. The availability of such new scientific knowledge and technological practices can give rise to more effective expenditures of public funds and improve the effectiveness of future software-intensive systems used in government and industry. Thus, a goal of this paper was to explore new ways and means for achieving cost-sensitive acquisition of OA software systems, as well as identifying factors that can further decrease or increase the costs of such systems.

We identified and examined six areas for research arising at the intersection of software engineering and acquisition that now confront the Defense community (and perhaps other industries as well). These six issues areas include (1) the lack of architecture representations and schemes for discovering or specifying OA system designs; (2) OA systems that integrate components or applications subject to diverse, heterogeneous IP licenses; (3) how to manage the cybersecurity of OA systems during system design, development, and deployment; (4) software process challenges and evolving disruptions in seemingly mundane process automation pipelines; (5) software evolution patterns, path, and practices in OA ecosystems; and (6) how new business models are upending software cost estimation practices and outcomes. All of these research areas are readily approachable, and research results are likely to have significant practical value, both within the Defense community and beyond.

These issue areas were investigated and addressed in the domain of command, control, communication, cyber and business systems (C3CB). We believe all are tractable, yet dense and sufficient for deep sustained research study, as well as for applied research in search of near-term to mid-term practical results.

In related work (Scacchi & Alspaugh, 2015), we have called for specific R&D investments into the development of open source, domain-specific languages for specifying open architecture representations (or architectural description languages) that are formalizable and computational, as well as supporting annotations for software license obligations and rights. While ADLs have been explored in the SE research community, the challenges of how software architectures mediate software component licenses and cyber security requirements are an open issue, with practical consequences. Similarly, ADL annotations that assign costs or cost models in line with new software business models are an open problem area. We have also called for R&D investment in new SE tools or support environments who purpose is to provide automated analysis and support of OA systems IP

and cybersecurity obligations and rights, as new requirements for industrial practice in large-scale software acquisition, design, development, deployment, and evolution. Such environments are the automated tools that could be used to model, specify, and analyze dynamically configurable, component-based OA software systems expressed using the open source architectural representation schemes or ADLs noted here.

Our research identifies and analyzes how OA CBC3 system capabilities can utilize software components and mission-specific widgets, with diverse IP license and cybersecurity requirements, and how new software business models can interact to affect total system costs across the system acquisition life cycle. The availability of such new scientific knowledge and technological practices can give rise to more effective expenditures of public funds and improve the effectiveness of future software-intensive systems used in Defense community, as well as elsewhere within government and industry. Hopefully, this paper serves to help throw light into how software engineering and acquisition research can inform and add benefit to software practices within the Defense community through ways and means that further advance Better Buying Power opportunities and outcomes.

References

Alspaugh, T. A., & Scacchi, W. (2012, September). Security licensing. In *Proceedings of the Fifth International Workshop on Requirements Engineering and Law* (pp. 25–28).

Alspaugh, T. A., & Scacchi, W. (2013). Ongoing software development without classical requirements. In Proceedings of the 21st *IEEE International Conference of Requirements Engineering* (pp.165–174). Rio de Janeiro, Brazil.

Alspaugh, T. A., Scacchi, W., & Asuncion, H. (2010). Software licenses in context: The challenge of heterogeneously licensed systems. *Journal of the Association for Information Systems, 11*(11), 730–755.

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

Bollinger, T. (2003, January 2). Us*e of free and open-source software (FOSS) in the U.S.* Department of Defense. MITRE.

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

Conley, K., Brockman, B., Diercks, P., George, A., Lam, W., Lozano, A., Palnter, R. & Tolentino, G. (2014, June). Achieving information dominance: Unleashing the Ozone Widget Framework. In *Proceedings of the 19th International Conference of Command and Control Research & Technology Symposium* (ICCRTS; Paper 109). Arlington, VA.

Garlan, D., Dwivedi, V., Ruchkin, I., & Schmerl, B. (2012). Foundations and tools for end-user architecting. In D. Garlan and R. Calinescu (Eds*.), Large-scale complex IT systems. Development, operation and management, Lecture Notes in Computer Science* (pp. 157–182), Springer, 7539.

George, A., Galdorisi, G., Morris, M., & O'Neil, M. (2014, June). DoD application store: Enabling C2 agility. In *Proceedings of the 19th International Command and Control Research and Technology Symposium* (Paper-104). Alexandria, VA.

George, A., Morris, M., Galdorisi, G., Raney, C., Bowers, A., & Yetman, C. (2013, June). Mission composable C3 in DIL information environments using widgets and app stores. In *Proceedings of the 18th International Command and Control Research and Technology Symposium* (Paper-036). Alexandria, VA.

George, A., Morris, M., & O'Neil, M. (2014). Pushing a big rock up a steep hill: Lessons learned from DoD applications storefront. In *Proceedings of the 11th Annual Acquisition*

Research Symposium (Vol. 1; pp. 306–317). Monterey, CA: Naval Postgraduate School.

Guertin, N. H., Sweeney, R., & Schmidt, D. C. (2015, May). How the Navy can use open systems architecture to revolutionize capability acquisition: The naval OSA strategy can yield multiple benefits (NPS-AM-15-004). In *Proceedings of the 12th Annual Acquisition Research Symposium.* Monterey, CA: Naval Postgraduate School.

Kazman, R., & Carriere, S. J. (1998). Playing detective: Reconstructing software architecture from available evidence. *Journal of Automated Software Engineering, 6*(2), 107–138.

Kendall, F. (2015, April 9). *Implementation directive for Better Buying Power 3.0* [Memorandum].

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

Reed, H., Benito, P., Collens, J., & Stein, F. (2012, June). Supporting agile C2 with an agile and adaptive IT ecosystem. In *17th International Command and Control Research and Technology Symposium* (ICCRTS; Paper-044). Fairfax, VA.

Reed, H., Nankervis, J., Cochran, J., Parekh, R., & Stein, F. (2014, June). Agile, adaptive IT ecosystem: Results, outlook, and recommendations. In *Proceedings of the 19th International Command and Control Research and Technology Symposium* (ICCRTS; Paper-011). Arlington, VA.

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

Scacchi, W. (2009). Understanding requirements for open source software. In K. Lyytinen, P. Loucopoulos, J. Mylopoulos, & W. Robinson (Eds.), *Design requirements engineering: A ten-year perspective* (LNBIP 14; pp. 467–494). Springer-Verlag.

Scacchi, W. (2010). The future of research in free/open source software development. In *Proceedings of the ACM Workshop on the Future of Software Engineering Research* (FoSER; pp. 315–319), Santa Fe, NM.

Scacchi, W., & Alspaugh, T. (2012, July). Understanding the role of licenses and evolution in open architecture software ecosystems. *Journal of Systems and Software, 85*(7), 1479–1494.

Scacchi, W., & Alspaugh, T. (2013a, May). Streamlining the process of acquiring secure open architecture software systems. In *Proceedings of the 10th Annual Acquisition Research Symposium* (pp. 608–623). Monterey, CA: Naval Postgraduate Schoool.

Scacchi, W., & Alspaugh, T. (2013b, May). Processes in securing open architecture software systems. In *Proceedings of the 2013 International Conference Software and System Processes* (pp. 126–135). San Francisco, CA.

Scacchi, W., & Alspaugh, T. (2013c, June). Challenges in the development and evolution of secure open architecture command and control systems. In *Proceedings of the 18th International Command and Control Research and Technology Symposium* (Paper-098). Alexandria, VA.

Scacchi, W., & Alspaugh, T.A. (2013d). Advances in the acquisition of secure systems based on open architectures. In *Journal of Cybersecurity & Information Systems, 1*(2), 2–16.

Scacchi, W., & Alspaugh, T. (2015, May). Achieving Better Buying Power through acquisition of open architecture software systems for web and mobile devices (NPS-

AM-15-005). In *Proceedings of the12th Annual Acquisition Research Symposium.* Monterey, CA: Naval Postgraduate School.

Software architecture. (n.d.). In *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Software_architecture

Velasco-Elizondo, P., Dwivedi, V., Garlan, D., Schmerl, B., & Fernandes, J. M. (2013). Resolving data mismatches in end-user compositions. *End-user development*. Springer Berlin Heidelberg, 120–136.

Womble, B., Schmidt, W., Arendt, M., & Fain, T. (2011). Delivering savings with open architecture and product lines. In *Proceedings of the 8th Acquisition Research Symposium* (pp. 8–13). Monterey, CA: Naval Postgraduate School.

Acknowledgments

This report was supported by grant #N00244-16-1-004 from the Acquisition Research Program at the Naval Postgraduate School, Monterey, CA. No endorsement, review, or approval implied. This paper reflects the views and opinions of the authors, and not necessarily the views or positions of any other persons, group, enterprise, or government agency.

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

www.acquisitionresearch.net

Achieving Better Buying Power for Mobile Open Architecture Software Systems Through Diverse Acquisition Scenarios

Walt Scacchi and Thomas Alspaugh

INSTITUTE *for* SOFTWARE RESEARCH
UNIVERSITY *of* CALIFORNIA · IRVINE

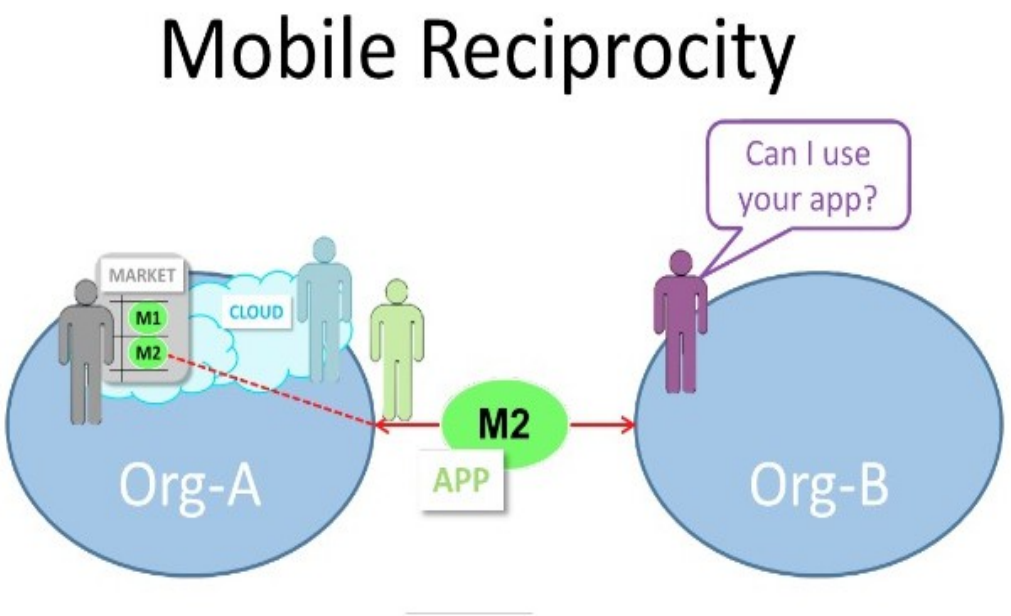
Overview

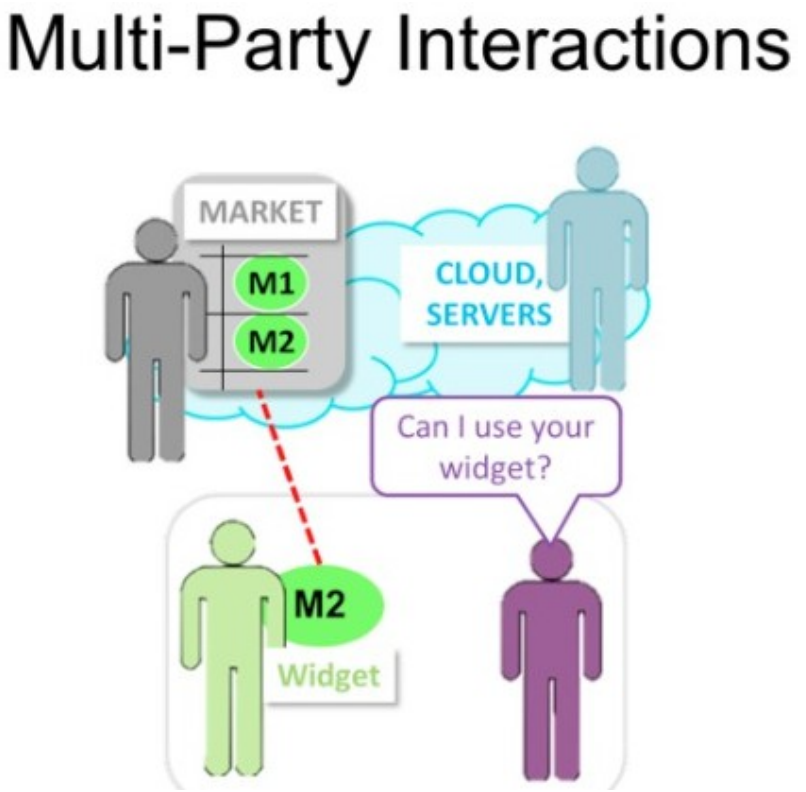
- Background
- *Case Study*: Multi-party acquisition of components for a secure Open Architecture C2 systems within an agile, adaptive software ecosystem
- Emerging R&D challenges in acquiring secure, component-based OA C2 systems
- Emerging challenges in achieving Better Buying Power via component-based OA systems
- Conclusions

Background

- New ways and means for acquisition, development, and deployment of C2/C3CB systems.
  - Development and deployment of *assembled capabilities* (AC) across the Defense open architecture (OA) software ecosystem
- Who is pursuing AC for C2/C3BC system capabilities?

Transforming to multi-party acquisition of software elements within OA ecosystems





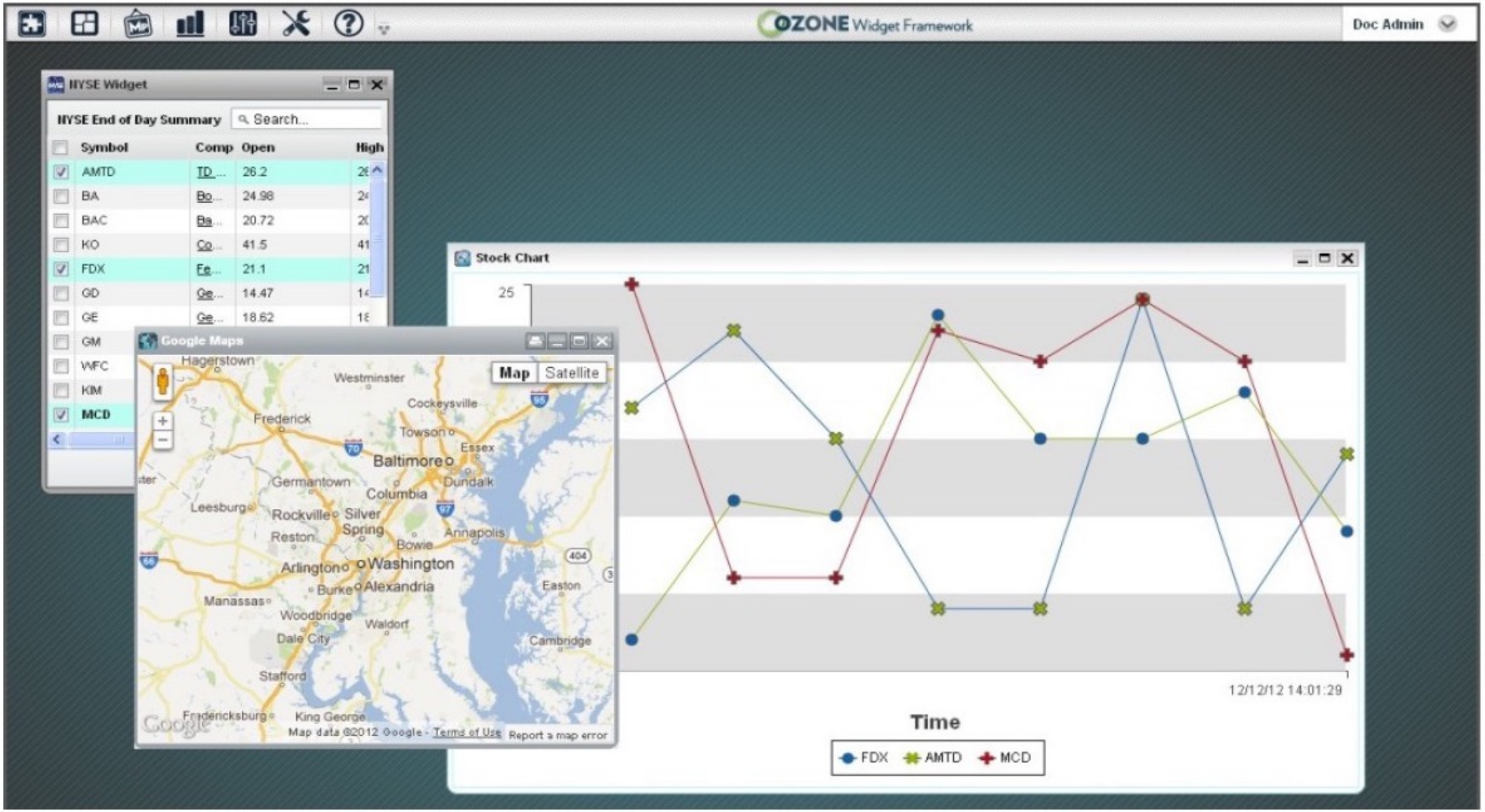
Customer/end-user organizations now looking for ways to reduce acquisition cost and effort through *shared development/use of common OA software system components* (apps, widgets).

C3CB Software Component Types

- *Mission Component*s enable C3CB processes and present common operating picture data to end-users.
  - Mission components realized as apps/widgets that may be deployed on mission-specific platforms including secured Web/mobile devices.
- *Common Development Technology Components* provide AC development tools and common run-time applications servers that support the mission components, where these servers are bundled with Shared Infrastructure.
- S*hared Infrastructure Components* combine local/remote application servers and data repositories with networking services and deployment platforms.

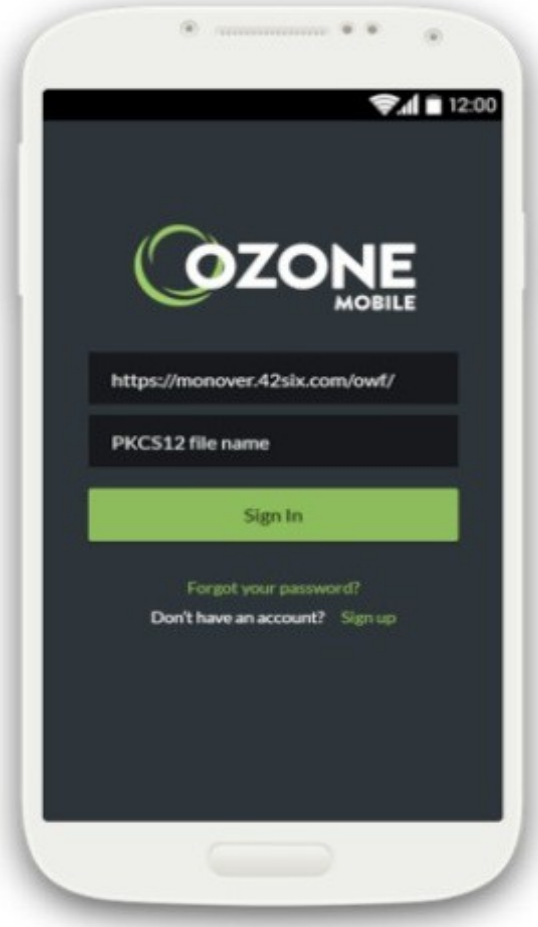
Sample of producers for mission components, common technologies, infrastructure components

New paths for software component acquisition and development using inter-communicating widgets/apps acquired from online App Stores



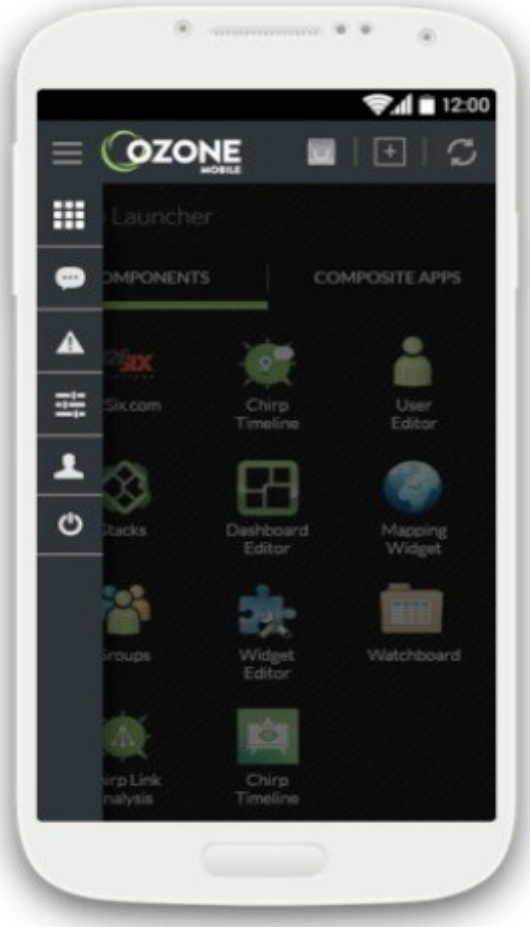
Shared development of Apps and Widgets as OA system components

CAS Sign In



App Launcher

Ozone Mobile Drawer Menu



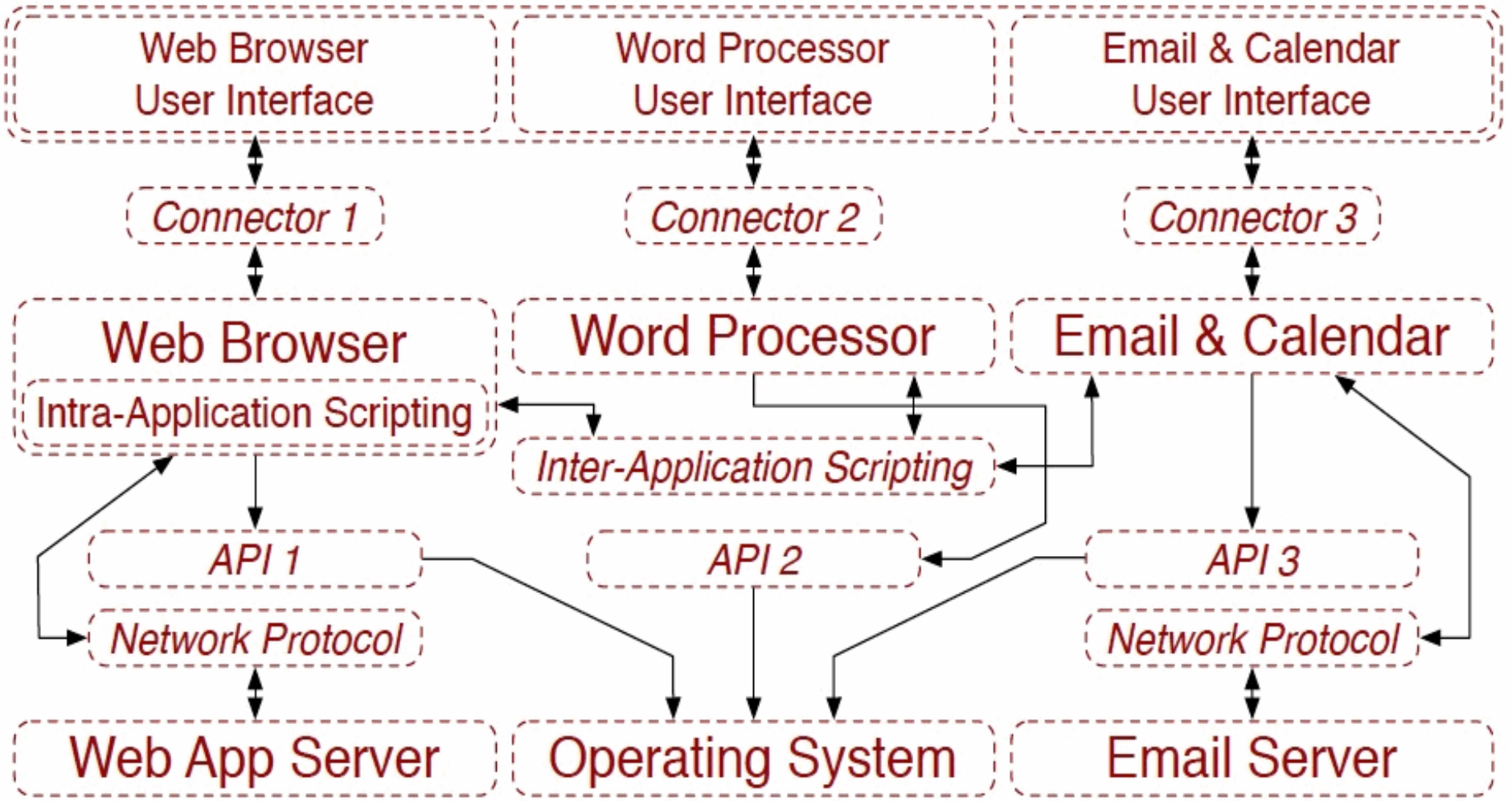
Ozone Platform for Mobile Devices

Who is pursuing AC for C2/C3BC systems?

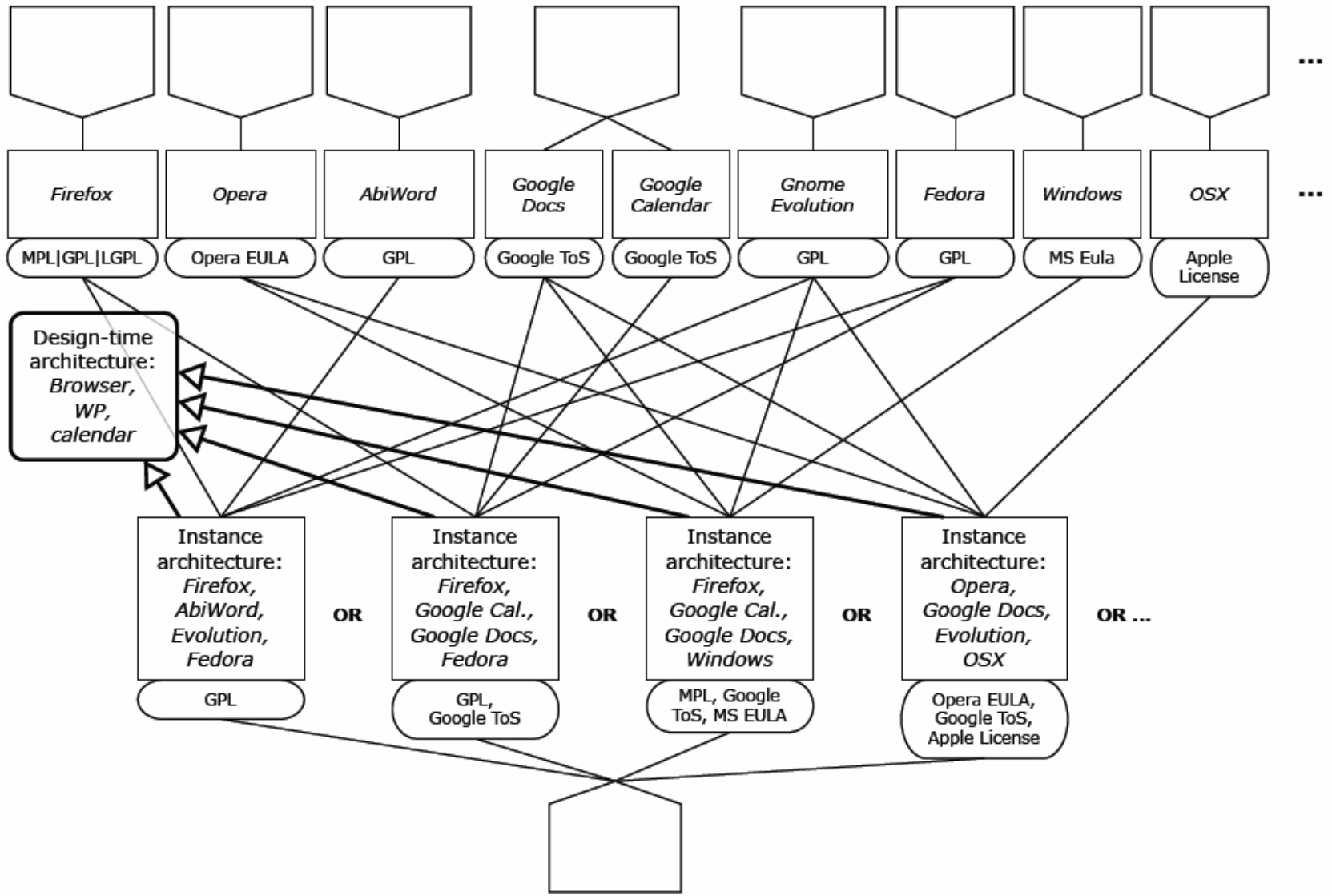
- OUSD (AT+L), DASD(A)-C3CB Working Group
- Air Force – TBMCS-FL (manages ATOs, manages Airspace)
- Air Force – AOC (Air Operations Center, using harvested components from TBMCS-FL, and CANES)
- Army – DCGS-A, DIB (DCGS Integration Backbone), and DMO (DIB Management Office)
- Navy – CANES and ACS (Afloat Core Services)
- Navy – PEO C4ISR Storefront and Tactical Cloud Marketplace
- DI2E

Case Study: OSS, open architectures, and software licenses for C2 or C3CB systems

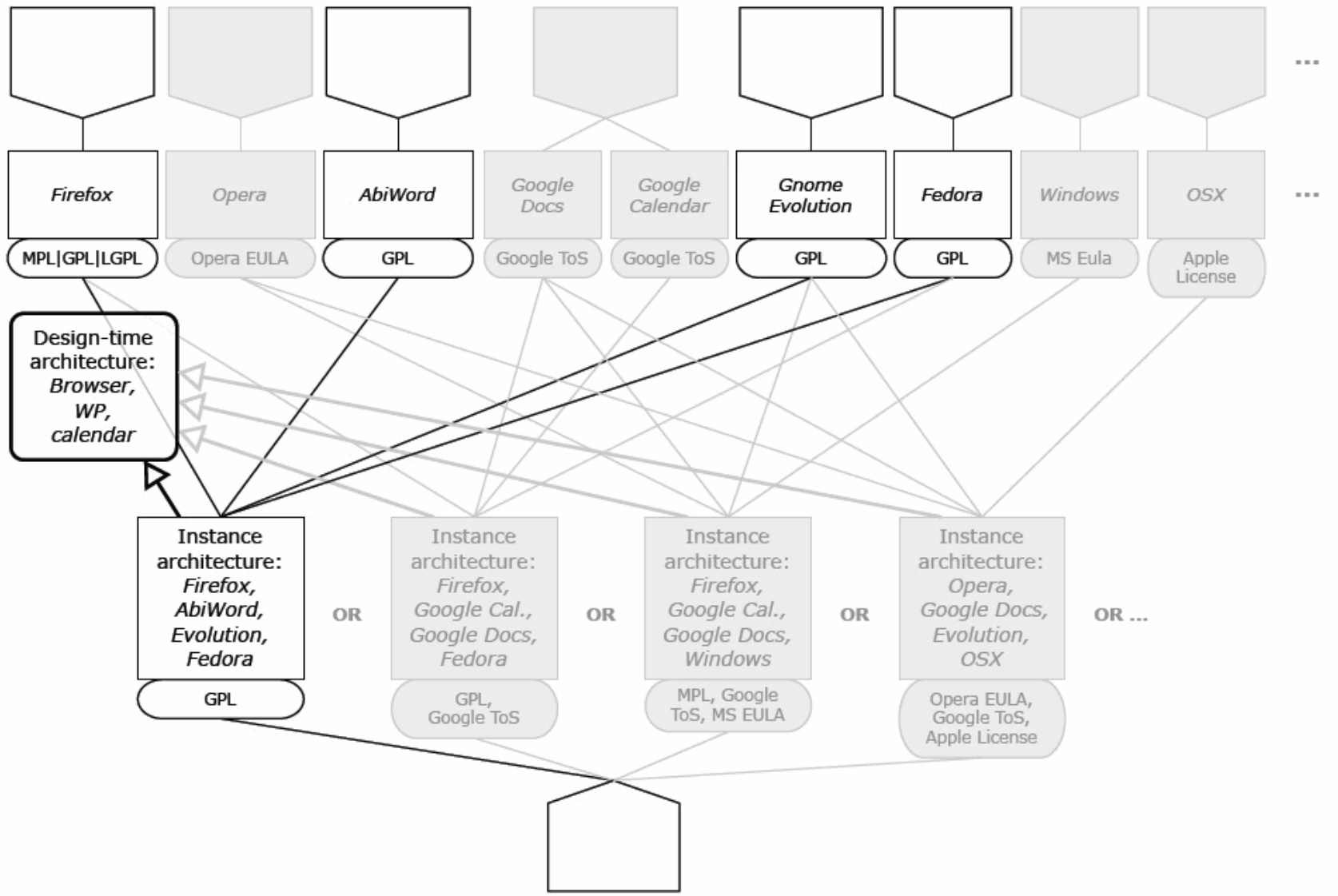
Design-time view of an OA system



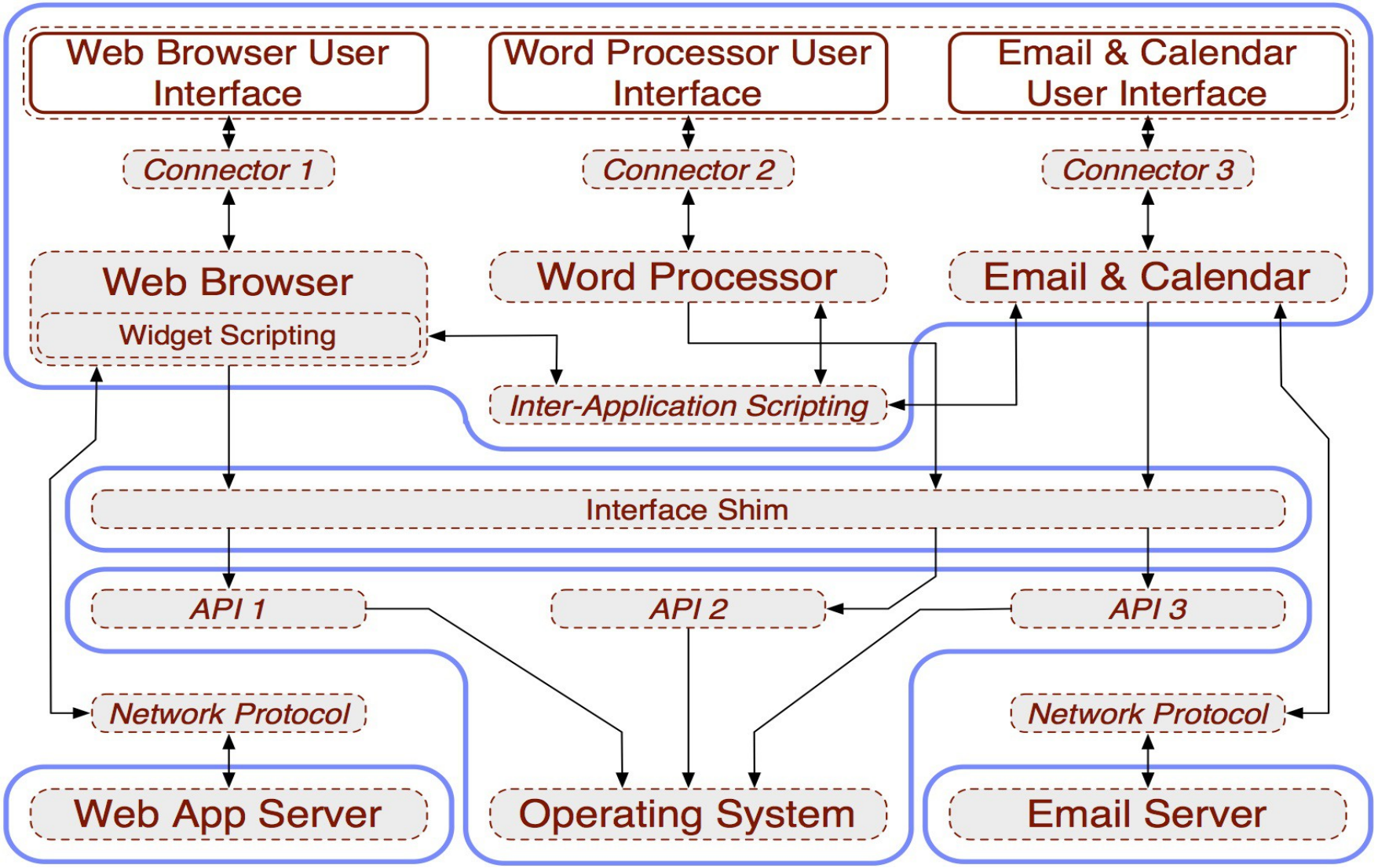
Software product line of *functionally similar* OA system alternatives



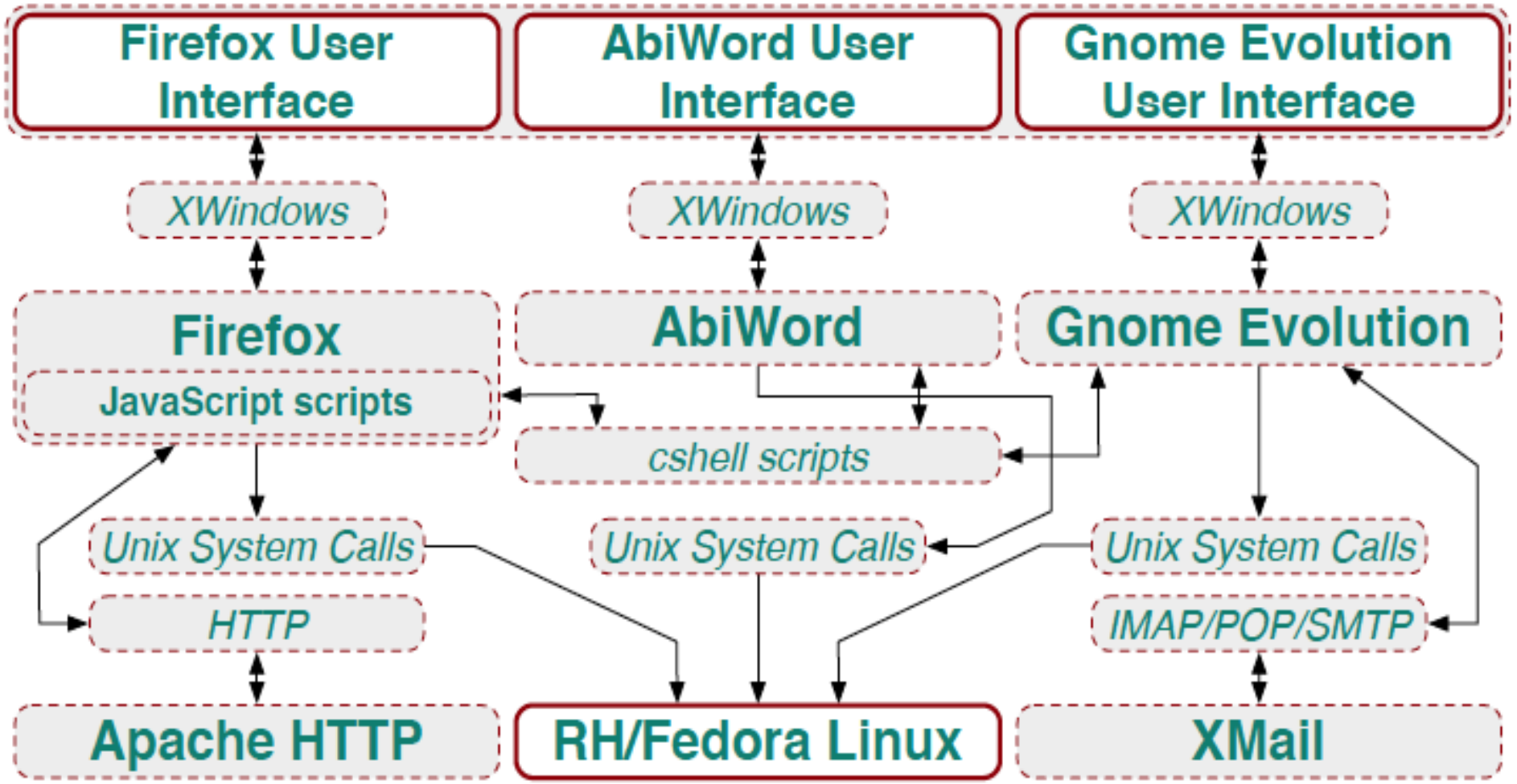
Product line selection of one alternative system configuration



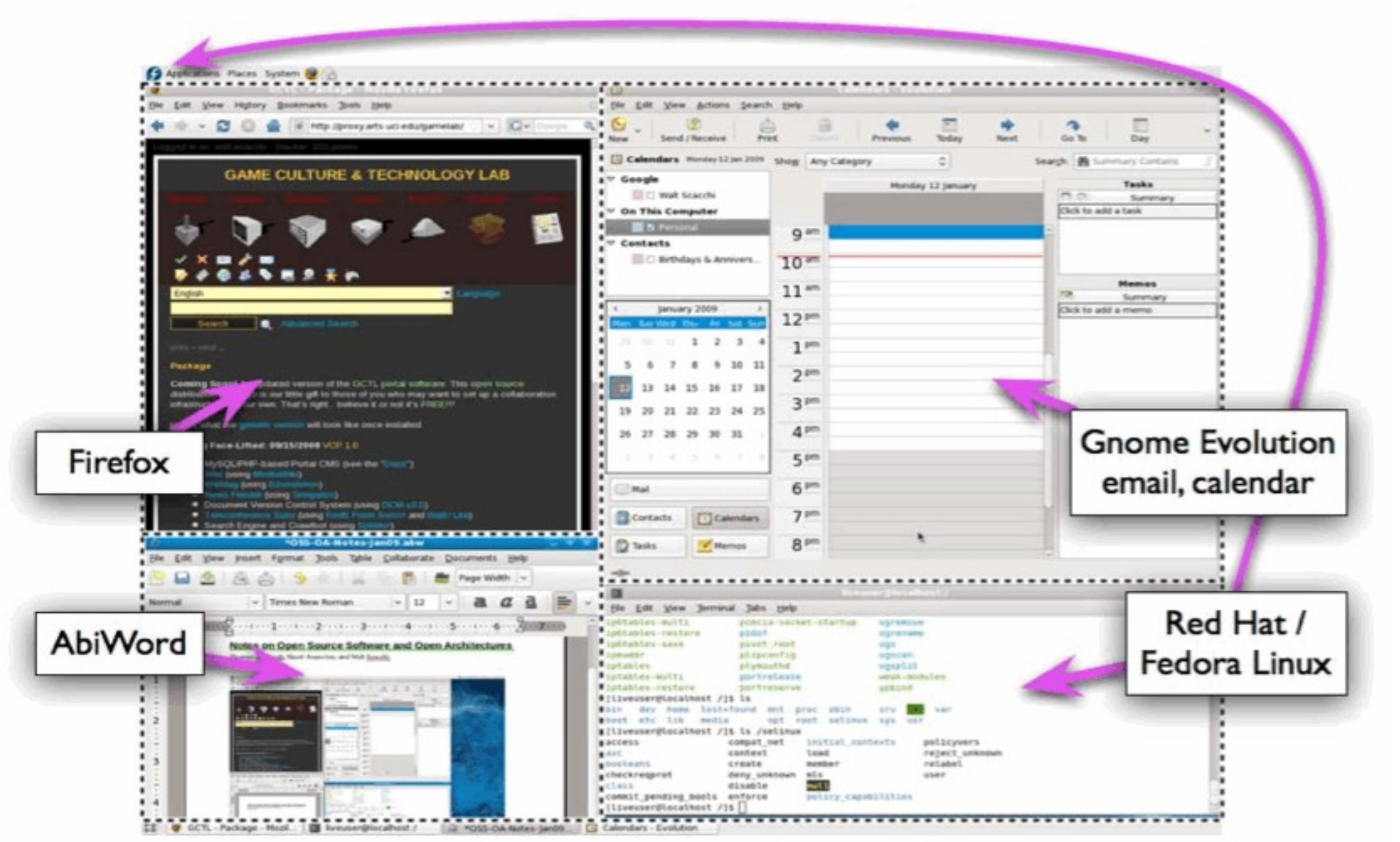
A security capability specification encapsulating the *design-time* configuration via multiple virtual machine containers



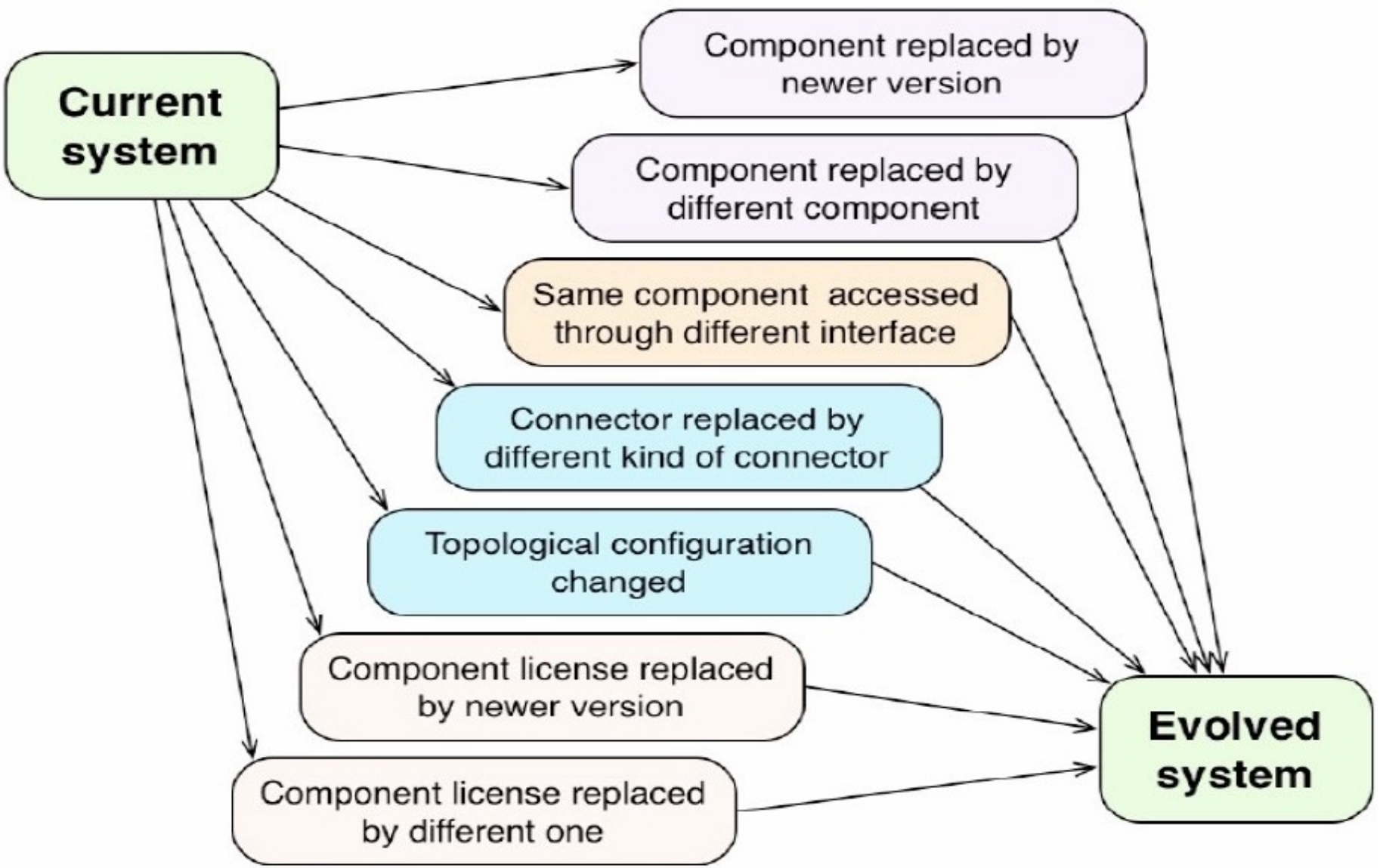
Build-time view of OA design selecting *OSS* product family alternatives



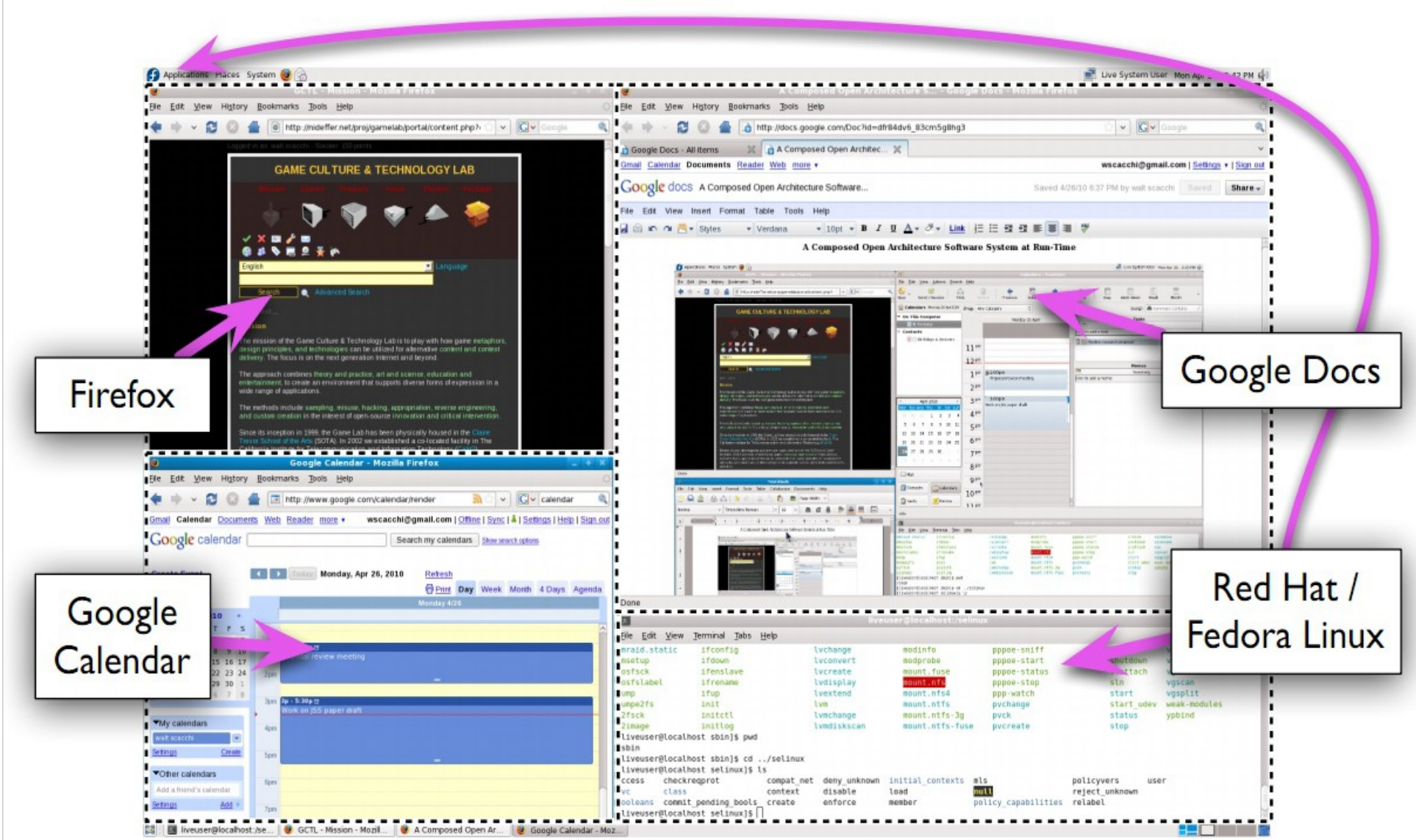
Run-time deployment view of OA system family member configuration



Evolution-time software changes



Evolved run-time deployment view of a *functionally similar* alternative OA system configuration



Challenges of securing open OA C2/C3CB systems

Current security approaches

- Mandatory access control lists, firewalls;
- Multi-level security;
- Authentication (including certificate authority and passwords);
- Cryptographic support (including public key certificates);
- Encapsulation (including virtualization), hardware confinement (memory, storage, and external device isolation), and type enforcement capabilities;
- Secure programming practices;
- Data content or control signal flow logging/auditing;
- Honey-pots, traps, sink-holes;
- Security technical information guides (STIGs) for configuring the security parameters for applications and operating systems;
- Functionally equivalent but diverse multi-variant software executables.
- Software component security assurance processes.

Current approaches to software cybersecurity do not address the challenges of continuously evolving OA C2 systems emerging within agile, adaptive software ecosystems!

New business/pricing models for OA software components

- Franchising
- Enterprise licensing
- Metered usage
- Advertising supported
- Subscription
- Free component, paid service fees
- Federated reciprocity for shared development
- Collaborative buying
- Donation
- Sponsorship
- (Government) open source software
- and others

Managing acquisition costs will be demanding. Acquisition workforce will need automated assistance*, else acquisition management costs will dominate development costs for OA software components!*

New practices to realize cost-effective acquisition of OA AC systems

- Need to R&D ***worked examples*** of reference OA system models, and component evolution alternatives.
- Need ***open source models of*** app/widget security assurance ***processes and*** reusable cybersecurity ***requirements.***
- Need precise ***domain-specific languages*** (DSLs) and ***automated analysis tools*** for continuously assessing and continuously improving cybersecurity and IP requirements for OA C2 systems composed from apps/widgets.

Emerging challenges in achieving *Better Buying Power* via OA software systems

- Program managers/staff *may not understand* how software IP licenses affect OA system design, and vice-versa.
- Software IP and cybersecurity obligations and rights propagate across system development, deployment, and evolution activities *in ways not well understood* by system developers, integrators, end-users, or acquisition managers.
- *Failure to understand* software IP and cybersecurity obligations and rights propagation can reduce DoD buying power, increase software life cycle costs, and reduce competition.
- DoD and other Government agencies *would financially and administratively benefit* from engaging the development and deployment of an (open source) automated software obligations and rights management system for the acquisition workforce.

Conclusions

- Our research identifies how new software component technologies, IP and security requirements, and new business models interact to drive-down or drive-up acquisition costs.
- New technical risks for component-based OA software systems can dilute the cost-effectiveness of BBP efforts.
- Need R&D leading to automated systems that can model and analyze OA system IP licenses and cybersecurity requirements
  - Empower OA C2 system development workforce
  - Identify and manage cost-effectiveness trade-offs

Acknowledgements

Research collaborators

- Assembled Capabilities Working Group, DASD(A)-C3CB (2014-15).

Funding support ***(No endorsement, review, or approval implied)***.

- Naval Postgraduate School
  - Acquisition Research Program, #N00244-1-16-0004.

Thank you!

INSTITUTE *for* SOFTWARE RESEARCH
UNIVERSITY *of* CALIFORNIA • IRVINE