2014 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
SYSTEMS ENGINEERING (SE) TECHNICAL SESSION
AUGUST 12-14, 2014 – NOVI, MICHIGAN

PROPER USE OF OPEN STANDARDS FOR COMMONALITY AND COMPETITION

Mr. David Jedynak
Chief Technology Officer, COTS Solutions
Curtiss-Wright Defense Solutions
Austin, TX

ABSTRACT

This paper will describe layers of open standards, demonstrate the problems of standard top-down requirements flow-down and derivation, and show how the standard Analysis-of-Alternatives, as used at highest levels of Department of Defense acquisition, is appropriate for use at lower levels. Examples of how to (and not to) use open-standards in systems engineering specifications for best commonality and competition will be provided, intended for use as templates in specifications.

INTRODUCTION

Today's open standards, either developed specifically for embedded military commercial-off-the-shelf (COTS) applications or for adjacent markets such as the commercial High Performance Computing (HPC) market, provide today's rugged deployed military system designer with well-known, well supported of Open Systems Architecture (OSA)-based methods for addressing the full range of system specifications including Power, Environmental, Network, Thermal, and Physical. Increased use of well-established and broadly supported open standards enables the design and manufacture of subsystems using off-the-shelf building blocks that deliver today's most advanced computing technologies while reducing design risk, development schedule, and cost.

Following the Weapon Systems Acquisition Reform Act of 2009, the U.S. Government takes a much greater role in defining the high level technical requirements for new system designs. The updated acquisition process also strengthened the mandate to increase the use of OSA design strategies that, when applied effectively, enable the faster, simpler design of Lowest Price Technically Acceptable (LPTA) solutions to meet emerging capabilities requirements.

For the decision-makers now tasked with defining system requirements, the key to tapping the full potential of OSA requires knowledge of today's extensive eco-system of open-standard interfaces and hardware. By designing subsystem solutions with proven open standards-based building blocks, today's system integrators can achieve great cost savings and reduced deployment schedules.

One of the most common contributors to added cost in system design is over-specification, which may exclude or complicate the use of open-standard systems and components. COTS vendors who design solutions using open standards can help Government reduce costs and optimize system space, weight and Power (SWaP) by providing architectural input based on their extensive knowledge and expertise in the proper use of off-the-shelf building blocks to satisfy the most essential technical requirements. These standards enable a community of competitive suppliers to offer cost-effective off-the-shelf alternatives to proprietary, and frequently over specified, alternatives.

THE PROBLEM OF SPECIFICATIONS

Specifications are critical and essential to business and technology. This is not up for dispute, and the global transition of economies and capabilities to a highly interoperable and interchangeable set of businesses, resources, processes, and technologies is solid proof that the use of specifications is highly advantageous. Whether specifications are physical, software, financial, process, regulatory, they are of immense benefit *when applied correctly*.

The problem, however, is in the level of detail in the application of specifications. Whether too much detail (over-specification) or not enough detail (under-specification), inappropriate use can create significant risk, additional cots, and unwarranted constraints. The system / product designer (systems engineer) needs to appropriately communicate their intent – no more, no less – through the specifications.

To illustrate this point, take the simple example of a screw. If, for whatever system capability reason, the designer wants a system to be assembled with screws (as opposed to rivets, nails, welds, or adhesives, etc.), then it's expected to see a requirement something along the lines of "Only screws shall be used for unit construction." It's understandable to immediately ask "what type?" and to further specify the specific type of screw to be used. It's here that the specification problems arise.

Let's ignore the capability provided by using screws – perhaps it makes it easier to disassemble and cannibalize physical parts, perhaps it's an institutional policy – "screws are better than glue" – but it doesn't matter. What matters is the further intent of the system designer. Consider the following intent and specifications:

Intent:
Standardize Screws for best purchasing volumes, driving to lowest cost.

Under-specification:
"Use screws" doesn't communicate this intent, as it does not constrain the types to increase volume

Over-specification:
"Use Company X part number 123" also doesn't communicate the intent, but tries to cut to the implementation, with the assumption that Company X part number 123 will be the type with the best volume pricing.

In this case, the communicating intent via a proper specification depends on the recipient of the specification. If it's an internal design resource, the proper specification could be "Use screws selected from our company's high volume common screw list." If it's an external resource (3rd party), then the proper specification could be "Use screws selected from your company high volume screw list."

What's interesting here is that there's actually a potentially flawed assumption up front, which is "highest volumes equals lowest cost" which illustrates the danger of specifying screws when what you really want is a box.

The argument can be made that "a box" is not all you want. It may be that you want "a box which can be assembled and disassembled using the standard repair tool-kit", at which point, it would be fair to specify "use screws which are compatible with the standard repair tool-kit". Over-specification, in this case, would be to call out the detailed types of screws, provide drawings of the screws, or otherwise focus on aspects of the screw which are not specifically related to the interface between screw and tool (e.g. "Slotted screwdriver, blade widths of 3/16th through ½ inch"). When over specifying, the implementer of the design – generally considered the subject matter expert – is constrained from applying their domain knowledge and best practices to the implementation.

Screws are an easy example of improper use of specifications causing problems. Another more complex example is computer architectures and processors, which introduce a moving specification target due to constant improvements in capabilities. Consider the following intent and specifications:

Intent:
The system needs a computer processor which is capable of implementing system capability X.

Under-specification:
"The system shall contain a processor" Nothing is provided with regard to performance figures versus the need of capability X (e.g. processing capability, memory size, and data throughput).

Over-specification:
"The system shall contain one Intel 4th Generation Core i7 Model 4690S, with 2 Gigabytes of DDR3-1333 RAM, and one 100Mbit/s Ethernet port." The problem here is that the 4690S is a mid-range option out of about 100 model variants of the 4th Generation Core i7 processor which itself will be replaced by the more capable 5th Generation within 18 months (and so on), the RAM specification is a mid-range specification which precludes higher and lower performance and density variants, and the Ethernet port specification is actually slower than the standard 1Gigabit/s performance currently included in modern chipsets. This specification locks implementation to a very specific design configuration that may meet the performance needs today, but constrains everything else severely, potentially to a highly non-optimal design with regard to cost and performance.

In this case, proper specification boils down to specifying the expected computing needs of an algorithm, its memory requirements, and its data paths using industry standard performance benchmarks and standards to constrain the solution appropriately. For example, rather than specifying a particular processor models and architectures, specify computing performance using such standards as the various benchmarks published by the non-profit Standard Performance Evaluation Corporation (e.g. SPEC CPU2006), and requirements such as "1.5 Gigabytes of RAM reserved for program executable and heap", and "80 Megabit/second of UDP data throughput with packets of 4096 bytes." With these sort of requirements, the implementer is free to select amongst multiple different computer architectures (e.g. Intel x86 or 64 bit, ARM, PowerPC, etc.) as long as it can host the application and provide it with the appropriate resources.

A typical argument is that perhaps the specific computer architecture is needed because the legacy algorithms make use of specific hardware acceleration features (e.g. AltiVec in PowerPC), and that, as a result, the same architecture is required moving forward. The problem with this is that when the algorithm was written, the hardware acceleration features were used to achieve algorithm performance levels, but a more modern, faster processor may be able to outpace, in software alone, the older architecture with hardware acceleration. This again illustrates the danger of just specifying a specific architecture when actually a performance level is the real concern.

What's underlying both of these examples is the concept of layers and interfaces. At some point, building blocks are assembled, via interfaces, and those assembled building blocks become the foundation for additional layers, and interfaces exist between those layers of additional building blocks.

What matters is the performance of the building blocks, and the interfaces to them, and specifications need to stay focused purely on these aspects, rather than the specific implementations. It's with this that open standards have significant impact on commonality and competition.

LAYERS OF OPEN STANDARDS

Ultimately, everything is a mix of interfaces and building blocks, whether physical or virtual. When presenting layers and interfaces, a fundamental question is the context and perspective – what's the bottom layer? What's the top layer? The specific subject domain (e.g. defense) helps define those layers and interfaces. Figure 1 present a framework for definitions specifically focused on the most complex systems – mixed hardware and software.

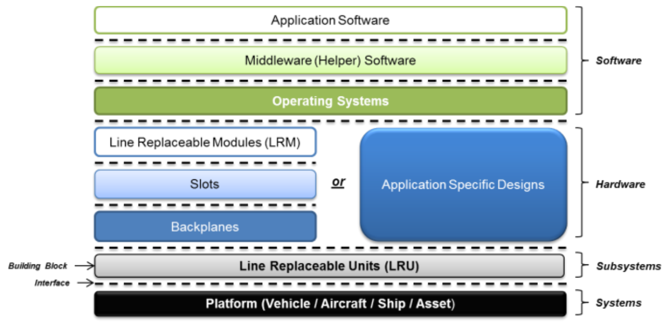
Application Software
Middleware (Helper) Software
Operating Systems
Software
Line Replaceable Modules (LRM)
Slots
or
Application Specific Designs
Backplanes
Hardware
Building Block
Interface
Line Replaceable Units (LRU)
Subsystems
Platform (Vehicle / Aircraft / Ship / Asset)
Systems

Figure 1: Layers of Open Standards

From an acquisition and typical requirements management perspective, the platform sits on top, with requirements flowing down. In this case, however, it's important to break that perspective and recast into performance / capability layers, with higher layers of capability depending on the performance of layers below. This forces a different perspective on use of specifications, and assists with the concept of using Analysis of Alternatives.

The sections below provide descriptions and insight to the various layers, their complexities, and where open standards can be of benefit.

The Platform

Although the highest level when it comes to acquisition and capability definitions (e.g. a tank, jet, destroyer, radar), it's far more instructive and useful to think of the platform as the foundation upon which to build capability.

The platform has specific performance characteristics which everything above / on it must fit (this is a recurring theme) in order to deliver the desired capability, usually thought of as top-down budgets, including physical size, weight, power, and cost to accommodate subsystems. Given that platforms are very unique and physical in nature, it's understandable that only a few open-standard interfaces exist for the platform interface, as shown in Table 1.

Table 1: Platform-Level Applicable Open Standards

Type of Interface	Applicable Open Standards	Notes
General Equipment Mounting	Screw / Bolt sizes (English and Metric)	Only the holes are standard, not the patterns of holes
Display Mounting	VESA standards	Standard display back 4-hole mounts
Commercial Rack Mount	Various from EIA, CEA, IEC	19 inch "relay rack" or "telco rack"
LRU Chassis sizes	Various ARINC Standards (ATR), VITA 58, VITA 75	This is really just a standardization of physical space and mounting
Power	MIL-STD-1275 / 704, Standard Single and 3-phase 110/220VAC 50/60Hz, 5V USB, 12V Automotive, Power-over-Ethernet	Noteworthy that all standards except MIL-STD-1275/704 have physical connector standards associated with them
Cabling	Various cable standards, wire gauge, standards such as EIA/TIA Cat-5 Twisted Pair, RG-6U Coax, etc.	Note that cabling standards may be tightly coupled with connector standards (e.g. Coax)
Connectors	MIL-STD-38999, USB, RJ-45 (IEC 8P8C), Coax, Various Power (as above)	Note that many of these connector types are tightly associated with the signal and cable types, except MIL-STD-38999

For the platform itself, there are a number of other standards which are more domain specific, e.g. standard fuel types, lubricants, ammunition types, battery sizes, wheel and track sizes, etc. What's interesting about these is that they are consumable / replaceable items, which recognize the benefits of commonality and competition quickly, as purchases and use occur on a regular basis. Many of the interfaces in Table 1 have been in significantly longer design / purchase / replace cycles, which means the benefits of commonality and competition have not been as readily apparent. This is an important historical distinction, and needs to be considered given the significant change in pace of technology / capability cycles in broader markets, and the need for significant up-tempo of capability updates in the defense market.

An office-building provides and good analogy for illustration of the benefits of well-defined open standard interfaces. The building provides standard interfaces, such as power and network wiring. This infrastructure serves a broad range of end-uses, allowing the use of the building to evolve and change over time and the end-users' desired purpose. It also decouples the specific technologies within the office building from the building itself. No one would ever buy a new office building just because they need to upgrade the printers, computers, and phones. In fact, those items don't need to be purchased with the building in the first place. As long as the necessary infrastructure is present, the technologies can move, evolve, change, or upgrade as desired.

Subsystem / Line Replaceable Units (LRUs)

Whether as a single LRU or a collection of associated LRUs, the platform is filled with various Subsystems which are intended to perform various functions to enable certain capabilities, e.g. provide satellite communications, stabilize a turret, or cook MREs. Regardless of the functions they perform and capabilities they provide, the LRUs are installed within the platform, which really means they must interface to the platform in accordance with particular specifications, while performing their intended functions to provide system level capabilities. To that end, the list of applicable Platform Open Standards is a significant part of the list of applicable LRU Open Standards. For example, power provided by the platform is power consumed by the LRU, and they must match.

Two critical points follow from this:

- The internal interfaces of a Subsystem are not necessarily dictated by the external Platform interfaces
- The capability of a Subsystem is not necessarily dictated by the Platform performance

It's these two points that are important to remember when applying open standards – they're not all-inclusive and comprehensive, often orthogonal, and more importantly, the use of an open standard does not preclude the use of proprietary interfaces *within* an externally open standard subsystem. Following from that, it's important to realize that a platform's performance (e.g. can provide only a maximum supply of 2 Amps of 28VDC MIL-STD-1275 power) allocated to a subsystem doesn't mean the subsystem should be considered a fixed capability over time, as LRUs from competing vendors or next generation LRUs may have significantly more capability than the baseline.

The corollaries to these points are:

- External Platform interfaces may constrain the set of applicable internal Subsystem interfaces
- Platform performance may constrain the current capability of a Subsystem

To the first point: a physical standard based on 10 inch long parts won't fit in a box that's limited to only 5 inches long. What's critical about the second point is the example above – the platform's performance (e.g. Power capacity) may limit the capability of a subsystem at the current time, but not necessarily tomorrow's more power-efficient implementation.

The take-away is that although interface and capability are interrelated, they aren't tightly correlated, having more of a "bracketing" relationship. Interface standards typically support a range of parameters (e.g. "up to 15 Amps", "up to 125MHz signaling", "from 6 to 128 pins"). Capability of a subsystem is more of a continuum which will eventually find boundaries at the edge of an interface range, for example, a limited amount of data processing capability because of the limits of the external data interface speeds, not the internal processing horsepower.

It's important to remember that increased capability may help drive interface requirements to a lower bracket. For example, more capable and efficient electronics mean lower power consumption for the desired capability, resulting in the applicability of another external interface which was not previously useable, such as lower capacity USB power or Power-over-Ethernet versus MIL-STD-1275 power.

This loosely-coupled relationship between interfaces and capabilities is important to using open standards, which are, by their nature, adaptable to a range of designs and applications, not just point designs.

LRUs have many different applicable open standards, beyond those of external platform interfaces. Some of these are shown in Table 2.

Table 2: Subsystem / LRU Applicable Open Standards

Type of Interface	Applicable Open Standards	Notes
General Mounting	Screw / Bolt sizes (English and Metric)	Just like the platform, not all hole patterns are standardized
Internal Circuit Board accommodation	Various standards, including VME, VPX, cPCI, PC-104, COMExpress, AT/X families, etc.	Many different standardized form-factors
Cooling provisions	Standardized fan sizes / mounting, IEEE standardized conduction cooling standards, other VITA standards for cooling mechanisms	Often these standards are a part of a form-factor standard, or called-out and referenced by those standards (e.g. VPX calls out IEEE cooling standards)
Power	Various standards based on various board form-factors like ATX power supply, VPX power supply, etc.	These standards typically collected a number of voltages, current ratings, noise parameters into standardized pin-outs, etc.
Data interfaces	Various interfaces, such as Ethernet, USB, SATA, Serial, Infiniband, Rapid I/O, etc.	Each of these is a mix of physical, data, and other network layers. Some include connector standards, but can be used without.
Internal Cabling	Various cable standards, wire gauge, standards such as EIA/TIA Cat-5 Twisted Pair, RG-6U Coax, etc.	Note that cabling standards may be tightly coupled with connector standards (e.g. Coax)
Connectors	Various connector types with different levels of performance, like VME, cPCI, VPX, xTCA, MXM, DIMM, etc.	Many of these are produced by single companies, but licensed under industry standard organizations to insure "openness"

The internals of a subsystem may or may not use various open standards. A box may have open standard external interfaces (e.g. MIL-STD-1275 power), but internally it could be an application specific point design (e.g. various application-specific power circuits) or it could leverage additional open standards (e.g. VPX standardized power supply to feed VPX standard modules in a VPX standard backplane). The important point here is that the particular internal implementation is isolated from the interfaces of the external box. Either way, the box does what it is supposed to do, and to the end-user, it's just a black box which does X, regardless of how it has actually been implemented.

Some LRUs utilize open standards internally, for multiple reasons such as modularity, serviceability, upgradeability, ease of development, etc. It's important to understand the concept involved in these, and some of the key standards available. For the defense industry, a number of standards have been used for over 30 years, including variants of various computer board standards like VME, Compact PCI, and most recently, VPX. The standards group, VITA, which governs VME and its replacement, VPX, is comprised of many companies who are actively engaged and supplying solutions in the defense industry, including component suppliers, prime contractors, and government funded university labs.

The family of VPX standards, including the subsystem-focused OpenVPX standard, defines a number of critical concepts useful in illustrating the concept of backplanes, slots, and Line Replaceable Modules. The backplane is either a passive or active electrical interconnect between multiple modules, with various connectors. OpenVPX defines backplanes according to:

- Form-factor / size of slots
- Types of slots
- Number of slots
- Topology (how everything is interconnected)
- Performance capabilities (e.g. maximum signaling bandwidth)

The backplane can be fully defined through the simple call-out of backplane specifications in accordance with the open standard. The backplane contains multiple slots, and each slot is defined with regard to various parameters, including:

- Form-factor / size of slots
- Connector types
- Common signals (e.g. Power, management, etc.)
- Slot type specific signal pin-out arrangements (e.g. number and groupings of differential signal pairs)
- Vendor or user-defined pin-out arrangements

The slots are connected to each other in accordance with the topology of the backplane specification. Note that the slots define pin-outs, but not the actual data types. Taken together, the backplane and slots are similar to the office building example at the platform level – it's the infrastructure, largely agnostic to what is being plugged into it as long as things fit within the general constraint brackets.

Line Replaceable Modules are intended to drop into the slot of the backplane. They may be of various types, such as storage modules, network switch modules, and processing modules. They have similar definitions to slots, but add:

- Cooling methodology (e.g. conduction cooled, air cooled, liquid cooled)
- Type of standardized data interfaces on the various signal pin-outs (e.g. Ethernet, PCI Express)
- Vendor-defined signals for vendor defined pins.

The "line replaceable" aspect of modules is actually part of the definition as well. Some modules may not require support for line replacement, instead requiring removal and replacement within a controlled maintenance environment. In the VPX family of standards example, VPX-REDI (VITA 48) specifically defines the provisions required on a module to make it line replaceable.

Operating Systems

Not all subsystems are electronic in nature, and not all of those have an element of programmability; however, the high availability of very small and low-cost microprocessors means that more and more subsystems on a platform will include some level of electronics, and that electronics module will include some level of programmability. For example, a pump may have a small controller built into it which monitors status, provides control, and performs diagnostics. As subsystems become more sophisticated, the presence of operating systems of one type or another will become increasingly common.

Operating systems, whether they are stereotypical full-fledged user operating systems like Windows, Linux, or VxWorks, or are little more than a handful of initialization, I/O, and interrupt routines along with a simple task management loop, provide one very significant task – they abstract the hardware so that application logic and algorithms can execute without direct and specialized interaction with the underlying hardware.

Like other layers, operating systems use a number of open standard interfaces to provide better interoperability, software portability, and lower risk development cycles. A number of the standardized interfaces are shown in Table 3.

Table 3: Operating System Applicable Open Standards

Type of Interface	Applicable Open Standards	Notes
Graphics Interfaces	OpenGL, DirectX (Windows), others	Allows complete abstraction of graphics processing hardware from graphics software
Audio Interfaces	ALSA (Linux), DirectX (Windows), others	Allows audio abstraction
Storage Interfaces	Various File system standards layered on top of various physical device interfaces (SATA, Flash, Network)	Allows abstraction and virtualization
Peripheral / Expansion Interfaces	USB and PCI/PCI Express including multiple device classes for different types of devices	Standard and vendor classes available
Network	Multiple standards for network access, including low-level Ethernet, various Internet-based protocols (TCP/UDP over IP, etc.)	Ubiquitous standards with significant maturity
Computing acceleration	OpenCL for heterogeneous computing	This is a development standard
General Operating System standards	POSIX, Sockets, etc.	Note that many of these are more than just things supported by operating systems, but also integral to the software development process

Middleware

The term "Middleware" has evolved, and continues to evolve. Regardless of a particular definition, Middleware is a "helper" layer of software which serves to simplify, abstract, connect, or otherwise help in the development, performance, and maintenance of applications. Depending on complexity of the desired functions, multiple middleware packages may be employed. In some cases, the middleware is full defined and does specific tasks, and in other cases, it may provide an agnostic framework for doing specific tasks. In practice, the middleware is usually embodied as a software library, or in some cases, it is a simply a network style service to communicate with.

One of the keys about middleware is that it can provide an abstraction between applications and the operating system itself, allowing an application written on one operating system to move easily to another as long as the middleware is implemented on both. Rather than presenting the types of middleware interfaces, various types of middleware functionalities are provided in Table 4.

Table 4: Middleware Applicable Open Standards

Type of Middleware	Examples	Notes
Scripting Engines	Bash, Lua, Ruby, Perl, Python, etc.	Many different types ranging from thin layers on top of operating systems to full-fledged application development environments
Application Development Frameworks	Java, QT, SDL	Java is unique in that it provides a portable Virtual Machine which may or may not require an operating system below it
Message Passing / Oriented	DDS, AMQP, XMPP, STOMP, OFED, MPI/OpenMPI, etc.	Some are well-defined by standards organizations. In the case of OFED, it includes optimization of drivers for performance as well
Parallel / Cloud Computing	HADOOP	Also thought of as a "distributed file system"
Databases	Multiple variants of SQL, various other databases, with standard connectivity	"SQL" as an interface is quite common, regardless of the vendor
Web Servers	Many various, including Apache	Massive ecosystem around web standards
App Ecosystems	Android, Tizen, FACE, etc.	Note that Apple iOS and Windows are similar, but proprietary

Applications

Ultimately, software applications sit at the top of the stack of layers and integrate the various functions into the desired capabilities. Although a turret may have completely manual controls (sights, hand cranks, triggers) for degraded-use, the turret stabilization and ballistics computation algorithms of a fire control application is what leverages the underlying hardware subsystems to provide the platform its desired capability.

The applications need to be written to a well-defined set of application programming interfaces (API). The layering, abstraction, and rigor are critical to software engineering, and have the benefit of now multiple decades of experience and evolution. The use of open standard API allows applications continual upgrades, enhancements, and refinements without requiring change to the underlying layers.

APPLYING ANALYSIS OF ALTERNATIVES

The Analysis-of-Alternatives (AoA) is a well-established and understood process within the defense acquisition processes; however, its use is only mandated at the top level of the acquisition process (capability to platform). General suitability, operational effectiveness and life-cycle cost for various alternatives are compared to determine if they meet the required capabilities. The concept and philosophy of AoA is applicable to all levels of abstracted design.

A direct top down requirements flow is illustrated in Figure 2.

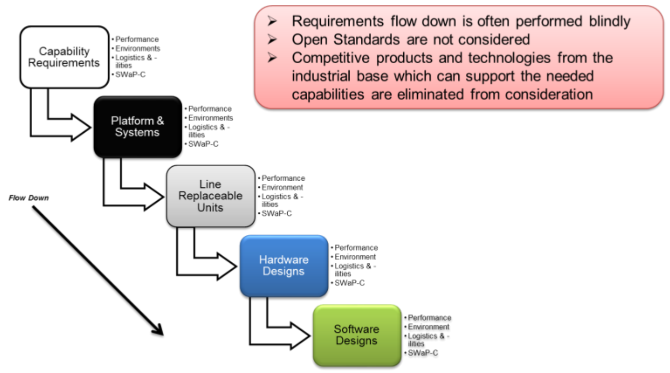
Requirements flow down is often performed blindly
Open Standards are not considered
Competitive products and technologies from the industrial base which can support the needed capabilities are eliminated from consideration
Capability Requirements
•Performance •Environments •Logistics & -ilities •SWaP-C
Platform & Systems
•Performance •Environments •Logistics & -ilities •SWaP-C
Line Replaceable Units
•Performance •Environment •Logistics & -ilities •SWaP-C
Hardware Designs
•Performance •Environment •Logistics & -ilities •SWaP-C
Software Designs
•Performance •Environment •Logistics & -ilities •SWaP-C
Flow Down

Figure 2: Typical top-down requirements flow

The problem here is that a *technically correct* solution (operational effectiveness and suitability) can be developed from a top-down process, but it may be *fiscally incorrect* with regard to lifecycle costs and commonality with other acquisition programs. It's important to realize the two things opposed here aren't "technical" and "fiscal", but "correct" and "incorrect". The correct solutions are infinitely outnumbered by incorrect solutions. The challenge here is to drive to the most correct technical and fiscal solution, in appropriate balance, rather than one or the other. These concepts are already embodied in the acquisition terms such as "Lowest Price Technical Acceptable", "Best Value", and "Best Performance", and "Not-to-Exceed" when an acquisition program evaluates bids. Similar to the many dimensions of technical performance (e.g. Size, speed, cargo capacity, etc.), fiscal performance has multiple dimensions: schedule (time is money), risk (representing statistical money), recurring cost, development cost, operational cost, maintenance cost, retirement cost, management cost, etc. All of these are summed into the general terms of life-cycle cost or total cost of ownership (TCO).

The AoA specifically requires a balance of both technical and fiscal, is already applied from the capability to Platform / System level, and should also be applied at all levels as illustrated in Figure 3.

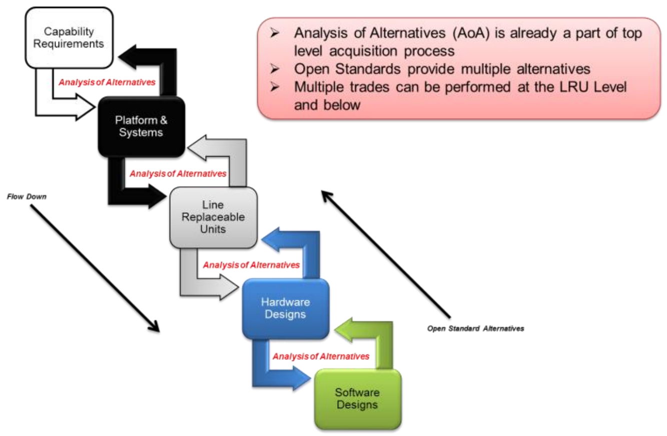
Capability Requirements
Analysis of Alternatives
Platform & Systems
Analysis of Alternatives
Line Replaceable Units
Analysis of Alternatives
Hardware Designs
Analysis of Alternatives
Software Designs
Flow Down
Open Standard Alternatives
Analysis of Alternatives (AoA) is already a part of top level acquisition process
Open Standards provide multiple alternatives
Multiple trades can be performed at the LRU Level and below

Figure 3: Analysis of Alternatives utilized at multiple requirements levels

With the AoA at all layers, the trade-off of technical capability and fiscal takes on a new challenge. As a major system is in consideration for acquisition to serve a capability, it is expected to have some interrelation with existing systems (e.g. different vehicles co-existing or cooperating in formations). Depending on desired capabilities, the AoA will take this into consideration and will make it part of the evaluation. The challenge is to continue the consideration of interrelation and synergies when evaluating subsystems, hardware designs, and software designs, rather than continuing in a stove-piped top-down manner.

In the absence of availability of open standards to leverage across multiple acquisitions, coordination would have to occur in the development of point-design specifications for different levels of abstraction. This is an extremely difficult task as each development would require highly coordinated working groups, or leader/follower policies which forced subsequent acquisitions to utilize previous work with the hope that it was generalized enough to reuse. On the other hand, the use of open standards means that program agnostic technical specifications are leveraged without concern that they were developed for one program or another, or by one commercial interest or another (e.g. two competing prime contractors).

This point needs to be emphasized. Open Standards are developed without isolated interest by a single company or organization. The very “open” nature of the standard is to encourage the growth of an ecosystem and business market, through increased adoption and straightforward usage. As the market and adoption grows, competitive forces drive evolutionary improvements into product offerings, resulting in more alternatives for both technically and fiscally correct solutions. Organizations will seek to share the market space by meeting the common open standard at the interface level while differentiating through various parameters, such as performance, cost, schedule, risk, or value-add support.

Nevertheless, a traditional conceptual disconnect is the assertion that commercial – or more generally “non-defense” – standards aren’t suitable for defense usage. This is a fiscally dangerous and incorrect perception, which eschews the modern demands of adjacent markets with significant number of congruent requirements and environments, and further ignores the encapsulation and isolation afforded by a well-layered approach.

Figure 4 illustrates the path forward, highlighting critical interfaces levels for focus in order to ensure technical and fiscal success.

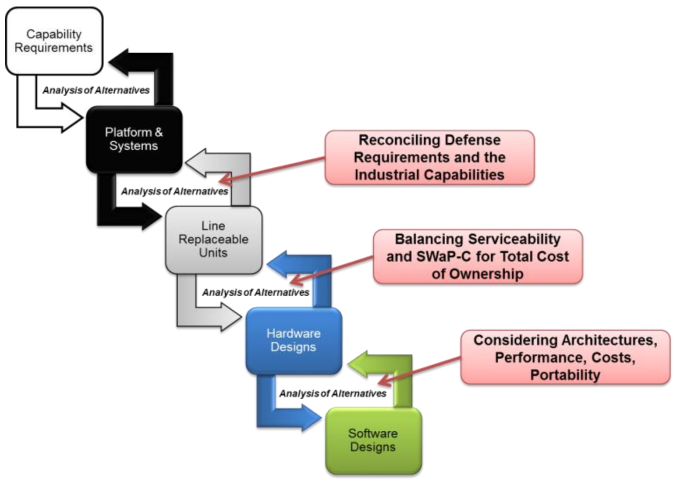

Figure 4: Critical AoA Interface Levels

The defense platform or system is just that – something defined specifically to provide set of defense capabilities. It must meet operational and other requirements specific to the domain; however, this does not mean that subsystems and components must be, from the ground up, specifically defined to meet defense capabilities. In some cases, a particular subsystem – for example stealth-enabling coatings or specific armor compositions – are designed specifically for the defense end-use as they are the capability providing subsystem (e.g. stealth or a certain level of survivability), but their use doesn’t mean that other subsystems must also have an equivalent level of ground-up clean-sheet design. This is the heart of the Analysis-of-Alternatives, and as it applies through the various levels, these key questions need to be asked:

<u>Moving from Platform / System to Subsystems</u>
How can defense requirements be reconciled with industrial capabilities? Is there a high level of congruence which can be used directly, some adaptation or refinement needed to a good starting point, or is there nothing even close?

<u>Moving from Subsystems / LRU to Hardware Designs</u>
How does internal modularity and SWaP-C balance against Total Cost of Ownership? Does it cost more to make a subsystem modular (for serviceability, etc.) than it would if it was just replaced? Is the SWaP-C burden of modularity worth it given the cost of the internal modules / backplanes?

<u>Moving from Hardware Designs to Software Designs</u>
What's more important, the reusability, portability, and long term maintenance cost of an architecture, or highly optimized point-designs? What's the focus: cost to develop, cost to maintain, cost to reuse, or cost to optimize performance?

With an understanding of the above questions, open standards can be evaluated in a succinct way: Is an open standard approach (subsystem, hardware, software) more or less suitable to the performance and cost requirements than a custom / proprietary approach?

Although this is a simple evaluation criterion, it relies on a very significant assumption. Open Standards, to meet their potential, must be used properly and as intended. Failure to understand or use properly will negate benefits, eliminate potential for competition, and destroy any chance of open standard based commonality.

EXAMPLES OF HOW TO USE OPEN STANDARDS

Unfortunately, not all Open Standards documents come with a "how to use" tutorial. Nevertheless, supporters of open standards have a vested interest in providing information how to properly use them. Information and guidance is available, and must be leveraged.

Returning to the example of a screw provides a simple example of how to and how not to use an open standard. Note that this is oversimplified for clarity.

<u>How to use:</u>
"The screw shall be 6-32 x 1 inch in accordance with the referenced screw standards."

<u>How not to use:</u>
"The screw shall be type #6 (root diameter 3/32) with 32 threads per inch and 1 inch long as measured from the point to the underside of the head, as shown in the attached vendor drawing."

The first example clearly calls out the pertinent information as needed in the screw standard – the gauge, thread spacing, and overall length, and uses the nomenclature as intended by the open standard. In the second example, additional information is provided in the form of additional measurements, spelled-out dimensions, measurement baselines, and reference to a vendor drawing, yet no reference to the screw standards are provided.

The first example limits information to what's needed, and leaves the rest to the standard. The intent communicated here is: "I want an open standard screw that fits."

The second example provides additional, potentially conflicting, information. The intent here is very unclear. Does the designer want that particular screw from that particular vendor? If so, then just call out the drawing / part number, and be done with it ("The screw shall be ACME P/N 1234 as shown in ACME drawing 1234.dwg"). Or do they just want an open standard screw, but have referenced additional documentation and measurements in attempt at "completeness" or to make the requirements specification stand alone? Regardless, this is a serious impediment to commonality and competition as it can inadvertently exclude alternative open standards conformant parts.

If the designer wants a particular vendor and part number, then the requirements specification has created a de facto sole-source. While this may be reasonable for a subsystem for which an open standards based market does not exist (e.g. stealth coatings), it will require significant sole-source justification. Why is that particular screw required? Is it due to a certain strength requirement, perhaps a self-drilling tip, or some other performance feature? If so, then specify those separately as additional requirements (e.g. "shall have self-drilling feature for sheet metal"); however, there needs to be a very strong justification as to why these additional non-standardized features are required (e.g. significant assembly time cost savings when using self-drilling screws which outweighs the incremental cost of self-drilling screws). What's important is that the open-standard be called out, ensuring that initial commonality and competition is enabled, and then performance constraints be levied to refine the selection of available products within the open standard market.

In the other case, if the requirements are over-specified to "complete" the specification, or to make it stand alone, then costs will rise. The number of requirements to be verified has increased in that example, and will require additional inspections / analysis. Each requirement needs management through the process, which equates to significant labor hours. Had the example simply called out the open standard nomenclature and parameters, the procurement, intake, and analysis of the part could be as simple as verification that the vendor's part description on a certificate of conformance matched the requirements (a supplier quality management task, not a design requirements management task). Instead, the inclusion of additional details and a drawing will require the inspection of the part versus mechanical drawing itself, and will also preclude the straightforward cross of the part by an alternate vendor for various reasons (e.g. a permitted vendor mark, a difference within tolerances of the standard, dissimilar part numbers, etc.).

The other problem with this is the concept that a proper development specification should incorporate or “pull-in” the language of external documentation to ensure control. In the absence of open-standards, this is a prudent way to maintain control in the face of conflicting program requirements which may diverge at any given time and without notice. Open standards, however, are governed by organizations which have a vested interest in stability, interoperability, transparency, and wide-spread adoption. The standards are designed to be referenced (rather than incorporated), and any changes tend to be well publicized and often backward compatible to ensure continued use. For this reason, it is sufficient to call out an open standard along with a version, with the understanding that each version will be widely supported.

Worth highlight is the common occurrence of open standards which reference other open standards. This is increasingly common when open standards build or evolve upon other stable, well-known, and mature open standards. For example, the ANSI / VITA standard for OpenVPX builds upon other ANSI / VITA VPX standards, which themselves build upon other various ANSI / VITA and IEEE standards. A common error is to flatten the standards unnecessarily, calling out the top standard and also calling out the sub-referenced standards. The requirements specification may feel more complete, but it introduces potential confusion, version conflicts, requirements management, and complication to the systems engineering process.

Vendor-specific, or proprietary, extensions to standards are not uncommon. Many standards allow for this as a method by which innovation and competition is encouraged, but standards-based interoperability is still ensured. As the standard evolves, illustrated in Figure 5, various vendor extensions will often become a part of the standard, while some may stay outside of it, and other vendors create extensions to the new standard version. In this way, the standards are able to evolve and adapt as technologies advance, while maintaining a core set of compatibility, interoperability, and competition through the current standard baseline.

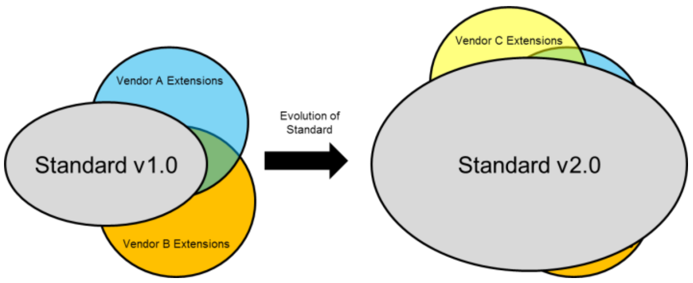

Figure 5: Evolution of Standards and Vendor Extensions

Ultimately, how to use open standards comes down to intent. How the open standard is referenced, the requirement written, and the performance specified must support the intent of using the standard in the first place. With the clear admission that the use of open standards may serve overriding policy (e.g. OSA), technical, and fiscal goals, the following examples presents a summary template for how best to use open standards based on intent.

Intent: Meet Policy

If the sole reason to use an open standard is to meet a policy requirement (e.g. “use Ethernet”), then the requirements language needs to be kept as broad as possible. A policy directive to use a specific open standard is not a technical or fiscal requirement on a specific program. It may be based on any number of high-level rationales, such as maintaining or growing an industrial-base capability, jump-starting a new standard and market, or laying a foundation for more explicit or specific use of the standard in later program stages (“if you build it, they will come”).

Note that a policy directive is often required to retire a technology in favor of a more advanced one. The move from analog (NTSC) to digital (ATSC) television transmission in the United States was forced via policy (in the form of law), as was the move from leaded to unleaded gasoline. In both cases, multiple business interests serving the broader market (e.g. television broadcasters, television manufacturers) were assured that adoption would occur, and no one part of the industry would be able to hold back, and no one part of the industry would be forced to assume undue risk that the market would not follow. When all competitors within an industry are forced to switch together, the cost and risk is a neutral to their overall standings, although it does provide new opportunities for performance differentiations (e.g. a higher quality digital tuner coming from a company who previously had mediocre analog tuners).

It’s recommended, at this context, to keep requirement and verification as broad as possible. Some systems engineers will not be comfortable with these requirements and verifications as they are more qualitative than quantitative;

however, that is the nature of policy. See the examples in Table 5 for policy requirements at multiple layers.

Table 5: Using Open Standards to Meet Policy

Requirement	Verification
The Subsystem shall leverage Open Standard A	Analysis describing how Open Standard A was used in the Subsystem.
The Operating System shall accommodate Open Standard B	Analysis describing how the Operating System accommodated the use of Open Standard B
The Middleware shall adhere to Open Standard C	Analysis describing how the middleware adhered to Open Standard C
The Application shall be compatible with Open Standard D	Analysis describing how the application is compatible with Open Standard D

Intent: Ensure Interoperability

Ensuring interoperability may be for a number of different reasons. These could range from pure interoperability between functions, to reducing unique test equipment needs, to reducing integration risks. Regardless of the need, the open standard itself needs to be analyzed for interoperability statements, e.g. between versions, forward / backward compatibility.

An open standard can have portions which do not interoperate with each other. For example, the USB standard has multiple connector types (standard, micro, mini, etc.) which are electrically compatible, but physically incompatible. The key is to ensure the proper variants within the standard are identified and called out. In addition, standards may discontinue, or deprecate, the use of earlier provisions in favor of evolved or refined standards. Standards may also allow vendor specific extensions or recommended provisions, which are not core to the standard. All of these variations need to be assessed and accounted for in the requirements language.

As with policy, it’s recommended that the requirements be kept broad; however, variations allowed within the standard must be constrained. To the intent of interoperability, it’s important that the requirements be levied at the interfaces of all entities intended to be interoperable.

See the examples in Table 6 for various interoperability requirements with Subsystems X and Y, built by different vendors at different points in time. Note the intent on Subsystem Y requirements.

Table 6: Using Open Standards to Ensure Interoperability across Vendors and Lifecycle Phase

Requirement on Subsystem X	Intent	Requirement on Subsystem Y
Subsystem X’s Interface shall conform to Open Standard A, Revision 1.6	New subsystem which uses current standard and has full interoperability with older versions of the standard.	Subsystem Y’s Interface shall conform to Open Standard A, revision 2.0. Subsystem Y’s interface shall provide backward compatibility to Open Standard A, revision 1.6, including the use of any interfaces deprecated from 1.6 to 2.0.
Subsystem X’s Interface shall conform to Open Standard A, Revision 1.6	New subsystem which uses current standard and has limited constrained interoperability with older versions of the standard	Subsystem Y’s Interface shall conform to Open Standard A, revision 2.0. Subsystem Y’s interface shall provide backward compatibility to Open Standard A, revision 1.6
Subsystem X’s Interface shall conform to Open Standard A, Revision 1.6	New subsystem which uses current standard will only interoperate with the older version of the standard per any provisions which are inherent in the standard.	Subsystem Y’s Interface shall conform to Open Standard A, revision 2.0.
Subsystem X’s Interface shall conform to Open Standard A, Revision 1.6, including recommended specifications 2, 3, and 5 and optional specification 10.	New subsystem which uses current standard has inherent interoperability with older versions of the standard, including recommended specifications and optional specifications. Assume recommended specifications 2 and 3 were moved to requirements in version 2.0, and optional specification 10 moved to recommended. Specification 5 remained at recommended.	Subsystem Y’s Interface shall conform to Open Standard A, Revision 2.0, including recommended specifications 5 and 10.

See the examples in Table 7 for various interoperability requirements for Subsystems X and Y using an open standard with variations for different roles. Note the intent on Subsystem Y requirements.

Table 7: Using Open Standards to Ensure Interoperability across Different Roles

Requirement on Subsystem X	Intent	Requirement on Subsystem Y
Subsystem X's Interface shall conform to Open Standard B, Revision 3.0, specification variants 7A and 8A (transmitter).	A complementary subsystem to the other	Subsystem Y's Interface shall conform to Open Standard B, Revision 3.0, specification variants 7B and 8B (receiver).
Subsystem X's Interfaces shall conform to Open Standard B, Revision 3.0, specification variant 15A (Optical)	A subsystem which supports multiple variants within the standard, and provides translation between them.	Subsystem Y's Interfaces shall conform to Open Standard B, Revision 3.0, specification variant 15A (Optical) and 15B (Copper) Subsystem Y shall translate signals from the Optical interfaces to the Copper interfaces in accordance with Open Standard B, Revision 3.0.
Subsystem X's Interfaces shall conform to Open Standard B, Revision 3.0, specification variants 20C and 20D	A subsystem that has some variant overlap with the other	Subsystem Y's Interfaces shall conform to Open Standard B, Revision 3.0, specification variants 20A and 20C.

Intent: Ensure Competition

Open standards enable competition. The misuse of open standards prevents it. To be very clear, many companies supporting open standards will seek competitive value-add for their products, sometimes extending the standard with vendor-specific features and capabilities, or implementing multiple open standards in one product. This is done to stave-off commoditization (e.g. milk is a commodity), because a commodity market is typically a "race to the bottom" in price. This can be a positive and a negative. Obviously, price is important; however, the level of investment in product innovation in commodity markets tends to be low, focused instead of operation costs. This condition remains until a new dimension for innovation becomes apparent, or the open standard is disrupted / abandoned in favor of a significantly more valuable offering, often proprietary.

Although price is good for discrete acquisitions, lack of innovation is damaging to overall continuous improvement in capabilities, and to the health of the industrial base. The use of open standards must be seen as the foundation for product evolution, not a constraint which drives commoditization. To ensure competition, open standard must be applied so that they only constrain what is necessary, but leave everything else open for innovation.

Innovation around open standards tends to come in two variants: *Combination* and *Adaption*. Both of these are used to optimize the value of products according to expected usage patterns.

In the first, separate functions are combined into one product that provides multiple functions in accordance with multiple open standards, often as lower SWaP-C and higher reliability than separate products. The risk in specifications preventing the use of competitive products like this is enforcing a "one box, one function" separation. There may be perceived reasons to keep functions separated into multiple boxes, such as survivability; however, depending on the number of functions in a single product, it may be better to have two of the product, replicating all the functions, rather than having a single separate box for each. For example, three functions combined into a single box, with two of the boxes providing redundancy across all three functions.

In the second, various interfaces which may or may not be open standard are separated from the main product because they are either low value add (e.g. few customers use it versus the cost of including it), the interface is legacy with declining use, or is unique to another 3rd party vendor. In these cases, a competitive product may remove or not support a fully integrated implementation of the interface, instead providing a separate adapter solution (for example, a USB to Serial adapter). The main product can be optimized around the newer, more common open standards, reducing SWaP-C, and the company can offer multiple alternative solutions for providing support for legacy or 3rd party interfaces. Requirements need to be flexible enough to allow for a main solution plus an adapter solution which meet the originally expected requirements of a single box. For example, a highly capable modern computing system may provide a software virtual machine to emulate older environments to host legacy applications, at a significantly lower cost than recreating the older processors and environments for native hosting of legacy applications. Another example is a smaller LRU with USB or Ethernet connections providing adaption to legacy interfaces where the SWaP-C of the combined product plus adapter is less than that of creating a box which meets the legacy interface requirements natively. In the case of 3rd party interfaces, it may be that other vendors are unable to incorporate the interface (e.g. legally prohibited from incorporating), therefore that interface may have to be kept separate, and purchased or licensed at the customer level or via customer authorization.

See the examples in Table 8 for how open standards can allow for competitive innovation. Note the explicit permission leaving room for innovation – these can be omitted if allowed at a global level.

Table 8: Using Open Standards to Ensure Competition

Requirement	Areas for Competitive Innovation
Subsystem X shall host applications in accordance with Open Standard A. Subsystem Y shall provide an interface in accordance to Open Standard B. Subsystem X and Y may be combined.	*Combination:* Product which hosts applications in accordance with Open Standard A and interfaces in accordance with Open Standard B.
Subsystem Y shall interface with existing Subsystem X in accordance with Open Standard A. Subsystem Y may use adaption / translation to interface with existing Subsystem X in accordance with Open Standard A.	*Adaption:* Product which fulfills requirements of Subsystem Y and interfaces with Subsystem X through adaption / translation mechanisms rather than directly incorporated interface.

Intent: Ensure Commonality

The intent of commonality is important. If it is for part number commonality (e.g. part 1234 is used across 5 platforms), then using open standards can be used to support Fit / Form / Function interchangeability. On the other hand, if the intent of commonality is to ensure common interfaces, common tools, and common training, then open standards are the most important driver.

Using an open standard interface allows a diversity of products and vendors which are all interchangeable. Two products from two different vendors may both adhere to common interfaces as defined by an open standard. If the configuration item specification is limited to what is in the open standard, then vendor specific additional interfaces which aren't used can be ignored. For example, a 10-pin connector with 6 open standard defined pins and 4 vendor defined pins should by specified only using the open standard pins, with silence on the vendor defined pins, assuming they are not used. Through this limitation, two products from two different vendors with differences only on the irrelevant vendor-defined areas may be considered interchangeable with regard to Fit / Form / Function.

In addition, constraining the interfaces to open standards, the various tools for maintaining, testing, and servicing can be common, even if the various products which plug-in to the interfaces are different. Most open standards also provide well-defined testing and diagnostic tools, or at least procedures, specifically designed to verify the solutions from multiple vendors. An added benefit when using open standards for common interfaces is that training for development or maintenance teams won't be restricted by proprietary information concerns. On the contrary, given the general goal of an open standard to increase its adoption, it means that training, tools, and information will be readily available.

The examples in Table 9 show how to use open standards to ensure commonality.

Table 9: Using Open Standards to Ensure Commonality

Intent	Requirement
Ensure Fit / Form / Function interchangeability	Subsystem X interfaces shall adhere to Open Standard A. Subsystem X interfaces specified in Open Standard A as vendor defined shall be left unused.
Ensure common interfaces	All subsystem interfaces shall adhere to Open Standard A
Ensure common maintenance tools	All subsystem maintenance interfaces shall adhere to Open Standard A *(Alternatively)* All maintenance tools shall be in accordance with Open Standard A.
Ensure common training	All training shall be in accordance with Open Standard A.

THE IMPACT ON COMMONALITY AND COMPETITION

Ultimately, open standards provide vendors and customers a way to focus on the performance and capabilities that matter, rather than on common and competitively non-value add features. It's easy to understand this when considering a counter-example: if gasoline formulations and gasoline nozzles were not standardized, then each automobile company would need to support its own complete ecosystem of gasoline types, nozzle types, testing, regulatory compliance, distribution, and resources. The end result is that any given automobile company would be forced to expend significant resources to sell gasoline (a commodity), thus diverting investment and focus from innovation in their core focus – designing and selling automobiles. For the customer base, it means that the decision about the purchase of an automobile would involve the proximity of company-specific gasoline stations, rather than focusing on the suitability, effectiveness, and price of the automobile itself.

In order to gain the benefit of open standards, it’s critical to utilize them correctly, as intended. If the standards are ignored or misused, the market may actually start to respond poorly, with vendors starting to abandon the open standards as a non-value add liability, or at least no longer relevant. Once this occurs, exclusive, proprietary standards will start to proliferate, fracturing the marketplace, and virtually eliminating the opportunity for commonality and competition.

CONCLUSION

Specifications are critical. Understanding the specifications which already exist within open standards and how to leverage them properly is essential to ensuring commonality and competition. The various layers of open standards provide foundations upon which to build capabilities in systems, and through proper use of the specification language of an open standard, the maximum amount of flexibility, interoperability, and interchangeability can be achieved. Using the concept of Analysis-of-Alternatives at multiple levels (not just platform / system), various open standards can be evaluated against each other and proprietary / custom solutions for best technical and fiscal correctness. It’s important to remember that open standards are developed and supported by multiple stakeholders within various markets, including vendors and user organizations, and that a significant amount of domain-specific work has been performed to ensure the best performance and suitability to the particular market when used as intended. Open standards, by their nature, strive for widespread adoption. Widespread adoption brings maturity, strong competition, and well-defined commonality, driving best value for the end customer. Circumventing open standards by improper use in specification reduces or eliminates these benefits, resulting in unnecessary costs, poor performance, and the risk of proprietary lock-in.