NPS-AM-11-C8P19R03-068

EXCERPT FROM THE

PROCEEDINGS

OF THE

EIGHTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM THURSDAY SESSIONS VOLUME II

Using Architecture Tools to Reduce the Risk in SoS Integration

Chris Piaszczyk, Northrop Grumman

Published: 30 April 2011

Prepared for the Naval Postgraduate School, Monterey, California 93943

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

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

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

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

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

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

Preface & Acknowledgements

During his internship with the Graduate School of Business & Public Policy in June 2010, U.S. Air Force Academy Cadet Chase Lane surveyed the activities of the Naval Postgraduate School's Acquisition Research Program in its first seven years. The sheer volume of research products—almost 600 published papers (e.g., technical reports, journal articles, theses)—indicates the extent to which the depth and breadth of acquisition research has increased during these years. Over 300 authors contributed to these works, which means that the pool of those who have had significant intellectual engagement with acquisition issues has increased substantially. The broad range of research topics includes acquisition reform, defense industry, fielding, contracting, interoperability, organizational behavior, risk management, cost estimating, and many others. Approaches range from conceptual and exploratory studies to develop propositions about various aspects of acquisition, to applied and statistical analyses to test specific hypotheses. Methodologies include case studies, modeling, surveys, and experiments. On the whole, such findings make us both grateful for the ARP's progress to date, and hopeful that this progress in research will lead to substantive improvements in the DoD's acquisition outcomes.

As pragmatists, we of course recognize that such change can only occur to the extent that the potential knowledge wrapped up in these products is put to use and tested to determine its value. We take seriously the pernicious effects of the so-called "theory–practice" gap, which would separate the acquisition scholar from the acquisition practitioner, and relegate the scholar's work to mere academic "shelfware." Some design features of our program that we believe help avoid these effects include the following: connecting researchers with practitioners on specific projects; requiring researchers to brief sponsors on project findings as a condition of funding award; "pushing" potentially high-impact research reports (e.g., via overnight shipping) to selected practitioners and policy-makers; and most notably, sponsoring this symposium, which we craft intentionally as an opportunity for fruitful, lasting connections between scholars and practitioners.

A former Defense Acquisition Executive, responding to a comment that academic research was not generally useful in acquisition practice, opined, "That's not their [the academics'] problem—it's ours [the practitioners']. They can only perform research; it's up to us to use it." While we certainly agree with this sentiment, we also recognize that any research, however theoretical, must point to some termination in action; academics have a responsibility to make their work intelligible to practitioners. Thus we continue to seek projects that both comport with solid standards of scholarship, and address relevant acquisition issues. These years of experience have shown us the difficulty in attempting to balance these two objectives, but we are convinced that the attempt is absolutely essential if any real improvement is to be realized.

We gratefully acknowledge the ongoing support and leadership of our sponsors, whose foresight and vision have assured the continuing success of the Acquisition Research Program:

- Office of the Under Secretary of Defense (Acquisition, Technology & Logistics)
- Program Executive Officer SHIPS
- Commander, Naval Sea Systems Command
- Army Contracting Command, U.S. Army Materiel Command
- Program Manager, Airborne, Maritime and Fixed Station Joint Tactical Radio System

- Program Executive Officer Integrated Warfare Systems
- Office of the Assistant Secretary of the Air Force (Acquisition)
- Office of the Assistant Secretary of the Army (Acquisition, Logistics, & Technology)
- Deputy Assistant Secretary of the Navy (Acquisition & Logistics Management)
- Director, Strategic Systems Programs Office
- Deputy Director, Acquisition Career Management, US Army
- Defense Business Systems Acquisition Executive, Business Transformation Agency
- Office of Procurement and Assistance Management Headquarters, Department of Energy

We also thank the Naval Postgraduate School Foundation and acknowledge its generous contributions in support of this Symposium.

James B. Greene, Jr.
Rear Admiral, U.S. Navy (Ret.)

Keith F. Snider, PhD
Associate Professor

Panel 19 – System-of-Systems Acquisition: Concepts and Tools

Thursday, May 12, 2011	
11:15 a.m. – 12:45 p.m.	**Chair: Rear Admiral David H. Lewis,** USN, Program Executive Officer, Ships ***Capability and Development Time Trade-off Analysis in Systems-of-Systems*** Muharrem Mane and Daniel DeLaurentis, Purdue University ***System-of-Systems Acquisition: Alignment and Collaboration*** Thomas Huynh, John Osmundson, and Rene Rendon, NPS ***Using Architecture Tools to Reduce the Risk in SoS Integration*** Chris Piaszczyk, Northrop Grumman

Rear Admiral David H. Lewis—Program Executive Officer Ships. Rear Admiral Lewis is responsible for Navy shipbuilding for surface combatants, amphibious ships, logistics support ships, support craft, and related foreign military sales.

Born at Misawa Air Force Base, Japan, Lewis was commissioned in 1979 through the Navy ROTC Program at the University of Nebraska–Lincoln with a Bachelor of Science degree in Computer Science.

At sea, Lewis served aboard USS *Spruance* (DD 963) as communications officer, where he earned his Surface Warfare qualification; USS *Biddle* (CG 34) as fire control officer and missile battery officer; and USS *Ticonderoga* (CG 47) as combat systems officer. His major command assignment was Aegis Shipbuilding program manager in the Program Executive Office Ships, where he helped deliver seven DDG 51 class ships and procured another 10 ships.

Lewis' shore assignments include executive assistant to the assistant secretary of the Navy (Research, Development and Acquisition), assistant chief of staff for Maintenance and Engineering, commander, Naval Surface Forces, where he also served as a charter member of the Surface Warfare Enterprise. Other ship maintenance and acquisition assignments ashore include the Navy Secretariat staff; commander, Naval Sea Systems Command staff; Aegis Shipbuilding Program Office; supervisor of Shipbuilding, Bath; and Readiness Support Group, San Diego. Upon selection to flag rank, Lewis served as vice commander, Naval Sea Systems Command. Lewis earned a Master of Science degree in Computer Science from the Naval Postgraduate School. He completed the Seminar Course at the Naval War College Command and Staff School, and received his Joint Professional Military Education certification. He is a member of the Acquisition Professional Community with Level III certifications in Program Management and Production Quality Management, and has completed his civilian Project Management Professional certification.

Lewis' personal awards include the Legion of Merit, Meritorious Service Medal, Navy and Marine Corps Commendation, Navy and Marine Corps Achievement Medal, and various service and unit awards.

Using Architecture Tools to Reduce the Risk in SoS Integration

Chris Piaszczyk—INCOSE Certified Systems Engineering Professional (CSEP), and Microsoft Certified Systems Engineer (MCSE). Mr. Piaszczyk is a New York State Licensed Professional Engineer (PE). In the course of his employment within the aerospace industry, he enjoyed a career spanning analysis and design applications from low earth orbit spacecraft to high energy physics particle accelerators. His systems engineering experience includes structural dynamics analysis and design, fatigue and fracture analysis and design, systems optimization, reliability, availability and maintainability, requirements analysis, and systems architecturing. Mr. Piaszczyk holds a doctorate in Applied Mechanics from the Polytechnic Institute of New York and a master's degree, also in applied mechanics, from the Polytechnic Institute of Warsaw in Poland.

Abstract

DoD acquisition is evolving from the traditional approach focused on individual systems to system-of-systems (SoS) integration. In DoD terminology, SoS is a collection of systems integrated together to obtain a higher level system that offers more than the sum of its parts, though the individual systems are acquired independently. System interactions within the SoS typically produce emergent capabilities that may or may not be desired. Any undesired behavior represents an integration risk and must be recognized, analyzed, and understood. Architectural tools are evolving to provide this understanding. These tools can be used for analyses of SoS designs to predict unexpected couplings and to avoid the potential for missed, underutilized or duplicated functionalities. Architectural artifacts developed with these tools expose potential issues to the design community. In addition, these artifacts provide a foundation for integration test planning by identifying and documenting the interfaces between hardware, software and humans that constitute the SoS. This presentation describes the related concepts and processes.

Systems-of-Systems and Systems

The term "system-of-systems" needs some discussion. A number of interpretations are in use by the systems engineering community. In a certain sense, "every system is a system-of-systems." Since every system-of-systems is, by definition, also a system, this way of thinking leads to a tautology that is not very useful.

One of the possible SoS definitions has been proposed by Mark Maier (famous Eberhardt Rechtin's collaborator on *The Art of Systems Architecting*) in his 1998 paper "Architecting Principles for Systems-of-Systems." To summarize Maier's definition of SoS,

1. SoS components must be able to usefully operate independently.
2. SoS components are independently acquired and maintain independent management existence.
3. SoS continues to evolve.
4. SoS exhibits emergent properties.
5. SoS components interact only by information exchanges (are geographically distributed).

This definition defines a subclass of the more general concept of a system. Hence, according to this definition, every system-of-systems is a system but not every system is a system-of-systems. As discussed in the following, the ideas contained here can be explored with useful outcomes.

Since Maier's definition consists of multiple parts, it leads to several weaker forms, each defining a subset of the set of systems with the class of system-of-systems defined according to Maier being as their set theoretical intersection. Below, this definition is examined more closely, with the conclusion that the most important characteristic of the SoS defined with it is the first part stating that SoS components must be able to "usefully operate independently." Thus, for example, a bicycle is a system but not a system-of-systems. There are parts of the bicycle, such as the frame, that cannot "usefully operate independently," except perhaps with some very creative ideas.

Interestingly, the criterion of operational independence immediately brings to mind the concept of a system consisting of loosely coupled objects known from discussions of open architectures. However, for these objects to form a system-of-systems, they also have to be able to completely decouple and act independently, in addition to being loosely coupled. Thus, there may be open architecture systems that are not systems-of-systems, and systems-of-systems that are not open architecture. Since it is possible to find examples of systems-of-systems with open architectures, such as for instance the Internet, the most one can say is that the intersection of the two sets is not empty.

The second part of Maier's definition, the criterion of managerial independence, requiring components of the SoS to be independently acquired and maintaining independent existence is a qualifying attribute that is perhaps not as important for the formal definition of system-of-systems in general settings. In his paper, Maier further distinguishes between "directed," "collaborative," and "virtual" systems-of-systems that represent variations on the degree to which the SoS satisfies this condition. A "virtual" SoS satisfies this condition completely, a "directed" SoS satisfies it to the least extent and a "collaborative" one falls somewhere in between. Maier's examples of virtual systems-of-systems include the World Wide Web and national (and even more so) international economies. The Internet, on the other hand, is presented as an example of a collaborative system-of-systems, governed by the Internet Engineering Task Force by means of standards published in the form of Requests For Comments (RFC). An integrated air defense network, such as NORAD, centrally managed to defend the US, is an example of the directed kind of SoS. One could say the need for including this criterion in the definition of a system-of-systems is somewhat questionable even for DoD acquisitions, because many DoD system-of-systems are acquired and managed by one and the same organization, the DoD, although its many branches do operate independently, to a degree.

As for the third part of Meier's definition, it can probably be safely stated that the evolution of a system-of-systems could already be a natural consequence of the fact that its components are independently acquired and maintain independent existence, including independent evolution. On the other hand, any system of sufficiently large size evolves out of necessity to keep operating. This may be forced by high cost of its replacement. Loose coupling of components in an open architecture system is a characteristic designed for facilitating this evolution.

It is also debatable if "emergent properties" are truly a characteristic limited to systems-of-systems. First of all, there is a problem with the word "emergent." This word carries an aura of mystery. It has given rise to its own school of philosophical thought going back to the post-Darwinian England. "Emergentists" included such luminaries as J. S. Mill. In his 1843 opus, *A System of Logic, Book III*, he expressed the idea that "to whatever degree we might imagine our knowledge of the properties of the several ingredients of a living body to be extended and perfected, it is certain that no mere summing up of the separate actions

of those elements will ever amount to the action of the living body itself" (Ch. 6, § 1). The term continues to be used in biology literature to this day.

Although it must be conceded that some systems and their properties are so complex that they cannot be computed even today, the properties of every system can be unexpected or expected depending on the level of understanding of the system behavior characterized as "emergent." To quote Arthur Clarke (1961), "Any sufficiently advanced technology is indistinguishable from magic." However, such esoteric situations are outside the scope of this paper.

The point being made is that the properties of any system are always more than the "sum of its parts." Thus, "emergence" is not a qualifying attribute that distinguishes systems-of-systems from systems in general. The emphasis of this paper is on finding ways to prevent potential undesirable "emergent" effects. All passengers of commercial air transport feel much more comfortable thinking of an airplane as a system rather than a "collection of parts flying together in close proximity." However, it is very desirable to know and understand all possible "emergent" properties of this system.

The fifth of the Maier's criteria is very applicable to computer networks, which must have been the focus of the systems-of-systems engineering in 1998. However, a common example of a system-of-systems satisfying this criterion is provided by any group of human beings, and these go back much further in time. Human beings exchanged information by voice, paper and other methods long before computers were even conceived of. Perhaps methods developed by the systems-of-systems engineering can find fruitful application in the field of sociology. On the other hand, restricting the entire systems-of-systems discipline to those that are geographically distributed and interact only by information exchanges may be overly limiting. It may be more productive not to impose it.

In the following, consideration is given to systems-of-systems defined either in the strict sense by the full set of the five criteria in Meier's definition or a wider class defined by its weaker form consisting of just a subset of these satisfying at least the first one of them.

Elements of Risk (and Opportunity) in Systems-of-Systems Integration

There are many forms of risk associated with the development and integration of any system. Some risks are technical and some programmatic. A full investigation of all systems-of-systems integration risks was outside the scope of this effort. A more complete discussion may be presented in the future. The intent of this paper is only to highlight the usefulness of the architectural products in mitigating these risks in general. The discussion is limited to selected types of risk that appear to be mostly associated with systems-of-systems, as a set of examples as follows:

1. Missed/underutilized functionalities and/or interfaces of the component systems.
2. Undesirable emergent behavior, sneak interactions and unintended consequences.
3. Independent components evolution drifting to non-compliance with original standards.
4. Evolving SoS not following stakeholder needs.

A very significant form of risk associated with integration of systems consisting of independent systems is that of potentially missed or unidentified functionalities and/or interfaces of the components systems. These would then remain untested while the system is being integrated and could "show up" suddenly when the system is deployed and in use.

This can happen because the components systems of the system-of-systems are not being designed to specifications flowing down from the requirements set of the system-of-systems but are used “as they are.” Being independent, the component systems are not modified for integration into a system-of-systems by their original developers but are only “stitched together” to provide a new desired functionality at a higher level.

Of course, if a particular functionality of a component system is not initially recognized and is discovered, this could also represent an opportunity for making the system-of-systems more efficient and less costly. Otherwise, undiscovered functions represent a very real risk of failure when the system is put in operation and two formerly unidentified functionalities of the components systems interfere with one another. Similarly, an unidentified interface could represent a risk of the system simply not functioning as necessary or interfering with the desired operation. Undesirable emergent behavior, sneak interactions and unintended consequences are all potential manifestations of the risks of missed functionalities or unidentified interfaces having been realized.

Since the components systems of the system-of-systems are independent and therefore independently evolving, they could evolve away from the original standards they complied with when they were initially selected by the system-of-systems architects to the point where they will no longer fit with the rest of the system-of-systems. In the integration construct represented by a system-of-systems, especially one of the “virtual” category, the original interfaces and functionalities of the system-of-systems component systems can be defined solely by means of voluntarily followed standards. If these are the only means of “control” over the evolving components, nothing prevents the developers and manufacturers of these component to switch to a different standard or discontinue their products altogether. When the original product was not widely available from many sources, it may no longer be available at some point in time. Such possibility represents a kind of risk that at the parts level that the logistics discipline treats as Diminishing Manufacturing Sources and Material Shortages.

Uncontrolled evolution of a system-of-systems can lead to a paradoxical situation where it no longer satisfies the evolving needs of its stakeholders. This situation may continue for some time in some cases, but eventually, the funding stop may be brought about for various reasons depending on the SoS under consideration.

Mitigation of Risk (and Extraction of Opportunities) in Systems-of-Systems Integration

This paper postulates that the systems-of-systems integration risks identified in the previous section can be mitigated with the help of architectural tools. The context for this use of architectures is the developing new Model Based Systems Engineering paradigm that focuses the three core systems engineering processes consisting of requirements analysis, system design and requirements verification and validation around a model of the system. Since systems-of-systems are systems, methods developed for reducing the risks associated with development of systems are applicable to the systems-of-systems.

The following discussion shows how the architectural tools can be used for mitigation of systems-of-systems integration risk examples identified in the previous section:

1. Use of architectural tools to identify component functionalities and interfaces
2. Use of modeling and simulations to predict undesirable emergent behavior, sneak interactions and unintended consequences

3. Use of open standards to permit use of suitable replacements for components that will not be available as time progresses (avoid proprietary interfaces)
4. Management of evolving SoS requirements

Architectural tools are highly relevant to the task of identifying component functionalities and interfaces. Primarily, these tools provide the means to generate a graphical form of documentation but also, at least with some of the tools available at this time, to verify consistency of the architectural information entered into the tool database.

Modeling and simulations come at many levels, from the highest system level to the details of physics and chemistry of the tiniest component. A great variety of modeling and simulations tools are being used throughout science and engineering as suitable and necessary. Architectural tools provide capabilities to model the system as it is defined by "business" rules, states and modes, and swim lanes. These are in the category of PETRI nets, executable UML/SysML, etc., that will be briefly discussed in the following section. The usefulness of this level of modeling and simulation consists of gaining insights into system level behavior and discovery of potential undesirable effects of integration of the formerly independent component systems into the system-of-systems in question. Analysis of these models can potentially uncover the so-called "sneak interactions" that weren't apparent at first sight and after integration could produce "unintended consequences." There are no guarantees that all such bad side effects of system-of-system design decisions can indeed be discovered as a lot depends on the skills of the modelers, however, without this effort even the simplest behaviors can remain hidden until disaster strikes.

Development of open architectures, open standards and open business models is a major DoD thrust expected to yield significant cost savings in all acquisition programs. The desired benefits can only be achieved if the program follows the open systems guiding principles from the start. Open system architecture requires an investment in infrastructure. Patching up an existing design at a system-of-systems level usually requires a major architecture redesign that may be a difficult cost-to-benefit ratio to justify.

The 2004 DoD Joint Task Force *Modular Open Systems Architecture (MOSA) Program Manager's Guide* lists five "principles" (that look rather like steps of a management process) necessary to achieve an open architecture system design. These principles are as follows:

1. Establish an enabling environment.
2. Employ modular design.
3. Designate key interfaces.
4. Use open standards.
5. Certify conformance.

Most of the contents of the MOSA guide could be categorized as programmatic (or SOW-type) requirements. It is, after all, a "program manager's" guide. Clearly, Principles 1 and 5 are program management responsibilities. Principle 2 is calling for a modular design, which in MOSA's terms means the following:

- The system is functionally partitioned into discrete scalable, reusable modules consisting of isolated, self-contained functional elements.
- System design makes rigorous use of disciplined definition of modular interfaces, to include object-oriented descriptions of module functionality.

- Components are designed for ease of change to achieve technology transparency and, to the largest extent possible, make use of commonly used industry standards for key interfaces.

Programmatically, modularity is a requirement to produce a set of architectural artifacts that show the modules with identified functions and interfaces. In technical terms, a list of required system functions needs to be identified and then allocated to a set of components in such a way that closely interacting functions are lumped together in one module while less closely interacting functions are split across different modules. This approach simply minimizes the interactions between separate modules, reducing the necessary number of interfaces between them. The resulting minimal set of interfaces is then carefully characterized and published, creating the openness of the architecture. This facilitates the design or acquisition of a replacement in case the original component system is no longer available on the market or better performance can be obtained with a software update.

MOSA's Principle 3 is calling for identification of the Key Interfaces. Again, according to the *MOSA Program Manager's Guide*,

> the focus of MOSA is not on control and management of all the interfaces within and between systems. It would be very costly and perhaps impractical to manage hundreds and in some cases thousands of interfaces used within and among systems. …A key interface is an interface for which the preferred implementation uses an open standard to design the system for affordable change, ease of integration, interoperability, commonality, reuse or other essential considerations such as criticality of function.

The MOSA guide tells the Program Manager (PM) that "Programs must determine the level of implementation (e.g., subsystem, system, system-of-systems) at and above which they aspire to maintain control over the key interfaces and would like these interfaces to be defined by widely supported and consensus based standards." Thus, the PM decides at what level the Open Architecture (OA) requirements flow down should stop. This requires careful considerations with architectural artifacts being a key ingredient.

The last but certainly not least risk example identified for the system-of-systems acquisition in the previous section was the risk that the system will simply evolve away from the stakeholders' requirements. Well, the most important step in mitigating this risk is to identify those stakeholders' requirements in the first place. One cannot see that the evolution of the system is drifting away from the target unless one has a clear picture of what this target is. Here again, the architectural tools come to the rescue.

Architecting is an integral part of the systems engineering iterations consisting of requirements analysis, system design and requirements verification and validation. Requirements are used to manage the entire process by clearly identifying the objectives of the system development. Systems engineering, as a discipline, evolved in the post World War II era to reduce the risk associated with acquisition of increasingly complex defense systems, beginning with the Inter-Continental Ballistic Missile (ICBM), through Ballistic Missile Defense (BMD), and so on all the way to today's software-intensive multilayer products consisting of thousands of humans and computers organized into networks distributed across several continents, air, sea and space. Identification of separate, individually defined requirements reduces this risk by reducing complexity. It is much easier to manage the development and evolution of a complex system if it can be broken up into smaller, more easily digestible pieces.

This contribution of architectural tools can be illustrated with the following figure that shows the standard process beginning from the elicitation of stakeholder requirements followed by development of system specs, followed in turn by system design and ending with system implementation. Obviously, this is a very simplified view of this process. In reality this process is highly iterative and ebbs and flows back and forth, challenging program management as iterations can accumulate program costs not originally budgeted for.

The risk of the system evolving away from the stakeholder needs consists of three sub-risks: risk of requirements gaps, risk of design gaps and risk of implementation gaps. Mitigating these three risks requires attention to the derivation of technical requirements from the stakeholder needs, conversion of the technical requirements to system design and the implementation of the design into a physical system. Gaps can appear in any part of this process. Each gap has its own mitigation method. For example, gaps that can occur in the requirements derivation step, from stakeholder requirements to system specs, are addressed with tools specialized for maintaining requirements traceability, such as the Dynamic Object Oriented Requirements System (DOORS). In the MBSE paradigm, the requirements derivation step is strongly supported by the architectural products.

The risk of design gaps potentially appearing in the transition from specs to design is addressed with a process architecture allocation in which requirements are allocated to specific parts of the system architectures (functional, system and physical). Clearly, architectural tools must be employed to produce the architectures required in this step.

Finally, potential gaps in implementation of the design into physical form are prevented with the Requirements Verification Matrix (RVM), which is used for managing the test program. Requirements verification is also supported with architectural artifacts in the form of graphical representations of systems connections into a system-of-systems.

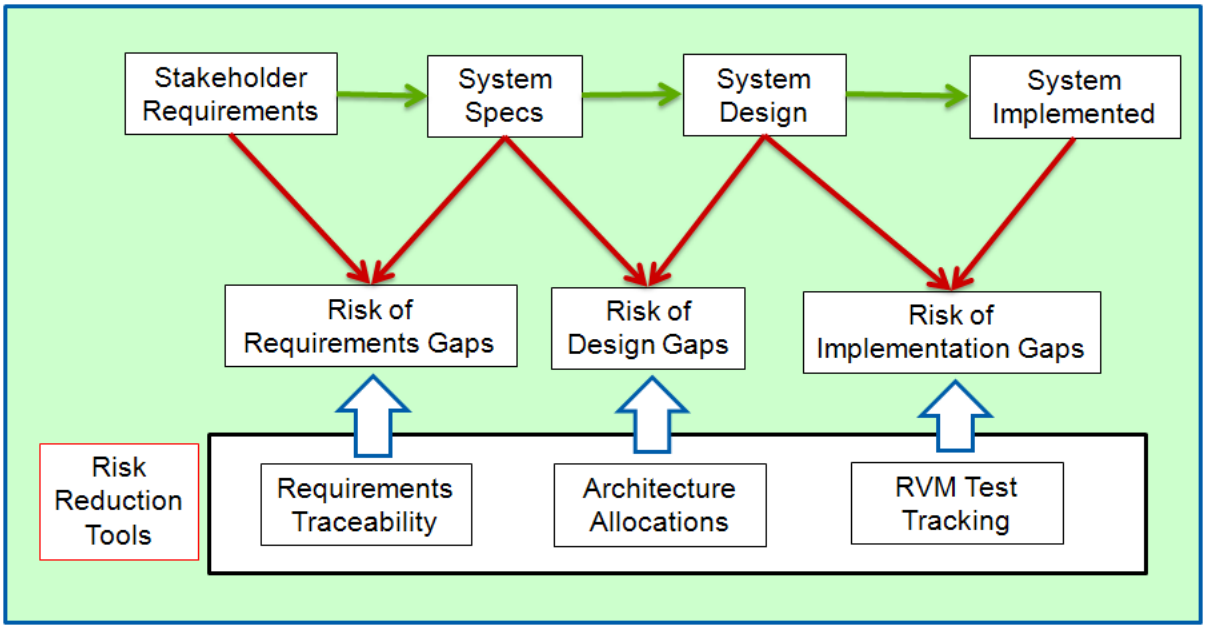

Figure 1. Architectural Tools Contribution to Risk Reduction in the General System Context

Architectural Tools and Products

Since the early 70s, an intensive effort has been underway to implement computer technologies in business environment. This effort has transformed the way we all work and live. The original leading agent of this change was IBM. Out of this organization came forth ideas that today are known under the name of Enterprise Architecture (EA). In his 1987 paper, "A Framework for Systems Architecture," John Zachman proposed a method for organizing the architectural artifacts into a matrix with six rows, corresponding to different levels of detail, and six columns, addressing the questions: what, how, where, who, when, and why. This construct became known as the Zachman Architectural Framework (ZAF). This brilliant idea was quickly adopted across a wide area of applications.

The DoD published its own C4ISR Architectural Framework in 1997, followed in 2003 by the DoD Architectural Framework (DoDAF), that by now is in revision 2.0, published in 2009. Other military organizations followed suite with the MoD Architectural Framework (MoDAF), NATO Architectural Framework (NAF), and so on. Civilian organizations were not far behind with The Open Group Architectural Framework (TOGAF), etc. Basically, the architectural frameworks define how to organize and structure the views associated with an architecture.

Computer science, another fast-developing field, brought us several generations of computer languages and programming approaches with the latest being the Object Oriented methodology incorporated into software development tools such as, for example, the Unified Modeling Language (UML). In 2006, the systems engineering community developed an extension of UML called the System Modeling Language (SysML) that can be used for modeling general systems.

Basically, SysML is a diagrammatic notation designed specifically to describe and understand general systems. Another category of graphical tools that can be used for the same purpose is the Integration Definition (IDEF) derived from the Structured Analysis Design Technique (SADT). DoDAF can be implemented with either the Object Oriented SysML or the Structured Analysis IDEF. DoDAF is another big subject, so the following is limited to the fundamentals.

DoDAF Operational Views

As mentioned previously, the Zachman framework defined six levels or viewpoints. The original, first version of DoDAF, used four: All-Views, Operational Views, System Views and Technical Views. The latest version of DoDAF, 2.0, defines eight viewpoints. Like in the Zachman's framework, each viewpoint in the DoDAF includes multiple types of views. This paper focuses on the most important operational view for this discussion, the OV-5, Operational Activity Model, and the two most relevant system views, the SV-1, System Interface Description and the SV-4, System Functionality Description. A more complete discussion of DoDAF products (views) recommended for SoS architecting can be found, for example, in the *Naval SoS SE Guidebook* (2006). Additional details can be also found in the original DoDAF documentation.

The Operational Views (OVs) are focused on the activities that are performed by the operators. They are normally developed as the first part of the analysis and assume that a system is a "black box," the details of which are as yet undefined. One starts with a very high-level overview, called OV-1, which is just a cartoon version of the proposed system in operation. Then, gradually, all operators, their activities and the information exchanges taking place are identified and documented with the OVs. In fact, the OV-5, Operational

Activity Model, consist of boxes representing activities and arrows representing the information exchanges.

An OV-2, Operational Node Connectivity Description, presents a complementary picture where the boxes represent operational nodes containing aggregations of activities and lines represent bundles of information exchanges between activities allocated to each node. Each operational node is a collection of activities, an abstraction that can be used to represent geographic separation or some other form of organization. In DoDAF 2.0, the OV-2 was renamed Operational Resource Flow Description, to extend the application of this view beyond information exchanges to more general resource flows. As a consequence, the operational views in DoDAF 2.0 can now formally be used to represent systems that are more general than information systems.

Generally speaking, the OVs are very useful for analyses of the human side of the system-of-systems design, roughly corresponding to an expanded concept of Use Cases known to software systems engineers using UML. While the OVs view the SoS as a black box, the System Views (SVs) define its internal workings.

DoDAF System Views

The SV-1, System Interface Description, uses boxes to represent the component systems of the system-of-systems and arrows to represent the interfaces between the systems. Thus, the SV-1 is essentially a block diagram which shows how the system-of-systems is integrated from its components. As such, it is an essential tool for managing any integration process, including integration test planning. One cannot envision integrating any system without some kind of graphical representation telling the integrators how the components connect together.

What's needed to complete this picture is a view that describes the functions performed by every one of the components and various layers of the system-of-systems assembly. One can begin by marking up the functions performed by the system components within each box that represents them. When integrating a system-of-systems consisting of existing or otherwise known components, this part is relatively simple to accomplish.

One needs to remember, however, as stated in the beginning, that a system is more than a simple sum of its parts. Hence, multiple layers of system-of-systems integration need to be documented with multiple SV-1s to show the functions emergent for every assembly of component systems, assembly of assemblies, and so on, to the final layer representing the complete system-of-systems as a single box with inscribed system-of-systems level functions. This leads to a multitude of SV-1s that may be difficult to digest. A hierarchical structure of functions at the various levels of system-of-systems assemblies can be summarized with one view called, SV-4, System Functionality Description. This view is also commonly known as a functional architecture of the system-of-systems.

The functional architecture essentially represents functional requirements in a graphical form. Functional requirements are basically a translation of the stakeholders' needs into technical terms. They need to match the activities previously identified in the OVs. While the system was represented by a "black box" in the OVs, here one takes a peek inside. All the details of the internal machinery of the system are not yet visible, only a set of smaller "black boxes" labeled with individual functions.

Functional architecture identifies required system-of-systems functionalities in a manner independent of specific choices made in selecting the component systems. This knowledge can now be used to accommodate changes in specific technological

implementation of one of the component system changes, by simply replacing it with another implementation that provides the same functionality. This reflects the principle of "separation of concerns" proposed by E. W. Dijkstra in *Selected Writings on Computing: A Personal Perspective* (1982, pp. 60–66).

A familiar example of this principle in action is the layered Open Systems Interconnection (OSI) stack used in computer networking. The stack allows a complex design problem to be split into less complex smaller problems with changes constrained to one layer at a time. Ultimately, such layered schema also leads to an open business model. Provided the vendors supply the desired functionality and comply with the interface standards governing the interactions with other layers, no one needs to see the inner workings of their systems. Accordingly, the system-of-systems (such as a computer network in this case) creates a market with many suppliers who are able to protect the intellectual property of their specific implementations.

Additional views are needed to provide a more complete picture of the system-of-systems for inputs to the modeling and simulation. These include the SV-10c, System Event Trace Matrix, that model the dynamics of events that occur when the system-of-systems is operating. This view corresponds to an OV-6c, Operational Event Trace Description on the operator side. For completeness, one should also include the SV-10a, Systems Rules Model that defines the conditions determining when specific system events are allowed to follow others and SV-10b, Systems State Transition Description that presents the states the system may find itself in and how it transitions from one state to another.

Using Architectural Tools and Products to Reduce the Risk in Systems-of-Systems Integration

As mentioned before, one of the risks encountered in integrating a system-of-systems is in missed or unidentified functionalities and/or interfaces of the components systems. The claim being made here is that using architectural tools that provide graphic representations of the component functions and interfaces that can be inspected by the SoS design team will contribute to the reduction of this risk in a significant manner. First of all, just adding a task that consists of creating such architectural artifacts forces the team to examine the component systems and document the results of this examination. Furthermore, functions and interfaces identified in such a task will provide inputs to the analysis of derived requirements that are part of the overall SoS requirements set. Finally, these requirements are the foundation for preparing the integration and test plans and procedures that will be used to manage the integration and test programs for the system-of-systems in question.

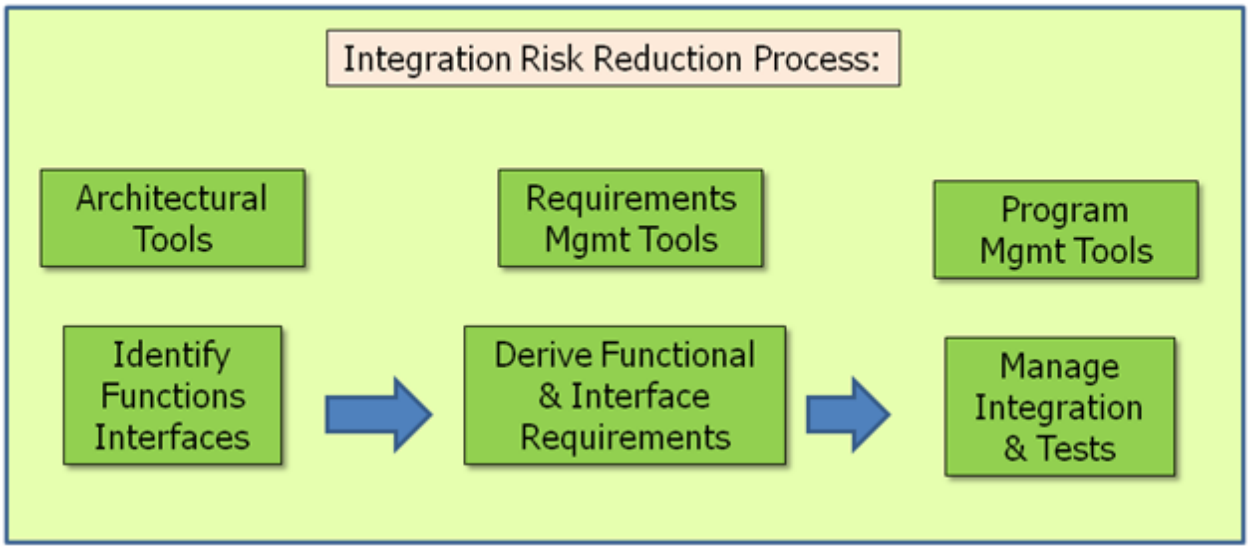

Figure 2. Functions & Interfaces Identified with Architectural Tools are Used to Derive Requirements and Manage Integration & Tests

The process is that of reverse engineering applied to each component system that will be integrated with the planned system-of-systems. The DoDAF SV-4 is used to document the provided functionalities and the DoDAF SV-1 is used to document the interfaces. Depending on the complexity of the component system under consideration, this analysis may be limited to the top-most layer or may delve into some internal details. Typically though, the top-most layer will suffice. The component system SV-1s and SV-4s are then used like LEGO blocks to construct the SV-1s and SV-4s for the entire system-of-systems.

Using Architectural Tools and Products to Reduce the Risk of Undesired Behavior

Another type of risk that exists in system-of-systems integration is the risk of undesired behavior that suddenly appears when component systems that have been developed for other uses get connected together. As discussed above, this undesired behavior can fall into many categories such as the so-called emergent behavior, sneak circuits or other unintended consequences. Although emergent properties have been associated with certain "esoteric" ideas, especially when the particular system-of-systems under consideration is at the edge of the current extent of accumulated human knowledge, many such effects can be uncovered through sufficiently detailed modeling and simulation. The architectural tools can be used to reduce the risk of unexpected behaviors of the system-of-system hiding behind an insufficient understanding of the functionalities and interfaces of the component systems. These functionalities and interfaces are an indispensable input to the modeling and simulation tools that can be used to analyze "sunny day" and "rainy day" scenarios where undesired SoS behaviors can be identified. Once identified, these behaviors can usually be mitigated.

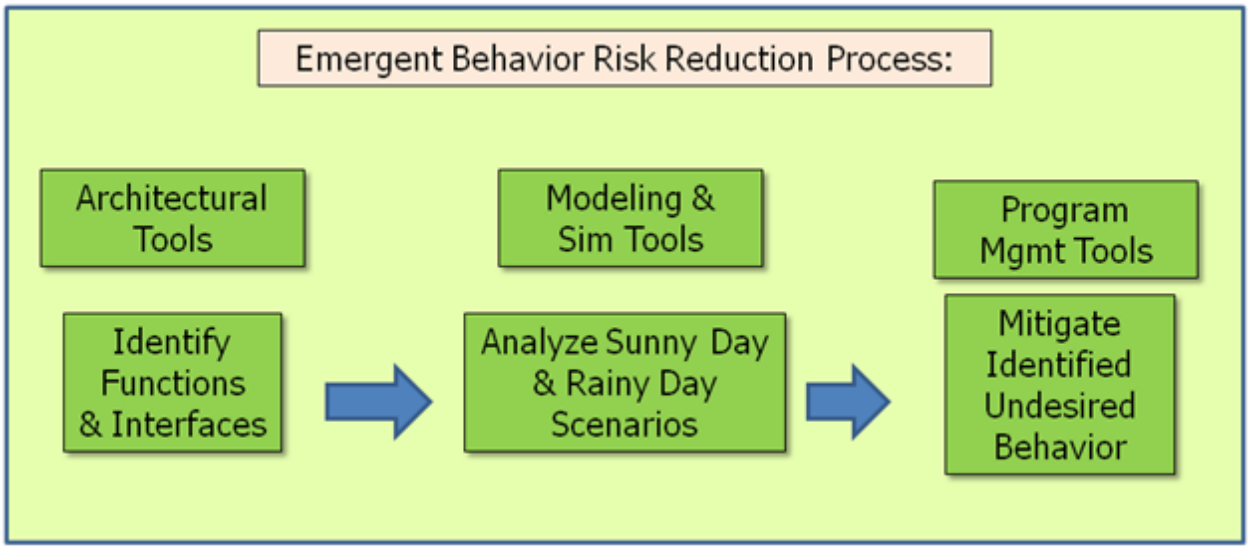

Figure 3. Architectural Products Help Mitigate Undesired Behavior

At the system-of-systems level, one can now use the information previously identified in the SV-1 and SV-4 to build DoDAF System Event Trace Matrix diagram, SV-10s. The system-of-systems architect will also develop an SV-10a, the Systems Rule Model that defines the conditional behavior of the SoS. System State Transition Description, SV-10b will help examine the system-of-systems states and modes. This diagram describes what response is to be expected for a given stimulus. System-of-systems response may vary depending on the current state, the type of stimulus, and the trigger guard conditions. Explicit responses to stimuli are not found in a functional architecture. Missing states, responses and conditions are equivalent to missing requirements.

Using Architectural Tools and Products to Reduce the Risk of Components Obsolescence

The risk of components obsolescence and the more general risk of the system evolving away from its original intent are two types of risk that are definitely more prevalent for system-of-systems that are integrated from independent component systems capable of evolving on their own than for systems developed as one entity. Principles of modularity and openness of architecture have been specifically developed to promote reuse and reduce obsolescence. These principles were defined and discussed in detail earlier in this paper.

As discussed, the architectural tools are key to identifying the functions and interfaces of the component systems. Carrying it one step further, the same tools can be used to identify functions and interfaces of assemblies of the component systems within the system-of-systems all the way up to the highest level. Implementation of modularity and open architecture in the design is basically the task of partitioning the design to isolate certain groups of functions to specific assemblies or modules and selection of open standards for key interfaces between these modules. An assembly here can consist of a single system. Once the design has been modularized in this fashion, multiple vendors can compete in the open market for each module and reduce if not eliminate the obsolescence risk, an effect experienced daily with computer technologies.

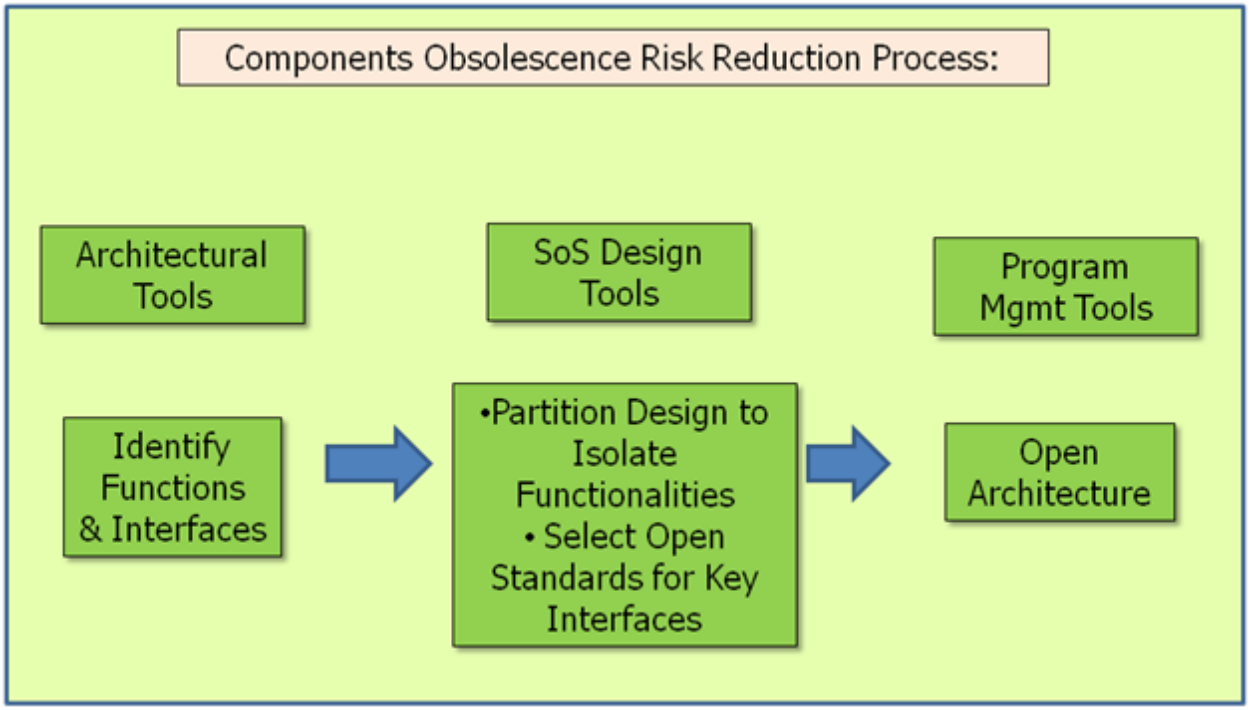

Figure 4. Architectural Products Help to Build an Open Architecture

Control of the evolution of the entire system-of-systems is part of its configuration management process. Architectural tools provide the necessary documentation. The system-of-systems SV-1 documents its overall configuration, identifying the component systems and their interfaces with other component systems. Their associated functionalities are documented with the corresponding SV-4 diagrams. Operational Activity views, OV-5, related to the system views via the SV-5a, Operational Activity to Systems Function Traceability Matrix, and SV-5b, Operational Activity to Systems Traceability Matrix, document the ways the operators use the system-of-systems. Through allocations, derived requirements are associated with each element of the system-of-systems architecture. Traceability binds the derived requirements to the top level originating stakeholders' requirements. Having the system-of-systems configuration documented in the form of architectures tightly bound with the top level and derived requirements allocated to the architectural elements is a great step towards reducing the risk of a system evolving in such a way that it would no longer serve its stakeholders.

Summary and Conclusions

In summary, several significant types of risk that appear in system-of-systems integration were analyzed and appropriate mitigation methods based on application of architectural tools were presented. Discussion of several available architectural frameworks and tools for developing architectural artifacts introduced the reader to these concepts and recommendations for further reading were provided. Specific ideas for application of these architectural artifacts bring us to the conclusion that use of architectural tools and products does reduce the risks in systems-of-systems integration as follows:

- Documented functionalities and interfaces for SoS components enable generation of requirements for better planning of system integration and test (and these reduce the risk of program failure).

- Documented functionalities and interfaces for SoS components enable higher fidelity modeling and simulation providing more insight into emergent behavior (and this reduces the risk of possible surprises).
- Documented functionalities and interfaces for SoS components facilitate creation of open architectures with layers of abstractions that will enable future integration of component replacements (and this reduces the risk of component obsolescence).

References

Clarke, A. (1961). *Profiles of the future.*

Dijkstra, E. W. (1982). *Selected writings on computing: A personal perspective*. , Berlin, Germany: Springer-Verlag.

DoD. (1997). C4ISR Architectural Framework*.* Washington, DC: Author.

DoD. (2003). DoD Architectural Framework (DoDAF). Washington, DC: Author.

DoD. (2009*).* DoD Architectural Framework (DoDAF; Rev. 2). Washington, DC: Author.

DoD Joint Task Force. (2004). *Modular open systems architecture (MOSA) program manager's guide.* Washington, DC: Author.

Meier, M. (1998). Architecting principles for systems-of-systems. *Sys Eng, 1*, 267–284.

Meier, M., & Rechtin, E. (1997). *The art of systems architecting*. New York, NY: CRC.

Mill, J. S. (1843). *A system of logic, Book III*.

System Modeling Language (SysML) Specification. (2006).

Naval SoS SE Guidebook (Vol. 1, Ver. 2.0). (2006).

Zachman, J. (1987). A framework for systems architecture. *IBM System Journal, 26*(3), 454–470.