What's Wrong with DoD's So-Called Information Architectures And What We Ought To Be Doing About It

Rex Buddenberg
Naval Postgraduate School
Monterey, Ca 93943
http://web1.nps.navy.mil/~budden
March 2000

What are we after?

Interoperability. Components of multiple programs, services and allies need to work together. This paper accepts without further argument that improved information systems can improve combat power, including the deterrence capability that combat power confers. This paper also recognizes that the list of other programs that one must be interoperable with is indefinitely long. Therefore, an open-ended solution is a fundamental requirement.

This leads us directly to the need for an architecture. We need a common design vision that each program manager or procurement agent works to fit into. Defining and articulating such an architecture is my purpose here.

Large Information Systems.

We have evolved and codified a program manager doctrine for executing large programs, both in the military and elsewhere in government. This methodology concentrates enough authority in the hands of the program manager that he can execute the program and deliver the product. For industrial age, platform-oriented weapons systems, this methodology works tolerably well.

But as we consider how to build large[1] information systems, we find that the conventional program manager metholology does not work - at least not without some modification. Information systems cut across multiple platforms. Indeed, interoperability impacts an indefinitely large number of diverse platforms when we consider multiple services and allies as within the scope of 'enterprise wide'. It is not conceivable that we would give any program manager that much authority. Further, if we tried, the mega-program would be so large that it would collapse of its own weight. Indeed, the landscape is littered with far less sizable information system programs that have failed.

We cannot deliver an enterprise-wide information system by simply assigning the job to a single program manager.

[1]Buzzword: enterprise-wide.

On the other hand, it appears entirely practical to achieve the end in two stages:

- First, require all information systems to be cross-program interoperable. How to achieve this is the subject of this paper.

- Second, include the interoperability requirements in each program manager's charter.

A large enterprise-wide information system consists of the deliverables of innately interoperable, but modest programs. Each program, because of its modesty, is practical; their sum is the enterprise-wide information system.

What's wrong?

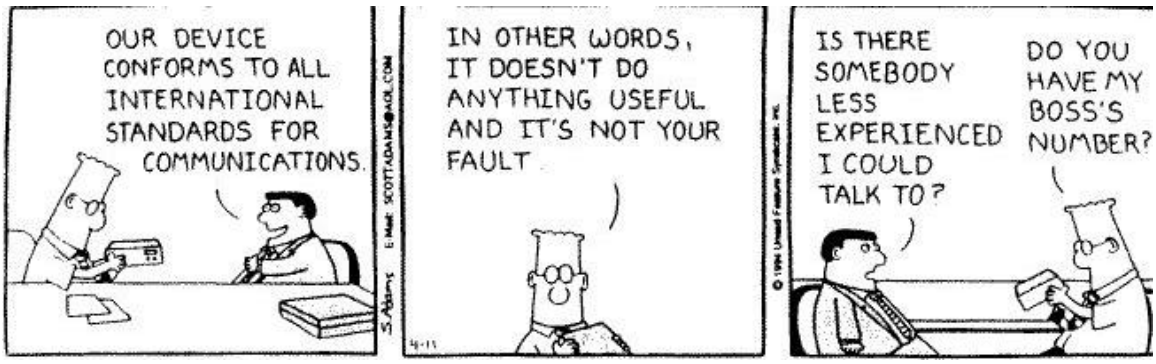
We must deconflict the definitions of architecture.

We have voluminous documents with the term `architecture' in their titles in DoD. But none of them are truly architecture and they don't provide the tools that the CIO needs to reach the lofty interoperability goals desired. Dealing with these documents and their committees over the years has convinced me that we are making this problem far harder than it needs to be. And much of the time we're shooting at the wrong target.

I include among this rogues gallery of architecture wannabes the several `Joint Tactical Architectures' , the Technical Architecture for Information Management (TAFIM), the C4ISR Architecture Framework, the Core Architecture Data Model, etc.; the JTAs are sacks full of standards but there is no architecture. Dilbert illustrates this point perfectly. The rest are simply chasing the wrong definition.

Further, polluting the landscape with multiple definitions of `architecture' - as in operational, technical and systems architectures - lends little clarity. It may help with infrastructure, but not much with architecture. This muddle badly confuses program management with architecture.

Dilbert and the Salesman[2]



How did we get into this mess?

In order to chart a way out, it helps to know how we got in. Part of the problem is undiciplined definition, part is committee dynamics.

Undisciplined language. We can all agree that an architecture is necessary[3], but definitions of architecture vary widely. Indeed, the US Department of Defense recognizes three definitions that are pursuing at least two divergent objectives.

The best, and applicable, definition I've found for `architecture' is "Design. The way

---

[2]Scott Adams. My clipping says 4/11 c 1994, reprinted in The Dilbert Principle, Harper 1996, p.216.

[3]In the context of the FY00 DoD Appropriations Law contains the following: "SEC. 8121. (a) REGISTERING INFORMATION TECHNOLOGY SYSTEMS WITH DOD CHIEF INFORMATION OFFICER.--After March 31, 2000, none of the funds appropriated in this Act may be used for a mission critical or mission essential information technology system (including a system funded by the defense working capital fund) that is not registered with the Chief Information Officer of the Department of Defense. A system shall be considered to be registered with that officer upon the furnishing to that officer of notice of the system, together with such information concerning the system as the Secretary of Defense may prescribe. An information technology system shall be considered a mission critical or mission essential information technology system as defined by the Secretary of Defense.

"(b) CERTIFICATIONS AS TO COMPLIANCE WITH CLINGER-COHEN ACT.--(1) During fiscal year 2000, a major automated information system may not receive Milestone I approval, Milestone II approval, or Milestone III approval within the Department of Defense until the Chief Information Officer certifies, with respect to that milestone, that the system is being developed in accordance with the Clinger-Cohen Act of 1996 (40 U.S.C. 1401 et seq.). The Chief Information Officer may require additional certifications, as appropriate, with respect to any such system."(2) The Chief Information Officer shall provide the congressional defense committees timely notification of certifications under paragraph (1). Each such notification shall include, at a minimum, the funding baseline and milestone schedule for each system covered by such a certification and confirmation that the following steps have been taken with respect to the system:

(A) Business process reengineering.

(B) An analysis of alternatives.

(C) An economic analysis that includes a calculation of the return on investment.

(D) Performance measures.

(E) An information assurance strategy consistent with the Department's Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework."

components fit together."[4] This definiton is the dictionary term that is useful for cross-program interoperability purposes and is prescriptive in nature.

The other legitimate dictionary defintion of architecture that is descriptional, as "architecture *n.* ... 3. A style and method of design and construction <Roman architecture>"[5]. 'All the buildings at the US Coast Guard Academy are of Georgian architecture.' This is a taxonomic, not prescriptive, definition so it's useful for describing a style, but does not help us in our quest for interoperability. This paper does not use this definition, although both the 'systems architecture' and 'operational architecture' official DoD definitions root to this version[6] (but not the IEEE Dictionary definition of 'systems architecture', confusingly enough).

Beyond these definitions, we have a lot of just plain sloppy terminology abuse. These things are blamed on the architect, but they are not architecture:

- - infrastructure ('system of public works')
- - architecture in the plural (usually descriptions of infrastructures)
- - provisioning ('allowance parts list', 'range and quantity of items', configuration)
- - systems engineering (getting the right boxes)
- - machine language dictionaries, as in 'instruction set architecture' for 'Intel Architecture' chips or MilStd 1750 processors.

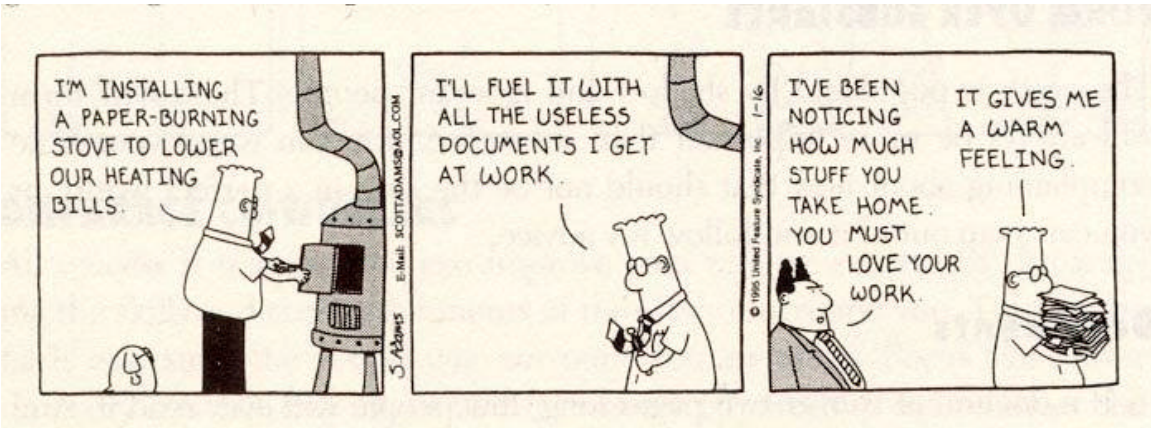
The prescriptional, design, definition of architecture - as the means to interoperability - is the proper area of concern to the architect (CIO). These other definitions of the term properly belong within each program manager's domain.

[4]From FOLDOC, Free On-line Dictionary of Computing, http://foldoc.doc.ic.ac.uk. My next favorite one is in the IEEE Standard Dictionary of Electrical and Electronic Terms. You have to look under S for system architecture: "The structure and relationship among the components of a system."

[5]Webster

[6]The DoD definitions of `architecture', `technical architecture', `operational architecture', and `systems architecture' can all be found in the DoD Joint Tactical Architecture (http://www-jta.itsi.disa.mil/jta). These fail the Occam's Razor test, and you should beware anytime anybody uses the A word in plural as does Section 3 of the C4ISR Architecture Framework - we need a single information architecture.

Committees. All of the existing `architecture' documents are a product of committees[7]. Enter the natural bureaucratic, committee tendencies: reach a common denominator that all on the committee can agree upon. Motivation was less to do something good; more not to do something bad. As a result, we got deforestation without compensation[8].



Some of the standards are mutually contradictory and unnecessarily complicated, but that's secondary: none of these committees produced anything so risky as a real architecture. Ironically, we seem to have produced the inverse of the crypto system that is described by Lt Keefer to Ens Willie Kieth in The Caine Mutiny: "The Navy is a master plan designed by geniuses for execution by idiots[9]." We can all agree that standards are a necessary part of architecture. But the various Joint Technical Architectures are mere collections of standards - not architecture[10]. The committees tended to work on the things they knew how to work on - compendia of standards - rather than the things that needed to be worked on. This well-meaning work has diverted us from the main objective of an architecture.

Measures of Effectiveness.

Settling on the applicable definition of architecture is the first step. But it won't do any good to settle on one if there's no payoff, so we need to tie the definition to some profits first. Here's my list:

- simple. Clauswitz was right. So was Occam. If an architecture requires two inches of paper to specify, it's unworkable.

- minimal and extensible. A `kitchen sink' approach routinely sinks of its own weight[11]. Good Sailors will find creative ways to use the tools provided and that

[7]Some were called IPTs: Integrated Product (or Process) Teams. They're still committees.
[8]Adams, Scott, The Dilbert Principle, Harper 1996, p76. Original print 1/16/1995.
[9]Wouk, Herman; Doubleday & Co, 1951. This quote appears on p.120.
[10]DoN CIO's IT Standards Guidance is more honest in this regard. It explicitly says `standards guidance', not architecture. Standards guidance is what it delivered.
[11]The Internet Engineering Task Force has demonstrated repeatedly that a minimal protocol that is extensible is a powerful and practical tool. The Multimedia Internet Mail Extensions (MIME) is an

creativity (which can translate directly into combat potential) needs to be enabled.

- scaleable. Local activities must fit into a larger whole. And the whole is likely to grow somewhat larger than originally planned[12].

- real. We can specify an architecture that requires no uninvented technology to implement, without compromising the extensibility to incorporate new technology as it appears.

- platform and function independent. A sensor module is a sensor module; whether it is carried by a ship, tank, individual, aircraft, or is fixed to a structure is not material to the architecture. Further, the architecture should be neutral regarding what the sensor does.

Real architecture.

Architecture is "Design. The way components fit together."

Applying the definition. All information systems, whether tactical or administrative, can be decomposed into Sense, Decide and Act functions, interconnected by communications[13]. Complexity can be explained by nesting and chaining[14]. `Bad' architecture can generally be identified as

1)situations where the module boundaries are incorrectly articulated - we got the boxes mismodularized, and

2) situations where complexity doesn't cleanly nest[15].

The core of architecture must be a modularization methodology, implemented by a standard set of interfaces for these three classes of boxes and the network that defines

---

excellent example: there are only a handful of top-tier standard data elements: text, image, audio, video, etc. But implementors are free to add and register second-tier elements to suit needs. MIME, as a result of its flexibility and extensibility, has become the means both for e-mail attachments and World Wide Web object definitions. By contrast, the X.400 e-mail standard was unblessed by concurrent implementation and proved far too complex to be usable. The internal conflicts only became apparent when programmers attempted to actually build X.400-compliant products. (Several of those implementors gave up in disgust and invented MIME).

[12]'Enterprise-wide' is indefinitely large. The Internet itself is my best example. None of the designers of IP thought that we'd ever exhaust a 32-bit address space and none of the designers of RIP ever thought that the diameter of the Internet would exceed 16 routers (the maximum hop count supported by RIPv1). The Internet has successfully scaled by several orders of magnitude.

[13]You can mentally apply the sense/decide/act model to garden variety information systems like the paycheck system - the personnel clerk is acting as the sensor, collecting data. Printing the paychecks is the Act module. The similarity to the late Col John Boyd's OODA loop is not entirely accidental.

[14]Chaining: in the case of the paycheck illustration, output (action) of this loop becomes input to your bank's sense/decide/act loop.

[15]These situations are most closely illustrated by Edsgar Dykstra's indictment of the GoTo statement in FORTRAN. He held that the infamous GoTo was the progenitor of unstructured spaghetti code.

how they fit together. If we succeed, then sensors bought from one program can be interconnected with decision support modules provided by a different program. Without knowing a priori what needs to fit together. The remainder of this paper outlines these interfaces.

The elements.

We need a small handful:

- A network.
- An interface definition for attaching end systems to the network.
- An interface definition between end systems.

Network Centric[16].

A coherent architecture must use a common network structure. To wit: internet technology. The Internet is a complex system. But most of the complexity can be masked and is not important to the end systems that we attach to it. The border of the network is a local area network.

The fundamental rule of network centric is that all end systems (sense, decide and act) attach to the network, never directly to each other. This allows for scale and redundancy - replication becomes easy. And it allows the end systems that are products of multiple programs to be attached to the same network.

We also place ourselves in a position where we can leverage commercial internet technology. We can adapt to meet our needs rather than designing from the beginning. Further, most of the scale problems likely to be present in a military intranet will have been found and solved in the commercial world first.

A few assumptions about the network need to be specific:

- within the network cloud we have e-mail Message Transfer Agents and
- network monitoring capability that uses SNMP.
- We need a PKI[17] and
- must support QoS services.

The first three of these services are actually delivered by end systems; the last is delivered by the routers in the network. The network can and should support many other services but these are the ones that are architecturally significant. This is an expansive

[16]See http://web1.nps.navy.mil/~budden/lecture.notes/net_centric.html for a fuller treatment.

[17]Public Key Infrastructure, in turn, implies a directory structure. This directory may do many things, but the architectural requirement is that it authentically serve public keys. Resistance to denial of service attacks, link crypto, low probability of intercept and detection are all issues that belong inside the network cloud; they are not of architectural concern to end systems attached to the network.

definition of the internet and its services, but nothing that isn't available today.

We now also have means for simplifying the communications requirements: all the programs within the network cloud, such as satellite communications programs, wireless LANs, link crypto, spread spectrum communications, etc., should be optimized to haul IP datagrams. These issues and components are invisible to the end systems attached.

Good network citizens: end system < - > network interface.[18]

The first architectural rule is that all end systems attach to the network; never directly to each other. End systems can be easily attached to an internet ... providing that these end systems qualify as Good Network Citizens[19]. A Good Network Citizen should have:

- - a LAN interface
- - an 'enveloping' interface.
- - a management interface.
- - a PKI-base capability to authenticate itself
- - an ability to request QoS services if best-effort delivery is not adequate.

This is deliberately sparse. Details of protocol stacks are not explicit and the architecture is neutral regarding the operating system used inside a module. However, the specification is sufficient. And if something is missing, the hooks are all there to do modifications without wholesale changes to the end system.

LAN Interface**.** In the vast majority of cases, a 10 Base-T ethernet interface will do. Few end systems can produce enough volume of data to significantly load a 10M ethernet. Reasonable alternatives include other routable networks[20].

Looking briefly inside the network cloud, we can easily translate from one of these LAN interfaces to something else by use of routers and bridges (switches). So more detail here will complicate the picture, but not improve it.

Enveloping interface. End systems should deliver data in MIME-compliant e-mail. If this interface component meets the needs, don't look farther[21]. You get these things:

---

[18]See http://web1.nps.navy.mil/~budden/lecture.notes/good_net_citizen.html for a fuller treatment.

[19]The 'toaster' moniker is Internet in-crowd. Early demonstrations of SNMP demonstrated the flexibility to manage devices other than conventional computers and routers. One of the legends is the SNMP-managed toaster at an Interop exhibition.

[20]I can agree that the RJ-45 physical connector for 10Base-T ethernet is a bit flimsy for exposed locations. But connectors can be beefed up; this is a packaging, not an architectural issue. Other reasonable alternatives include routable LAN technology such as FDDI or FiberChannel. But eschew serial interfaces (RS-232, Mil-1379) and non-routable data links (Mil-1553, Link 11).

[21]E-mail is an 80% solution. One place to look farther, especially for information dissemination tasks, is world wide web technology. Coupled with multicast and distributed servers, this can be a very powerful

- - the data will tend to be organized in fields[22] in order to meet the MIME definitions. This allows the data to be poured into a database at the destination.

- - we've gotten to the point where data element standardization can be performed. This part of the definition definitely does not address or solve this important problem, but it sets you up for it. See the next section.

- - it's hard for the vendor to provide you with what you asked for without also providing you an e-mail user agent, and a TCP/IP protocol stack in the end system. Whether you asked for it or not.

- The e-mail interface affords us enormous flexibility. If we need to direct the data to an unplanned-for destination, all that must be done is to amend the e-mail addressee list (or an alias in an MTA somewhere[23]).

Management interface**.** Since we want the data remotely, we need to be able to `read the dials' and 'twist the knobs' remotely too. The proper vehicle is a Simple Network Management Protocol interface. Specifically in the form of the Management Information Base (MIB).

Public Key Infrastructure. With the growth particularly of remote sensors, the need to authenticate the data to preclude spoofing will become quickly evident. With stovepipe information systems, membership on the network implied authenticity; this assumption now disappears. And the data from some end systems requires confidentiality in addition to authenticity. These requirements can be achieved by upgrading the e-mail user agent in the end system to support S/MIME and by provision of a Public Key Infrastructure within the network support.

Quality of Service (QoS). If best effort delivery (the norm in the Internet today) is adequate for the end systems in a particular information system, then skip this paragraph. If differential services or bounded delay delivery[24] are requirements, then the interface needs to support Differential Services or RSVP, respectively.

The purpose of this interface definition is to allow an end system, whatever its function or location, to be attached to a network. More importantly, it's about all that an end system designer needs to know about the network in order to function effectively - we are partitioning the knowledge requirements in a way that allows the divisions of labor that

---

and efficient capability. Other internet applications are acceptable but beware those that must penetrate firewalls and cannot be easily proxied. Also beware those that cannot provide authenticity and confidentiality via Public Key Infrastructure.

[22]The e-mail folks call these 'body parts'; the database folks call them 'fields'.

[23]MTAs also do (incomplete) multicasting. If you send e-mail to multiple addees that are all served by the same MTA, your MTA will only send a single copy to that site; replication occurs there. The incomplete part (technology exists, implementation doesn't) is that we don't yet have the practical mechanism for multicasting to multiple MTAs (as in a mail server on each ship). All this is an efficiency issue, not an architectural one.

[24]Bounded delay delivery or deterministic delivery is disciplined language; 'real time' isn't so I don't use it.

typify industrial engineering.

The first big step. It is worth noting that even if we fail to achieve true interoperability across systems (the modularizaiton issue in the next section), we have taken a big step forward in getting even the non-interoperable components onto a common network foundation. Getting just this far has profit.

Modularization: end system < - > end system interface[25].

Making the modularization match the sense, decide and act taxonomic functions is the second big step. This transforms the taxonomy into an architecture. Three rules apply:

- make the functions of sense, decide and act match the module boundaries. Avoid, in particular, placing single sensor integration functions in the decision module[26]. Modularize the end systems consistently to increase the probability that a sensor originally part of one program can provide data effectively to a decision support module that was part of another.

- Nest cleanly. The best illustration is in structured software langages that make it very difficult for a subroutine to return to any place other than where it was called from. Clean nesting allows reuse of modules and building of arbitrarily complex information systems.

- Chain properly. Ensure that the act function (not the decide) of one system represents the sense function of the next system. Recognize sense-decide-decide-act chains not as chaining at all, but as poor (but often necessary) halfway steps that should only be indulged in to accommodate legacy.

My classes have produced case studies of information systems that followed all the standards checklists, and still produced Bad Architecture. Most of these had to do with poor modularization between end systems, such as attempting to perform Sense functions in the Decide module. The parts had the right interfaces, but they were misshapen. In addition to the specific information system implementation being klutzy, all the opportunities to take output from one sensor and pipe it into a different decision support module were lost.

Proper function-to-module mapping in sensors might also be characterized as complete data reduction. Examples might be integrating noisy data over time to improve its quality (scan-to-scan correlation in radars is one instance). Another facet is to have the radar produce its data in a tabular form (track reports) that are easily exported to a common tactical picture database rather than in the customary form of video[27]. Our architectural

---

[25]See http://web1.nps.navy.mil/~budden/lecture.notes/it_arch/modularization.html for a fuller treatment.

[26]We have a lot of legacy where the first order fusion is done in a central computer; the legacy reason has to do with the expense of CPU horsepower. That's clearly not the case today.

[27]Note that a side effect of improving the ability to integrate the data is a reduction in the number of bits that

rule of thumb is that Sense modules should do all the first order fusion in situ.

The data fusion community produced a taxonomy and dictionary that perhaps has some value here. They define first order fusion as single-sensor fusion - analogous to the the previous paragraph. Second order fusion is similar-sensor fusion such as producing a tactical plot using data from multiple radars or producing a comprehensive paycheck database from input data from multiple (but similar) payclerk sources. Third order fusion is heterogeneous sensor fusion. An example might be overlaying the composite radar picture with meteorological data. For architectural purposes, these complex topics can be reduced to a single thumb rule: second and third order fusion belongs in the Decide module. Trying to value-add it into a Sense module will inhibit scaling and is consequently not clean architecture.

Data element standardization. Once we do the modularization - but not before - we can tackle the data element standardization problem with some discipline. Now we can standardize the data elements either by using the repository system or some other means[28].

Conclusion.

This paper has prescribed, at the top level, an information system architecture. It:

- is top down. Complexity can be decomposed. The overall model is simple.
- is arbitrarily scaleable. We can make the infrastructure as large and complex as needed as long as we maintain the building blocks.
- leverages commercial internet technology.
- allows incremental program building, controlling risk.

The part of the architecture that I have described that cannot be reduced to checklists is the modularization problem. The best I can offer are the thumb rules; beyond that subject matter expertise - the domain of the program manager - is still important. The architecture cannot be entirely designed for execution by idiots.

A tacit part of this prescribed architecture is that it minimizes the interference that the architect poses to the multiple program managers that he must harmonize.

---

must be transmitted across the network.

[28] Another of the publications that has architecture in the title but isn't is the Core Architecture Data Model (CADM). It becomes useful once we get the modularization problem solved, but not until. Precisely how the data element standardization occurs is not a primary issue, as long as it happens. There are at least two usable approaches. One is the procedure-oriented approach extant in DoD today, using the data repositories for storing metadata. Another is the object-oriented approach of CORBA and netCDF where the metadata is included with the data itself.