Toward an Interoperability Architecture.

Rex Buddenberg
Naval Postgraduate School
Code IS/Bu
Monterey, CA 93943
budden@nps.navy.mil

Abstract

In moving toward an interoperability architecture, the concept of network centric is a step in the right direction - all modules connect to the network, not to each other. And a handful of good network citizenship rules provide a syntactical guide for attachment. From the point of view of the network designer this is sufficient - we have enough to build internetworks for the common good. The continued burgeoning of the Internet constitutes an existence proof.

But a common networking base is insufficient to reach a goal of cross-system interoperability - the large information system. Many standardization efforts have attempted to solve this problem, but appear to have lacked the necessary scope. For instance, there have been many efforts aimed at standardizing data elements; these efforts, if followed through, yield some gains, but never seem to quite reach the interoperability goal. If we are to truly erect an interoperability architecture, we need to broaden the scope. This problem of cross-program, cross-service and cross-ally interoperability requires that we agree on the *what* of modularization, not just the *how*.

This paper is aimed at framing the interoperability architecture problem.

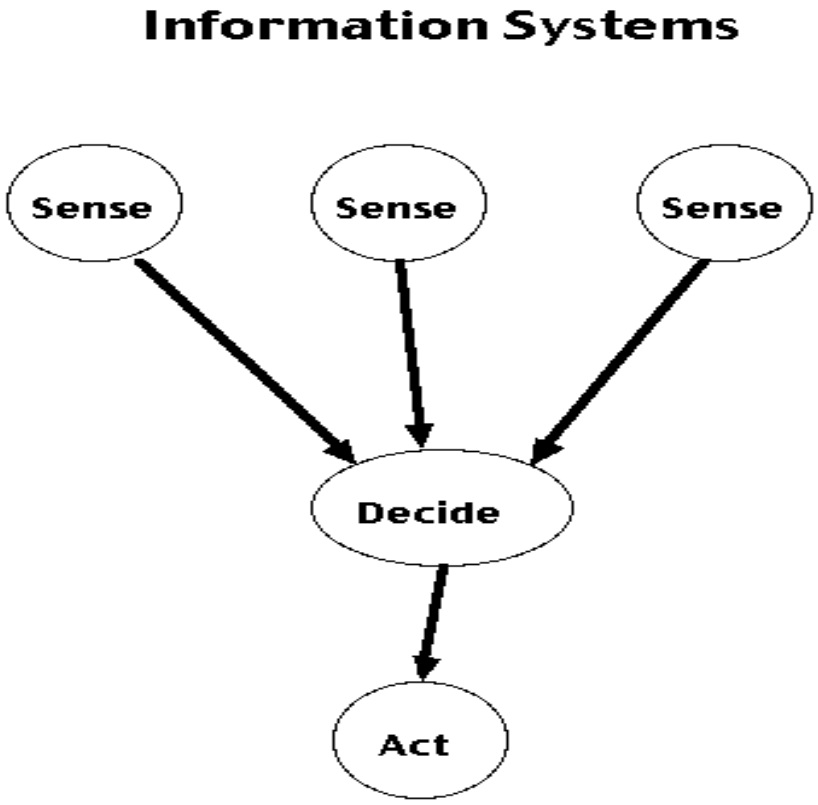
On modularization

The core of architecture - the way things fit together - is a sense of modularization. This is the part of the problem that is perhaps the least mechanical and requires judgment. Experience, no doubt, helps. Architectural conformity must be traded off against other desired characteristics. The objective is that modules become inherently interoperable so we have components delivered by multiple programs that can be assembled for particular tasks.

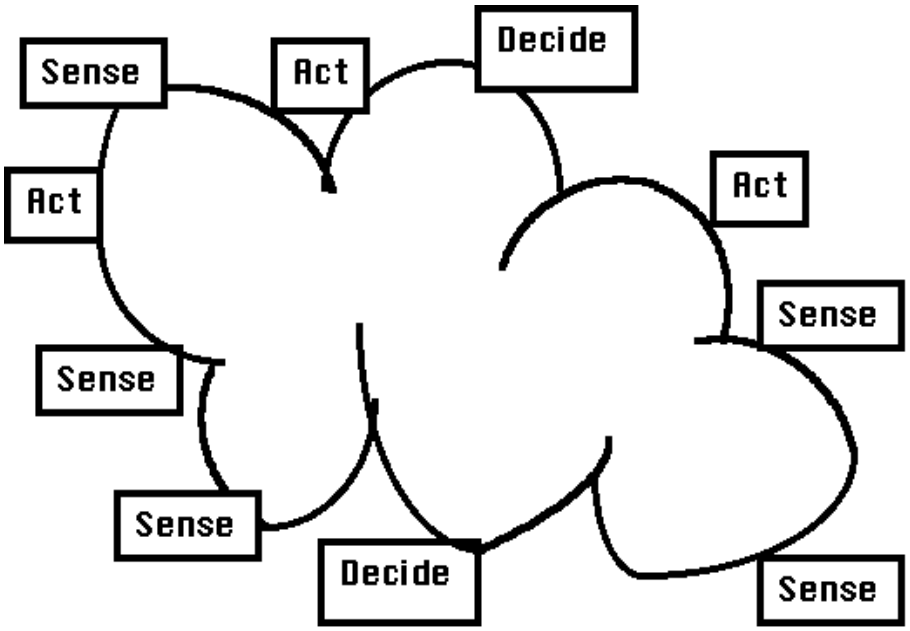
Prerequisite – network centric.

The systems engineering concept of network centric is provision of a single network and all information system functional modules connect to the network, not to each other. From the point of view of the network designer this is sufficient - we have enough to build internetworks for the common good. This is necessary but insufficient to reach a goal of cross-system interoperability - the large information system.

The non-scaleable approach maps neatly onto the existing program manager way of doing business. A program is responsible for designing and delivering a complete information system, including the communications between the functional nodes. While this delivers programs, it delivers stovepipes.



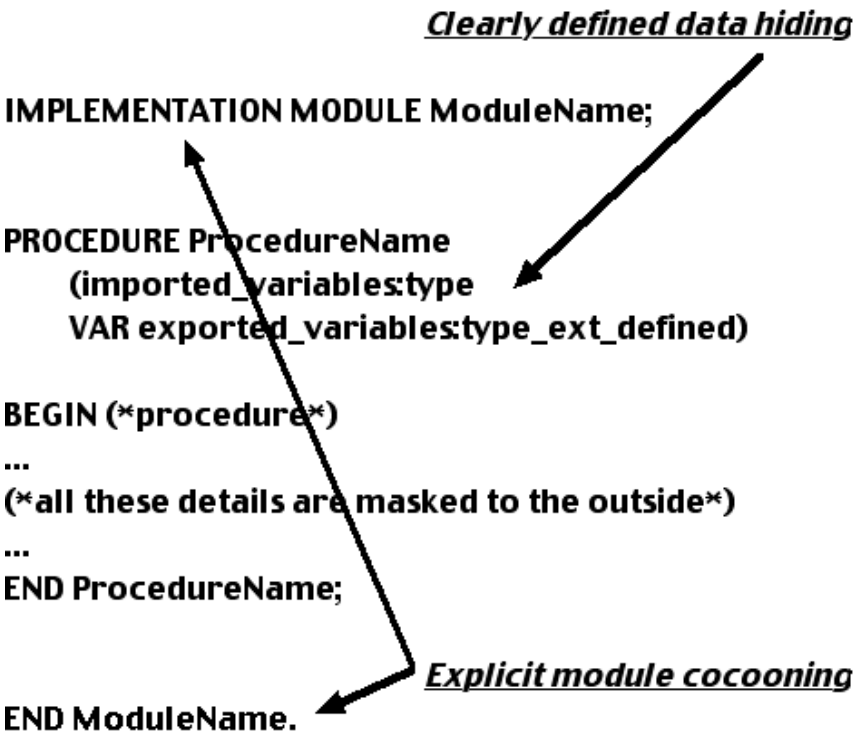
A better division of labor is to have a program that delivers network plumbing for multiple applications. And then multiple program managers perform application engineering that rides on the common networking infrastructure.



There are certainly technical and organizational issues regarding how we handle security, availability, and quality of service control within this network, but the network is scaleable and the problems solvable. But it's also abuseable by the application engineers.

An example of the problem is easily seen in software engineering.

Syntax of Modularization



This code fragment illustrates what we have and what we lack. We have good modularization syntax rules, and we have ever since the advent of Pascal[1], with its carefully constructed data hiding rules as a replacement for easily-abused FORTRAN and BASIC. This syntax is the computer programming language equivalent of the S/MIME recommendation in Good Network Citizenship[2]. What we lack is agreed upon content of modularization: what functions go into which module? We have seen that information systems can use all the right interface standards and still be constructed so that they both perform poorly by themselves and offer a poor basis for cross-program interoperability. The standards checklist is right but the system simply doesn't qualify as Good Architecture. There's more to the problem[3].

We need to develop a feel for what belongs in the Sense module, the Decision Support module and the Act module. This modularization horse sense then needs to be applied across multiple programs. That consistency will then allow the Sense module from one program to feed usable data to a Decision support module of another. Observing a

---

[1] This particular example is Modula-2, but all modern procedural and object oriented languages qualify.
[2] http://web1.nps.navy.mil/~budden/lecture.notes/good_net_citizen.html
[3] C4ISR Architecture Framework and Core Architectural Data Model are almost entirely silent on this subject

taxonomy[4], we can see that all information systems are simply assemblies of Sense, Decide and Act functions strung together with communications. And we can decompose large information systems into these functions, explaining complexity with nesting and chaining. But a taxonomic description is only a first step to an architectural prescription.

Match module to function

The first - and apparently the most important - rule to grapple with is that sense, decide, and act *modules* must match the sense, decide, and act *functions*.

Requirements: recall that platform independence is one of our goals. And since most operational military systems are connected by radio segments of our network, bandwidth will be forever limited[5], both in terms of bits per second and quality of service (latency, jitter, interactivity). This drives us to a need to couple Sense and Decide modules as loosely as possible[6].

Get all the sense functions (single sensor integration, data reduction) into the sensor and out of the decision support module. Perhaps the most ubiquitous interface that needs to be gotten right - and is most often fouled up - is that between the Sense and Decide modules. Sense modules frequently do too little, requiring the Decide module to do too much[7]. This catches up with us when we want to send sensory data to a different decision support box than the one originally envisioned.

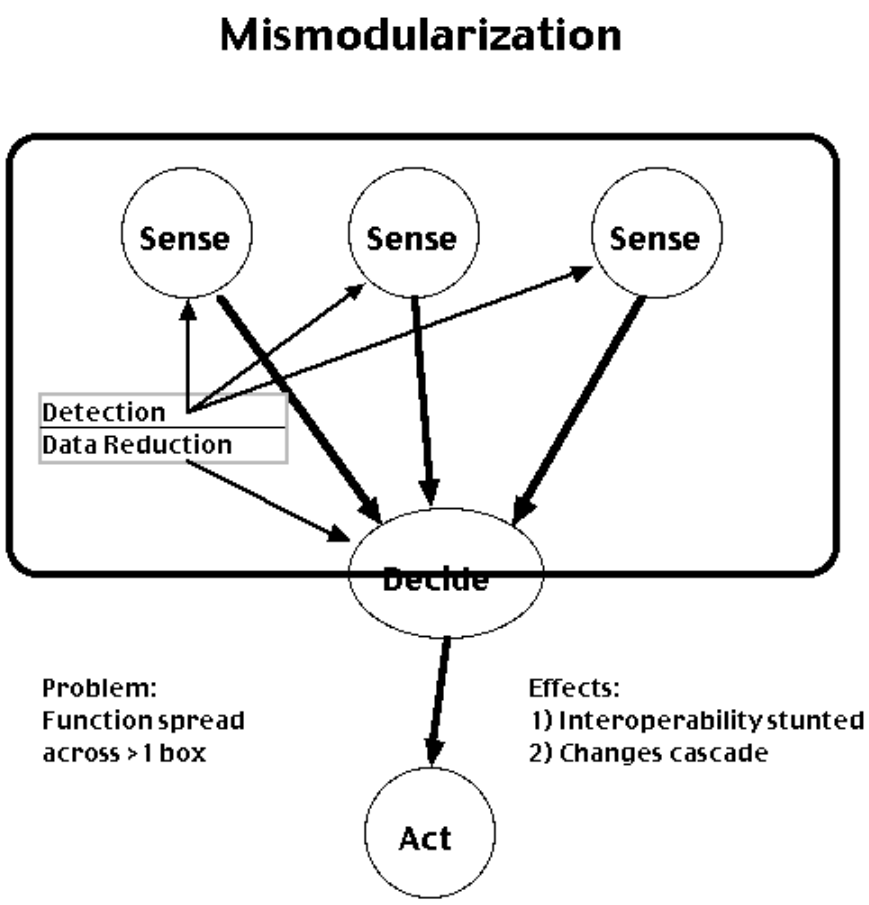
---

[4] Taxonomy is a system of classification. It is descriptive, much as the 'style' definition of architecture is descriptive. While taxonomy is helpful, we haven't succeeded at the cross-program interoperability goal until we progress from descriptive taxonomies to prescriptive architecture. This is the root reason that the Operational Architecture and Systems Architecture definitions in the C4ISR Architecture Framework are irrelevant to interoperability.

[5] Yes, capacity is increasing in both radio-WANs and wireless LANs, but that capacity will always lag that in wired LANs and WANs by several orders of magnitude.

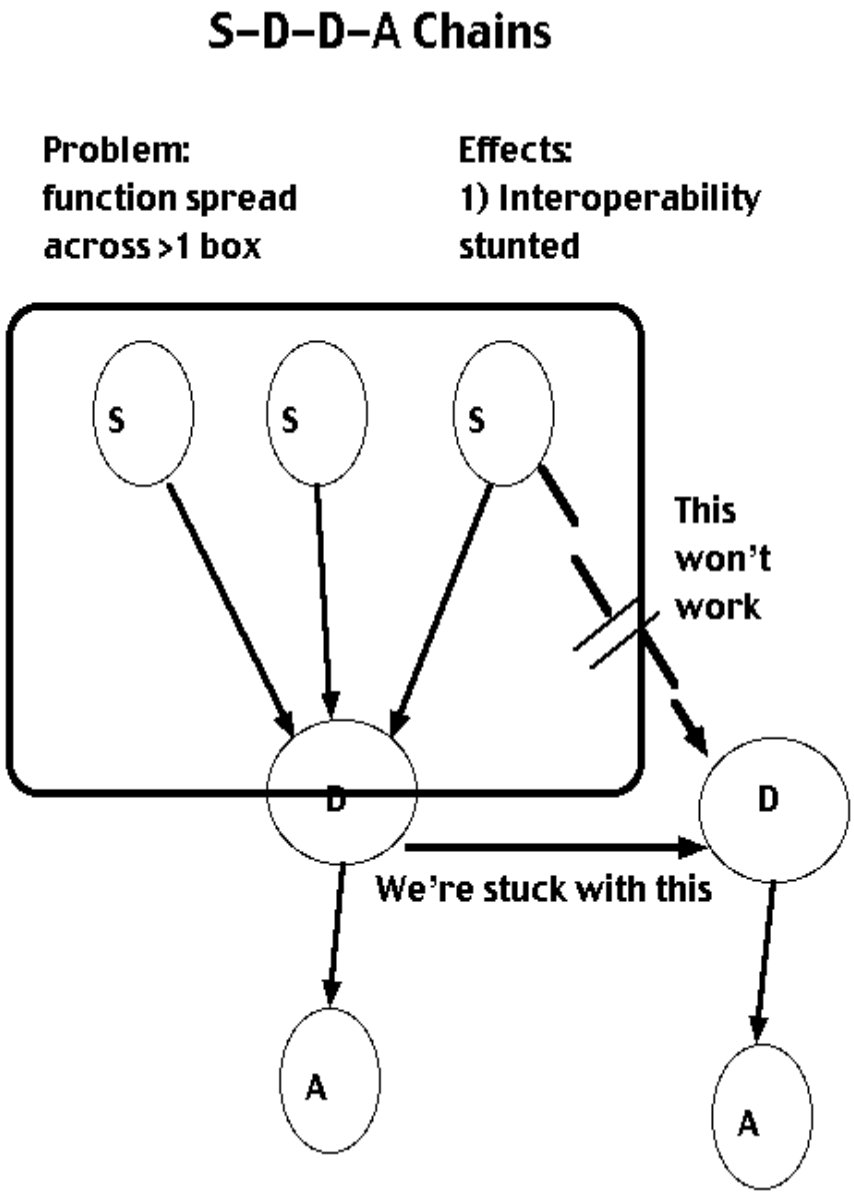
[6] I'm again borrowing lexicon - tight and loose coupling - from the software engineering literature.

[7] This often results from a 'mainframe mentality' deriving from the days when CPU horsepower was expensive so you didn't put any of it into the sensors.



A reliable symptom of poor modularization is the need to feed a new decision support system from another decision support system rather than it's contributing sensors. In other words, we see S-D-D-A chains instead of S-D-A ones. S-D-D-A chains may be a proper intermediate step from stovepipe legacy to properly modularized systems, but it is an improper destination. This intermediate step should be characterized by a modular boundary, using the good network citizenship rules, between the legacy and new decision nodes, pending the obsolete-equipment replacement of the sensor and legacy decision node[8].

[8] Global Command & Control System (GCCS) is made up of somewhere around three dozen S-D-D-A chains. This is clearly a step forward from non-integrated systems, but clearly represents at best a halfway point toward good modularization.



We have evidence[9] that violation of this rule can harm a single program, never mind cross-program interoperability issues. The most important aspect from an interoperability perspective is that we cannot take the data out of one sensor and pipe it into a decision support module other than the one in the original system because there is missing sensory function in the original system's decision support node. Another symptom of the same problem occurs when we want to update the decision node in the original system - we run into the cascading effects problem.

Absorbing legacy systems where we are inheriting poor modularity again makes the problem harder. Modules are often misshapen where the sense function, for example, is

[9] See http://web1.nps.navy.mil/~budden/lecture.notes/it_arch/modularization.html for some.

spread across and intertwined with the decision support one. This usually results in the necessity to take sensor feeds out of a decision support box where clean architecture would indicate the data should come from a sensor module directly.

Symptoms and reasons. Symptoms of tight coupling between Sense and Decide nodes include a need for high-capacity, low latency communications between them[10]. These requirements are hard to meet in internets involving radio segments, and one of the rationales for separate, stovepipe communications. Often the link is effected with a serial line[11] which means that adding more sensors or decision support nodes immediately encounters the n**2 problem - the implementation isn't scalable. Many existing systems got this way because they were originally designed in an era when CPU horsepower was much more expensive than today ... or designed when the culture of a mainframe priesthood persisted, even though the technological drivers had changed.

We need to handle the problem of multiple-sensor data fusion before we can conclude this section[12]. The problem shows up in several military systems. In anti-submarine warfare, several sonars each may be receiving a bit of stimulus indicate the presence or whereabouts of a submarine - this data must be fused, the result being a track. Before the track is gained, all tactical decision making is moot. Tightly coupling all the sonars on a single ship bears little interoperability architecture penalty (but it may bear a life cycle update one). The penalty occurs with close coupling to off-ship nodes.

Another example is the navigator's problem: it comes in two parts. The first part is figuring out where you are - position fixing. The second is voyage planning - what to do about it. In both of these cases, the sensor and the multi-sensor fusion process are necessarily closely coupled. On the other hand, the decision support function - what one does with the track data - is much more loosely coupled. Further, the decision support module typically has other data input (in both of these cases cartographic data is an example). Therefore, the proper modularization boundary is between the multi-sensor fusion function and the decision support one[13].

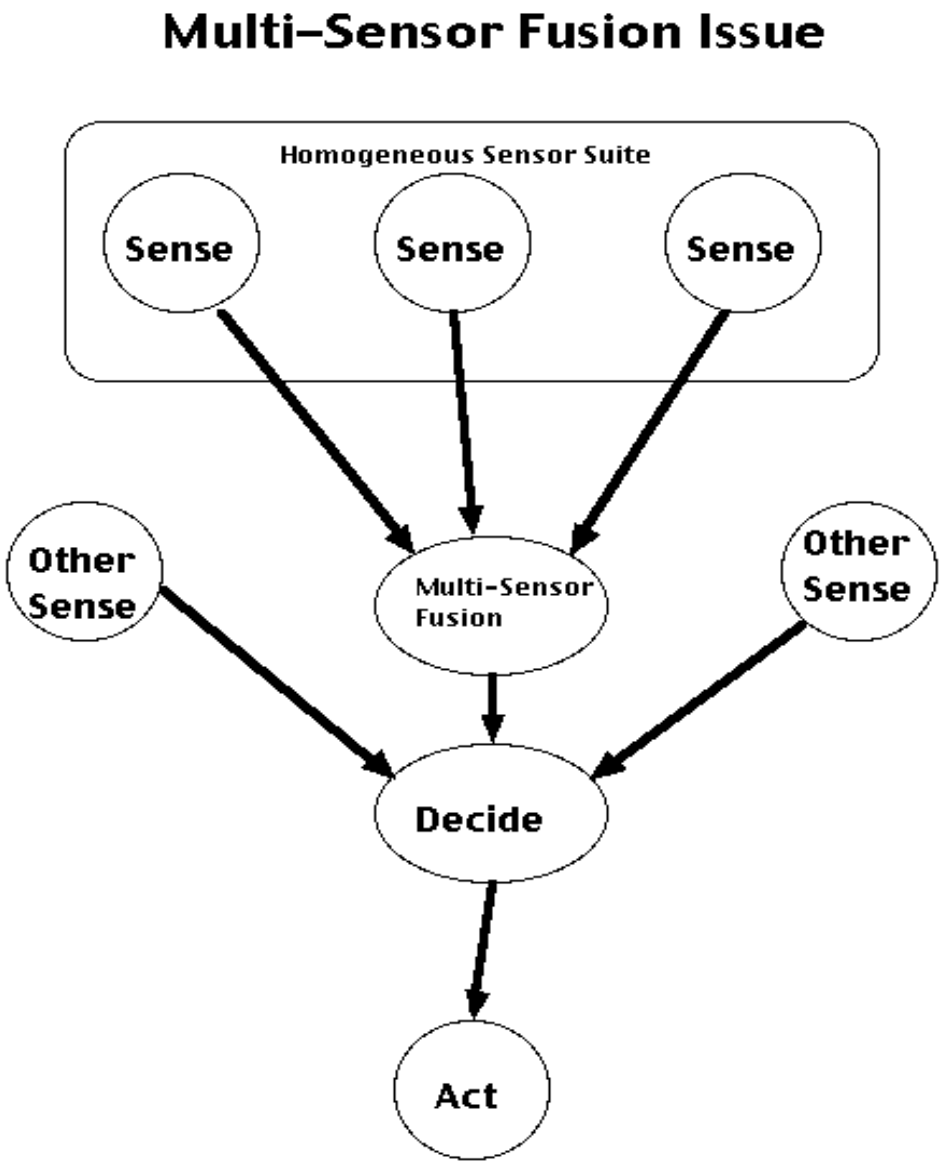
---

[10] Far better to have Sense modules timestamp their data and achieve synchronization using the timestamps rather than to rely on low latency.

[11] Aka point-to-point or trunked communcations lines in the radio environment. Very spectrum inefficient.

[12] In the Data Fusion Taxonomy, one of the few cases of prior art for this thinking, multiple-sensor fusion maps partially onto 'second order' and 'third order' fusion.

[13] It may make sense to carefully preserve the sensor-multi-sensor fusion modularizaiton boundary, but the justification is more intra-program growth (traditional P3I) rather than cross-program interoperability. This is especially so if multiple vendors are involved, or if the sensors are distributed across multiple platforms.



Mismodularization appears to be the most ubiquitous and pervasive engineering problem inhibiting interoperability[14]. We need to briefly discuss nesting and chaining, but it appears that poor chaining is simply another manifestation of S-D-D-A chains. Poor nesting doesn't seem to occur very often. But they are the means of adding complexity to information systems so, for completeness reasons, we need to treat them.

Nest cleanly

One example of complexity is nesting of one sense-decide-act cycle inside another. The basic rule to follow here is the one illustrated by Dikstra's indictment of the GOTO

[14] We're ignoring policy inhibitions here.

statement. The procedure is to exit from a nested cycle the same way you went in, not with an undisciplined jump to somewhere else.

The judgment comes with knowing where to stop (or rather relax) forcing the interface exposure rules because the cost/benefit has passed a useful level. Software engineering provides us outstanding guidance here: modular decomposition and data hiding is the basic rule; when to stop dividing up subroutines and write the code is the judgment call on when to stop decomposing. Clearly this halt to modular decomposition must occur at a level below where components will be used in cross-program interoperability situations.

Chaining

Chaining of information systems is the third of the complexities to handle. In this case, the interface is properly between the Act function of one cycle and the Sense function of the receiving cycle. In other words, we should see S-D-A/S-D-A. The A/S interface is often special-purpose and not feasibly subject to the Good Network Citizen rules. This situation turns into bad architecture when the chain corrupts into S-D-D-A chains, the central problem we've already noted.

Conclusion

One can recognize good interoperability architecture by the degree to which the system modules match the sense, decide and act functions. This modularization work, the real core of architecture, must be done before the mechanics of documenting the data elements and applications of the standards else those tasks will be irrelevant.

Multiple programs must agree on common modularization boundaries if we intend to use multiple programs to build our large information system. This need for modularization agreement is at the heart of the interoperability problem.

Architectural views

The C4ISR Architecture Framework outlines several ‘views’ of information systems. Most of these views have little or nothing to do with interoperability. Further, they seem to have little to do with life cycle supportability, especially when one realizes that a third of the costs are in IT infrastructure support - all people costs incurred after initial operational capability. The three views offered here target these aspects; suggest replacing several existing views with these.

View 1: System modularity model

This is a view of allocation of functions across system modules.

Primary issue: interoperability. Guage the ability to take data from System A and pass to another hypothetical system. Where are the interfaces between Sense and Decide? Can

we take the Sense data and pipe it into a different Decide module or must we go through the System A's Decide box first?

Secondary issue: life cycle maintainability. How easily can System A be upgraded -- how susceptible is it to cascading effects?

Rationale. The crowning hallmark of good IT architecture is modularity that allows a Sensor, Decision support node or Actor from System A to also feed data to System B. Without the designers of System A knowing about System B a priori - after all, it may not exist yet or it may belong to an ally that isn't an ally yet.

If we view System B as nothing more than System A with a mid-life upgrade, then the life cycle intra-system issues (cascading effects) is the same as the inter-system interoperability issues. With precisely the same premiums on modularization.

View 2: Human modularity model (engineering)

A view of procurement and maintenance engineering allocations of expertise.

What's the minimum amount of knowledge that an engineer working on one part of System A needs to know about the other components? (The less global knowledge required, the better).

Rationale. Typically sensor designers are good, say, radar engineers. Or good MRI designers. Or good satcom designers. But they do not possess intimate global knowledge of the complete information system. As the information system becomes more complex, global knowledge becomes increasingly difficult and expensive to maintain. (Software engineering is a microcosm of this problem - linear increases in code sizes tend to cause exponential increases in difficulty in completing a program).

View 3: Training & Logistics modularity model[15]

A view of operations personnel allocation of expertise.

How much does each operator and decision maker in System A need to know about the entire system in order to function?

Rationale. The military, especially, brings its workforce in at the entry level. Further, a large fraction of that workforce leaves the military (relad) at the end of the first enlistment. Training for support and operation of IT systems needs to be 1) effective - so it functions and 2) efficient - so we can get significant payback from the training investment in the first enlistment.

[15] This could also be titled 'Human modularity model (operations).'