12TH ICCRTS
"Adapting C2 to the 21st Century"

Why is C4I Software Hard to Develop?

Track 1: C2 Concepts, Theory, and Policy

"We choose to go to the Moon in this decade, and do all the other things not because they are easy, but because they are hard." John F. Kennedy, 1962

Dr. Lee Whitt
Northrop Grumman Corporation
9326 Spectrum Center Blvd
San Diego, CA 92123
858-514-9400
lee.whitt@ngc.com

Why is C4I Software Hard to Develop?

Dr. Lee Whitt

Abstract

Service Oriented Architecture (SOA) promises to transform the design, development, and deployment of C4I software, heralding a revolution in advanced and flexible warfighting capabilities…..but it's not going to happen, at least not for the next 10-15 years. Legacy C4I software, such as the Global Command and Control System (GCCS), will continue to prosper and evolve during this period, with the most visible "SOA revolution" consisting of point-to-point web services bolted onto legacy functionality….but don't confuse this progress with the promise of SOA.

SOA technologies have been available for about 5 years now – a time frame that exceeds the threshold of patience for the next version of GCCS – yet viable SOA-based C4I systems remain elusive. So what's the problem? The problem is that the truly hard issues of C4I are not being addressed, most prominently the complex business logic specific to C4I. By way of analogy, why is it hard to write software for stock market investors to select winners and avoid losers?

Developing C4I software is significantly more difficult than stock market software, because the business logic is far more complex. This complexity and the attendant implications are the focus of this white paper.

§1 Introduction

Many years ago, Ada was promoted as the silver-bullet programming language that would tame the C4I software beast – it didn't happen. Then Java came along with its portability, mobility, and flexibility to wrestle C4I software into submission – it didn't happen. Then the browser was offered as the fast path to success, delivering complex C4I capability with the simplicity of point-and-click access – it is still a work in progress, with only a few successes after 12+ years of intense development. Other past contenders for the silver-bullet technology award include C++, Common Object Request Broker Architecture (CORBA), Computer Assisted Software Engineering (CASE) tools, and Integrated Development Environment (IDE).

Most recently, Service Oriented Architecture (SOA) constructs, based on web services, have been anointed as the next technology silver-bullet for C4I. SOA governance is being institutionalized as the critical organizational/social component for success, complemented by the artifact-generation processes resident in Capability Maturity Model® Integration CMMI (), Six Sigma, and DoD Architecture Framework (DODAF). All of these efforts have certainly created value, but they have not addressed the core challenge with the design and development of C4I software, namely the role of "business logic" in C4I. In particular, the C4I requirements process (defined and managed through organizational governance) has chronically failed to provide sufficient specificity for developers to deliver software that meets user expectations.

Today, we are besieged with SOA technology visions and marketing bluster, strongly influencing (even dominating) DoD thought and re-directing DoD resources into the transformation of legacy C4I software to SOA-based net-centric C4I capabilities. Indeed, the popular SOA headlines pervading Information Technology (IT) journals and web sites demand urgent action or face dire consequences:

- SOA – The Holy Grail of IT
- SOA isn't optional – it's imperative
- SOA - Start using it today or risk losing everything tomorrow
- SOA – Ignore it at your own peril
- SOA - The silver bullet to reduce costs, improve agility, and fast-track the delivery of products and services
- SOA – The pixie dust that makes everything marvelous
- SOA is sweeping through organizations, upending the competitive order
- SOA will completely re-define IT, providing infinite flexibility at the speed of light
- SOA is really quite simple and very powerful….and it will quickly fix all that ails your broken IT systems
- SOA is a career choice – don't be unemployed

Fear seems to be a key motivator in the SOA discourse and the effect is corrosive in creating an environment for a balanced approach to defining the core issues in the design and development of C4I systems. Even a discussion on what's easy and what's hard in building C4I systems would be useful; there's no balance in the SOA mantra that proclaims everything is easy if only a few common mistakes are avoided. To help the reader calibrate his thinking relative to this paper, it is suggested the reader take a moment to mentally compile of list of what is relatively easy and what is truly hard in the design & development of C4I systems.

As we clamber to create the next-generation of C4I capabilities, it is not prudent to confer SOA technology on every C4I mission area and hope that a C4I mission solution will emerge. Technology is not a solution – only a solution is a solution – and a solution requires that data be processed, managed, and analyzed, under the tight control of well-defined rules (equivalently, business logic). SOA is not about defining complex C4I business logic and, so far, organizational SOA governance has not addressed the necessary details.

Today, legacy C4I software is being placed in a holding pattern while new SOA-based software initiatives are urgently (and sometimes recklessly) launched with the intent of quickly replacing legacy systems. On the surface, the strategy appears reasonable and straightforward, buttressed by a cavalcade of technology standards from industry groups and product offerings from SOA vendors. Some SOA evangelists (representing prominent SOA vendors) may even assert that web services can be developed in only a few days and entire SOA systems completed in just a few months. Case studies provide ballast by documenting real-world SOA successes, where new capabilities were rapidly

delivered at reduced cost and risk. For C4I, the blaring message is that most (all?) legacy software can be expeditiously replaced, often in less than 18 months, once the defense industry gets serious and sponsors commit the necessary resources.

So what is holding us back? SOA technologies have been around for over 5 years and the DoD commitment to SOA has been manifest and material. Given this and given the putative ease with which SOA-based systems can be created and delivered, it is reasonable to ask:

> *Where (in the battlespace) have SOA-based C4I systems replaced legacy C4I systems?*

The stubborn fact is that there has been no replacement….and replacement is the litmus test - not running legacy systems in parallel, not maintaining legacy systems on the back-end, and not keeping legacy systems for "insurance". The success of SOA prototypes in demonstrations and exercises is no substitute for real-world systems providing real-world capabilities.

So in spite of the great SOA shopping spree, the harsh reality is that legacy C4I software continues to thrive in the operational environment and, as argued below, will continue to be successful for another decade or more. Certainly, technology advances continue to provide a powerful engine for progress and innovation, but the core engine behind today's C4I capabilities is the implementation of rule-sets: Complex, often highly-coupled, technology-agnostic, and context-dependent on the mission. As such, it will be very difficult to capture and replicate these rule-sets in a new system, and particularly difficult in an SOA-based system explicitly designed to mitigate the dependency on mission context, system state, and inter-process coupling.

This paper will focus on the business logic embedded in legacy C4I software and make the case that this logic is the key to successful C4I. It is the expression, implementation, and synchronization of this logic, developed over many years (sometimes 10 years or more), that is responsible for the effectiveness of today's C4I systems. Ignoring the critical role of this logic will reduce SOA-based systems to thin shells for basic data transfer and distributed processing, with legacy C4I systems continuing to provide the "heavy lifting" through the processing, management, and analysis of tactical data.

§2 Background

The business logic in C4I applications takes many forms, so it will be helpful in this discussion to provide some examples. Perhaps the most obvious example is correlation logic, specifically the rule-set responsible for analyzing and correlating track data received in various forms (e.g., GOLD, TDIMF, TIBS, M-Series, J-Series, COP messages)[1] from various sources/sensors (e.g., Radar, Sonar, EW, JSTARS, GPS,

[1] Acronym definitions: Tactical Data Intercomputer Message Format (TDIMF) , Tactical Information Broadcast Service (TIBS), M-series are LINK 11, J-series are LINK 16, Common Operational Picture (COP).

TADILs, overhead)[2]. Correlation logic is responsible for comparing the new track information contained in an incoming track report against a database of tracks and, based on various scoring factors (e.g., attribute "weights", source confidence, correlation thresholds, user configurations), make a correlation decision. The decision can take various forms, such as:

- Correlate to existing track
- Create a new track
- Create an ambiguity, along with a list of correlation candidates
- Ignore/discard the report

Furthermore, the correlation logic may spawn secondary correlation effects, such as the automatic merging of two existing tracks into a single track. The simplicity of this discussion belies the remarkable complexity inherent in track correlation logic.

Another example of business logic is the deconfliction of forces, such as the deconfliction of subsurface activities (e.g., submarine movements and surface ship towed array operations) and the deconfliction of airspace activities (e.g., manned & unmanned flight missions, weapon fire zones, controlled ingress/egress routes). Creating and managing these activities in space and in time, while achieving mission goals and accommodating dynamic re-planning in response to unexpected events and conditions leads to a complex rule-set for deconfliction and operational safety.

Yet another example is the business logic embedded in security data guards, designed to inspect data "objects" at various levels (e.g., individual data fields, collection of data fields, metadata) and then take an action to sanitize the data object. The action can be as simple as making no change to the data or as invasive as excising portions of the data and replacing other portions with suitable data substitutes. The diversity of security enclaves and the diversity of data exchange agreements across enclave boundaries, particularly in coalition, multi-lateral, and bi-lateral operations, leads to companion complexity in the rule-sets….and the software implementation of the rule-set must be validated through a comprehensive and rigorous certification process.

In many legacy systems (such as the family of Common Operating Environment (COE) based C4I systems), the embedded complex rule-sets are not amenable to easy decomposition and distribution across the network in the form of web services, for reasons that are inherent to the operational context (not bound to the design of COE-based systems), as will be discussed in this paper. Furthermore, many of these rule-sets are inter-related and operate in concert to support mission planning, execution, and situational awareness across the battlespace. This dependence engenders an additional level of complexity that likewise defies decomposition and distribution in the form of services.

[2] Electronic Warfare (EW), Joint Surveillance and Target Attack Radar System (JSTARS), Global Positioning System (GPS), Tactical Digital Information Link (TADIL)

§3 A Relevant Perspective on the Role of Business Logic

Consider the challenge of writing software designed to optimize an investor's stock portfolio. The high-level system requirements are simply stated:

1. Identify stocks with a high probability to increase in value
2. Compute the maximum expected value and time frame to achieve it
3. Generate "sell" alerts whenever a stock falls below a computed threshold (computed based on a high probability the stock will decrease in value)

For this investment software project, assume that all economic information, market conditions, company performance information, etc., are readily available in real-time and in a well-defined format (e.g., XML)…..and assume further that there are no technology obstacles in terms of software tools, standards, architectures, etc. Under these conditions, will it be hard to create the investment software?

By our assumption, it will be straightforward to collect streams of information on markets and economies, and to write software to perform routine manipulations (e.g., extract and compare elements), but it is not at all clear what rule-set is needed to "intelligently" process and analyze the information to accurately predict the behavior of individual stocks. It's not even clear how to *a priori* test the rule-set for consistency and completeness, nor how to verify the software faithfully implements the rule-set. A highly modular design will make it easier to build and modify the rule-set, but the path to success depends on deep domain expertise and disciplined software engineering.

In a similar fashion, it is relatively straightforward to collect streams of information across the battlespace, and SOA technologies will likely improve collection, support routine processing, and facilitate distribution. But C4I systems are responsible for advanced information processing and analysis, supporting complex mission planning and predictive battlespace awareness. C4I domain expertise, combined with detailed mission requirements and disciplined software engineering, is the key to defining and implementing viable mission rule-sets.

From a systems engineering perspective, C4I planning/execution and investment planning/execution have many similarities in the collection and processing of information, and subsequent analysis for decision-making. Hence, at a design level, SOA approaches for C4I software can be re-cast into SOA approaches for investment software, with the exception that C4I software is (arguably) more difficult to design and develop – not because of technology, but because of the underlying business logic. The motive of an investor is well-defined and linear, characterized mostly by greed with elements of risk mitigation. The motives of today's enemies are complex and highly non-linear (perhaps even chaotic), defying simple characterization. In this respect, the business logic for C4I software will necessarily be more complex than investment software.

Figure 1 is a recent Office of Naval Research (ONR) slide on a candidate information integration framework for C4I planning and decision-making. Replace the reference to "Warfighter" with "Investor" and the references to "Platforms, Sensors, Weapons, Sources, People" with "Industries, Financial Data, Marketing & New Products, Analysts, Consumers & Economic Activity" (respectively) and the result is an information integration framework for investment planning and decision-making.

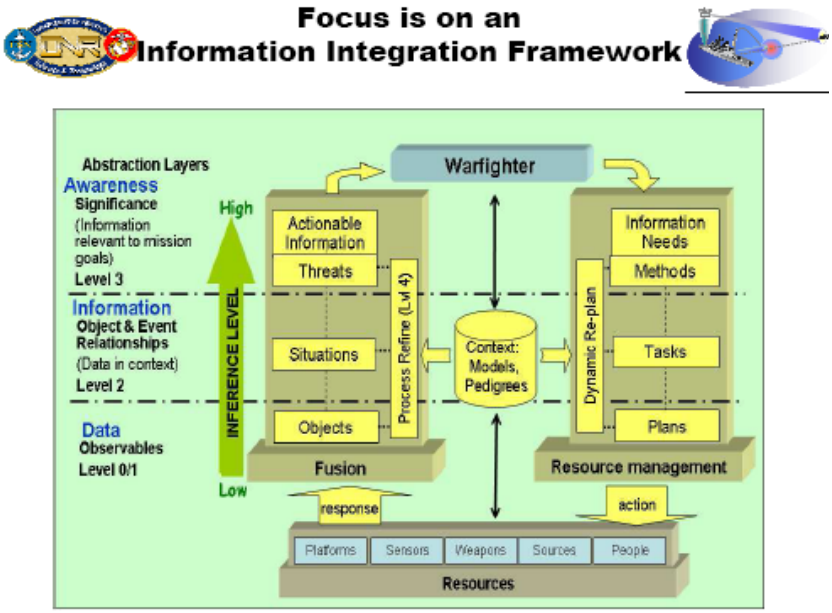

Figure 1: ONR slide showing a candidate Information Integration Framework for C4I planning and decision-making

The business logic of C4I, and specifically the implementation of this logic, is the core engine that is responsible for the effectiveness of C4I systems. Get the logic wrong (represented by the yellow boxes in levels 2 & 3 of Figure 1) and the system will be degraded, perhaps rendering it unusable. It is the central importance and pivotal role of business logic – surpassing any other system characteristic, technology, or interface – that define and differentiate software applications. Of course, business logic must be consistent with policy directives and organizational goals so that, for example, investment logic can be tailored for different types of investment strategies (value investing, growth investing, income investing, etc.).

In summary, business logic is the genome of effective software. This perspective, along with the prodigious complexity in crafting, implementing, and deploying C4I business logic, is the central theme of this paper and the case for why C4I systems are hard to build…..and by extension, the case for the longevity of legacy C4I systems. The following sections provide amplifying details.

§4 Edge-Cases, Uncertainty, and Ambiguity

Whenever business logic is designed, edge-cases, uncertainty, and ambiguity must be accommodated and managed. Data elements often carry inherent uncertainties and data

sets may contain internal conflicts that lead to unexpected and ambiguous situations. Managing these situations is non-trivial and can have a profound effect on the viability and reliability of the business logic.

In C4I and financial investment, edge cases and ambiguous conditions often present the predominant challenges when trying to craft a complete and consistent rule-set….and highlight the conundrum that one person's edge-case event is another person's mainstream event. For example, how do we judge a negative report by one investment analyst on a particular stock – a failure to see the bigger picture or an insight into a trend? In C4I, does an enemy maneuver represent a deception or a pattern of activity? In general, there is no well-defined methodology for defining business logic to handle the diversity of conditions that occur at the boundary. Domain experts can disagree on the nature of a condition (boundary or mainstream), what it means, and how to respond, as evident to anyone who has watched financial experts disagree over stock picks or military experts disagree over tactics.

So what is a reasonable approach to designing C4I (or investment) business logic that can accommodate a broad range of known and unknown conditions, replete with uncertainty and ambiguity? Perhaps the optimal approach is to mitigate decision errors, specifically to balance two types of statistical errors:

- Type 1 Error (omission error – failing to do something that is correct)
- Type 2 Error (commission error – doing something that is incorrect)

The balance is an uncomfortable one. Type 2 errors lead to bad system decisions that occur at computer processing speeds, yielding a high volume of problems. Type 1 errors require user intervention to complete a process or task that the system failed to finish, yielding an overload of manual tasks queued for user action. In both cases, these errors can quickly degrade system utility and usability, erode user confidence, and lead to delays in decision-making and/or poor decisions.

§5 Context, Scalability, and Certification

All software processing occurs within a context. Sometimes the context is shallow requiring no knowledge or management of the current system state, but more meaningful processing requires a context. The example of track correlation is instructive, since an incoming track report cannot be processed in the absence of a track database, as the current state of this database is the context within which the correlation logic operates (cf., §2). Similarly, deconfliction logic requires a comprehensive battlespace context of unit activities (equivalently, a database of unit positions, movements, mission assignments, and mission dynamics) in order to deconflict new mission activities.

A track correlation service, devoid of an underlying track database upon which to operate, would be ineffective at best. In an SOA environment, a user (or system) request to the correlation service to process and correlate a new track report would need access to the user's (or system's) current track database in order to effectively compute. Context is

everything or, to paraphrase Vince Lombardi's credo, "Context is the only thing". Legacy systems do many things well, but they excel at managing system "state" and processing data "in context".....because tightly-coupled systems are well-suited for state management and context monitoring. SOA designs cannot achieve success in this area because the tenets of loosely-coupled design and dynamic discovery are anathema to preserving system state and operational context.

It is easy to expose data on the network through web services, but the underlying business logic (and associated context) responsible for the creation and fusion of data is not exposed and, hence, cannot be accessed by client applications. As a consequence, these clients have freedom of action in terms of actions and manipulation of the data, including data display, secondary fusion, data routing & distribution, etc. As a concrete example, if an F/A-18 flight route is created by mission planning software that faithfully models F/A-18 Hornet flight characteristics (e.g., turn rate, climb rate, maximum flight time), then it is a simple matter to create the flight route as an XML document defined by the Common Route Definition (CRD) schema and post it to a pub/sub repository for discovery and retrieval by applications. However, once an application pulls an XML-formatted flight route, there is no guarantee that the application will respect – or even know - the supporting business logic in the original mission planning application responsible for creating the route. If the route is modified by a (de-coupled and stateless) client application, then the resultant may be an "illegal" F/A-18 route (illegal according to the original rule-set) which could be re-submitted back into the pub/sub repository for other applications to discover and retrieve. In extreme cases, the route change might have the F/A-18 stopping in mid-air, flying into an obstacle, or remaining airborne beyond its fuel capacity.

The decoupling of data from applications that "own" them – without pedigree or reference back to the application and without constraint on allowable operations on the data – will lead to a significant degradation in data reliability and loss of confidence by users. Returning to one of the main motivations for object oriented programming (perhaps even the key driver for object oriented designs), the object construct bundled data with functionality, so that applications pulling data "objects" would also implicitly pull the underlying functionality responsible for managing the data. A worthy goal to be sure, but it had limited success in practice for many reasons (beyond the scope of this paper), but suffice it to say that much of the object's embedded functionality was designed for accessing specific data elements and for data presentation, not the responsible business logic. Today, the replacement of data "objects" by XML documents as the lingua franca for data sharing translates into the replacement of data functionality (devoid of business logic) by data semantics (devoid of state and context)……a sideways step in the IT landscape, in this author's opinion.

Scalability issues are closely related to context, because the more extensive the context, the more relevant the scalability to guarantee acceptable system performance. In the case of COE-based systems, recent advances in track correlation have expanded the track database to "unlimited", meaning that there are no constraints on the size of the track database or its composition (i.e., the number of tracks of different "types", such as LINK,

ELINT, Acoustic, and Missile). Given the requirement to manage global track databases, exceeding several hundred thousand tracks and with update rates on the order of several hundred per second, it is imperative that correlation be operationally efficient and highly reliable. In this setting, it is unlikely that a context-free, distributed, loosely-coupled web service will offer much value in satisfying the performance and state management requirements for correlation over a broad range of data feeds, from (soft) realtime combat systems to non-realtime sources.

C4I legacy software has been generally developed to support a pre-defined client/server environment, meaning that the number of clients connected to servers is typically constrained (e.g., perhaps several hundred clients in a large installation). Indeed, efficiencies have been employed to optimize performance in this client/server environment, such as the use of the UDP and multi-cast protocols. In contrast, web services must be able to support thousands of clients, and efficiencies will be more difficult to achieve, particularly for web services that are bolted onto legacy software, as scalability will be constrained by both the web service and the backend legacy system. Since legacy business logic remains firmly entrenched in legacy systems, these scalability issues cannot be resolved without significant architectural changes.

Software certification is the test and evaluation of software to verify that it will perform as designed within the target operational environment (i.e., context) and subject to the expected data inputs and outputs (i.e., scalability). Legacy systems (such as the GCCS family of systems) generally have a well-defined certification process because the operational environment and input/output channels are well-defined and constrained. SOA designs, which permit dynamic discovery of services, ad hoc composeability of services into functional workflows, and unbounded data sources and data rates (e.g., via pub/sub repositories) are no longer subject to traditional constraints……and indeed may be unbounded. The difference between a good idea and a bad idea is that a good idea has bounds.

In this respect, as noted previously, it is not feasible to simply bolt web services onto legacy systems and expect the resultant SOA systems to be well-behaved. Furthermore, certification of legacy systems does not seamlessly yield certification of the associated "attached" web services nor does it confer certification onto ad hoc capabilities comprised of dynamically discovered web services.

The next-generation SOA enterprise will likely be a thin shell through which legacy systems continue to perform the "heavy lifting", based on their embedded, complex – and operationally validated - business logic. As such, the next-generation SOA enterprise will fall short of expectation and the return-on-investment will be disappointing.

§6 Interoperability

Interoperability is the holy grail of C4I systems. Unfortunately, interoperability is rarely defined with precision (assuming a precise definition is even possible), though we seem

to share an instinctive understanding of what it means – we'll know when we don't have it.

Much progress has been made over the last 10-15 years to field interoperable C4I systems. Perhaps the premier exemplar of interoperability is the COE-based family of systems, which has achieved an unparalleled record of success in delivering compatible C4I capabilities to US and coalition forces since 1995. These systems, and the underlying business logic, have been refined through years of operational use and direct interaction with the warfighter.

However, the emergence of SOA constructs for easily deployed and discoverable services breaks this traditional system engineering approach while creating new challenges (and perhaps set-backs) in furthering C4I interoperability. The problem is not with SOA technology standards and the attendant technical interoperability (defined by published specifications). Rather, it is the ease with which new business logic can be hosted in a net-centric environment and exposed as a discoverable web service….discoverable in terms of the interface requirements, but without discovery (or guaranteed interoperability) of the underlying business logic.

The tight controls imposed on the development of legacy C4I systems (e.g., COE-based systems), and the long test cycles - many will reasonably argue too long - prior to deployment, have been largely responsible for ensuring system effectiveness and deep interoperability across the battlespace. The paradigm shift to web services as rapidly-deployed independent components (independent of a larger C4I context), subject to compliance with SOA standards and approval to operate (e.g., connect to the network), exposes many new entry points that, if unchecked, will erode C4I interoperability. Over time, the proliferation of web services will result in many similar services, similar in terms of advertised functionality (discovered via UDDI and WSDL), but governed by different embedded rule-sets…..and "SOA Governance" is not a magic elixir, in spite of the rhetoric.

An example may help clarify this matter. In military exercises, some of the exercise data may be synthetic, intended to simulate real-world units and mission activities, while other data is real-world (e.g., real-world units reporting their real-world positions). In COE-based systems, there is a clear distinction between synthetic data and real-world data (identified by a "flag" in each track structure), preventing synthetic reports from updating real-world positions and vice versa. Furthermore, synthetic tracks cannot be merged/combined with real-world tracks to form a single track. Without arguing the rationale for this business rule, all COE-based systems provide a consistent (interoperable) foundation for managing and distributing exercise and real-world data across the global network.

In the future, it is conceivable that circumstances will arise when a community of interest (COI) will decide that it is acceptable to blur the distinction and separation between exercise and real-world data, perhaps allowing selected tracks to be merged. Certainly, a web service that performs the merger could be easily developed and quickly deployed,

based on an agreement (within a COI) as to its intended use. But once the web service is registered on the network, it can be discovered by a larger community of users (and processes), perhaps leading to usage beyond the original agreement - the unintended consequence.

This example represents a simple case, but the fundamental concern is the emergence of inconsistent and incompatible business logic, built into discoverable web services, that control the processing and analysis of tactical information for decision-makers. We already see this situation with the deployment of web services, provided by financial institutions, to help customers monitor and manage their investment. Different rule-sets, operating on the same market information, often lead to different investment recommendations, even for the same investment strategy. Furthermore, efforts to compute an "average" recommendation by aggregating and smoothing the recommendations from different financial institutions may not yield a viable investment strategy (e.g., it is not reasonable to "average" a buy and sell recommendation for the same stock).

To be clear, there is value in developing different sets of business logic, based on policy, goals, or other factors. In the last century, mathematicians actively studied game theory (cooperative and non-cooperative games) with a focus on developing optimal strategies. It was hoped that these strategies could be successfully applied to economic models (as rule-sets that capture the role of dominant economic factors), in the same way that calculus was successfully applied to physical models (e.g., computing planetary orbits based on the law of gravitation)….and of course, different models will yield different results.

In the context of C4I, it is a provocative question to ask whether interoperability trumps the rule-set (or model). Restated, the question is whether it is better to have a shared understanding of the battlespace based on an incorrect rule-set or to have different (and divergent) understandings of the battlespace based on better, but different, rule-sets. The core premise of the COP – and the reason for it being called "common" – is that interoperability trumps the rule-set. The reasoning is that it is better to improve a single rule-set than to deconflict and converge different/divergent rule-sets (and iterate this process for every new rule-set that emerges).

For SOA, the success of interoperability at a technical level (e.g., standards compliance) will be off-set by the failure of interoperability at a system level, as conflicting business logic is implemented in web services and hosted on the network for discovery and use. Governance may slow the proliferation of conflicting web services, but it will be generally ineffective at forcing convergence to a common rule-set or limiting the use of conflicting web services. The resulting SOA environment will jeopardize the decade-long trend toward improved cross-service and coalition C4I interoperability, which began in 1995 with the release of the DII COE.

C4I interoperability is a deep concept, penetrating far below the technologies that permeate the veneer of system interfaces, data formats, and network protocols. C4I

interoperability at a global level requires a shared, consistent methodology for data processing, management, and analysis across all systems…..and this means shared business logic across the network.

§7 Coalition Operations and Security Boundaries

One of the most difficult problems in C4I is seamlessly bridging security enclaves. Senior US military commanders have repeatedly asserted that all future military engagements will involve significant coalition forces, so we must be able to seamlessly share information, mission plans, and C2 orders across coalition boundaries. The US Navy's Numbered Fleet Commands (C2F, C3F, C5F, C7F)[3] annually collect and prioritize their top 10 operational C4 requirements to drive the acquisition process. For FY07, coalition and multinational C4I interoperability remained the top priority.

Today, the C4I data exchange environment is relatively simple, consisting mostly of strongly formatted messages, such as GOLD and TADIL messages, and various generic data sets, such as email, images, and database records, often with associated metadata tags. Data guards operate on the data payloads to sanitize the contents, according to an embedded rule-set that complies with a security policy specific to each boundary between two enclaves. Although data guards are improving, the development and test process remains very long, with little allowance to rapidly respond to a dynamic coalition battlespace.

Given the current state of data guards, the introduction of web services and the supporting infrastructure (e.g., UDDI, WSDL, BPEL, WS*)[4] presents an overwhelming raft of new challenges. The ad hoc composeability of web services in realtime to create mission capabilities and workflow, coupled with the flexible transport protocols, SOAP handshake protocols, and formatting requirements of XML documents cannot be *a priori* defined. Even the seemingly simple case of sanitizing the contents of an XML document require that the data guard have access to the XML schema and the dictionary for every field to ensure that a legal XML document is produced.

The diversity of enclaves and the lack of controlling doctrine/policy to completely and consistently define the rules for processing XML documents, accessing (or replicating) UDDI, WSDL, PBEL, etc. across security boundaries presents challenges that will defy our best efforts through the next decade. This is not a failure of organizations or people, but rather a consequence of the complexity inherent in defining rule-sets, while ensuring sufficient flexibility to support SOA deployments, complying with organizational doctrine/policy, and achieving certification through rigorous and comprehensive testing of the implementation.

---

[3] Acronym definitions: Commander Second Fleet (C2F), Commander Third Fleet (C3F), Commander Fifth Fleet (C5F), Commander Seventh Fleet (C7F)

[4] Acronym definitions: Universal Description, Discovery, and Integration (UDDI), Web Service Definition Language (WSDL), Business Process Execution Language (BPEL), Web Service Security, Policy, Reliable Messaging, Addressing, etc. (WS*), Simple Object Access Protocol (SOAP), eXtensible Markup Language (XML)

A cogent argument can be made that it will not be possible to build and deploy SOA-tempered data guards, given the dynamic nature of SOA and the expected spate of new/evolving SOA standards, web services, and XML schema. Instead, the state-of-the-art will be represented by minimalist data guards in specialized environments, defined by support for a limited number of static web services and XML schema.

§8 Why is it hard to develop C4I capabilities

In this section, the issues discussion above will be threaded into a perspective on the challenges in developing C4I capabilities, along with recommendations on how to address these challenges.

There are no easy solutions to the hard problems in C4I and most of the easy problems have already been solved and implemented in software (meaning that we understand the business logic for doing most of the relatively simple things in C4I, such as message processing, attribute-based track correlation, simple data visualization on a map, basic data distribution on a network, and attribute-level semantic interoperability). Indeed, the evolution of C4I systems over the last 2 decades (the period of this author's involvement) has been astonishingly lackluster, faring much worse than the evolution of computer technology. During this period, advances in software technology have not yielded comparative advances in the C4I applications that leverage this technology. By contrast, advances in hardware technology have been stellar, sometimes besting the initial promise and yielding computer products that exceed expectations.

The slow progress in C4I software is not a technology shortfall but, as argued above, a reflection of our inability to specify the business logic for effectively processing and analyzing all of the battlespace information needed to address complex C4I requirements. The same argument applies to the slow progress in financial investment software, and the following "challenge" items apply equally to both C4I and investment domains.

- **Challenge - Business Logic**: Today, we cannot articulate C4I requirements with sufficient granularity to create comprehensive rule-sets for information processing and analysis that accommodates the broad range of battlespace information. This failure is manifest is the generalities found in current C4I requirements, leading to inconsistent interpretations and incompatible implementation. If there are any "grand challenges" in C4I, a detailed specification of requirements would be on the list.

  Recommendation: Governance oversight must be expanded in several areas of the requirements definition process:

  1. Most obvious, requirements should be expanded to provide more details. Recognizing the impossibility of doing this completely and consistently, even modest progress would be significant. Of course, the requirements

process should generally avoid the common pitfall of specifying implementation details, but the SOA juggernaut makes this difficult.

2. System engineers (the ones responsible for translating the requirements into rule-sets and managing the implementation) should be involved in the requirements definition process. Generally, C4I domain experts craft the requirements, but they are not system engineers…in the same way that an airline pilot is a domain expert in all aspects of flying commercial aircraft, but he is not an aeronautical engineer.

3. Governance should mandate the inclusion of (proposed) test plans to clarify requirements and to assist with system verification….as part of the requirement specification. If the statement of a requirement is too generic for a system engineer (or a system tester) to define a material test plan, then the implementation will suffer from unwanted/unexpected behavior. If the buyer can't define the criteria for acceptance in precise and unambiguous terms, then caveat emptor.

4. Governance should recognize the importance, prevalence, and diversity of edge cases and anomalous conditions, and developers should not implement type 1 or type 2 errors in code without clear guidance, including test cases.

- **Challenge - Preserving Context and State**: SOA designs are context-free and stateless, allowing any web service to be dynamically swapped with an equivalent web service located anywhere on the network. As discussed previously, this design is not always optimal or desirable.

  Furthermore, business logic is often context dependent, so specific rule-sets (or subsets) become active when specific conditions are met…..and segregating business logic from context and state may be problematic. As an example of the coupling between rule-sets and context/state, consider the challenge of managing bandwidth, specifically optimizing the use of available bandwidth. Bandwidth management tools generally rely on IP addresses and port assignments, along with the quality of service flags in the packet header, for routing and flow decisions. Each packet is managed independently of other packets, relying only on the information in the packet header.

  With the emergence of IPv6, better bandwidth management tools will be available, but none of these tools have a deep contextual understanding of the data payload, such as the relevant mission areas, the business logic responsible for generating the information, and the relationship and priority vis-à-vis other packets on the network. Is a data stream on threat missile tracks a higher priority than a data stream on threat submarine activity? If the bandwidth is shared with missile defense systems, then the answer is easy. If the bandwidth is dedicated to a ship engaged in anti-submarine warfare and if the threat submarine is in the

vicinity of the ship, then the answer is easy. Without access to context, bandwidth management tools will make suboptimal decisions. Furthermore, if some of the data is already available to the end systems (e.g., through organic sensors), then the data streams can be down-sampled, providing only the portions needed by the end system…..but this requires information on the state of each end system. The optimization of bandwidth requires a complex combination of business logic, context management, and state management. Technology alone will not suffice.

Recommendation: C4I system design must allow flexibility in technology decisions; SOA is not the best approach to every C4I requirement, particularly those where context and state are indispensable. In fact, many C4I requirements have context and state dependencies, leading to the following recommendations:

1. Governance should mandate that requirements include a description of the applicable operational context or system state. If there is no inherent context or state, then a statement to that effect should be included with the requirement.

2. In the definition of test plans (as described in the previous recommendation), the plan should accommodate context and state, if necessary.

3. To the extent possible, business logic should be defined and segregated according to context and state (recognizing that much logic may be shared across context and state boundaries).

- **Challenge - Interoperability**: The perennial challenge is to ensure interoperability across C4I systems. Interoperability with technical standards is relatively easy to achieve, but interoperability of rule-sets has been extraordinarily difficult to achieve, except when the rule-sets are shared through the use of a common implementation (as in the case of the COE). From a governance perspective, the core issue is whether consistency trumps flexibility. If the former, then interoperability will be advanced; if the latter, then interoperability will be elusive.

  Recommendation: For complex software modules, it is generally better to fix problems and re-factor the software to support new technology, rather than starting from scratch (with due recognition that there are exceptions). This approach preserves the embedded business logic and avoids lengthy (and costly) test and fix cycles. Of course, the re-factoring process should address scalability for network environments and efforts should be made to mitigate any new certification requirements….easier said than done!

Even if we could solve many of these hard problems in the evolution of next-generation C4I systems, an even deeper problem looms. In science, we look for patterns of behavior

that can then be abstracted into principles (e.g., laws of physics, mathematical equations). This process allows us to capture the lessons learned of scientific research in a well-defined form. In software development, lessons learned are often captured in design patterns, derived from routine and repeatable software development tasks. The abstraction of these patterns into general constructs allows them to be used – and reused – in many settings. Examples of design patterns include the Model-View-Controller (MVC) pattern for Graphical User Interfaces (GUIs) and the Create-Retrieve-Update-Delete (CRUD) pattern for database interaction. The same approach for capturing patterns applies to CMMI and Six Sigma, where governance and quality control patterns are abstracted and broadly applied to the management of software development. The DODAF portfolio of Operational Views (OV), System Views (SV), and Technical Views (TV) provides various design patterns for architectural views of a C4I system.

Unfortunately, we have not yet been able to identify general design patterns for C4I mission rule-sets….and this in spite of decades of work and volumes of lessons learned. Perhaps we haven't framed the issues correctly, or looked hard enough, or perhaps this area does not lend itself to patterns and abstraction. If the latter case, then we can expect C4I business logic to continue being hand-crafted by teams of domain experts and system engineers working long hours over many years. In this case, C4I systems will continue to be hard – very hard – to develop and advances in technology will be little help.

§9 Concluding Remarks

C4I program managers, stake holders, and sponsors spend too much time on technology, architecture, and user interfaces, while ceding the core software tasks (e.g., responsible for data processing, aggregation, and analysis) to coders who often have minimal operational experience, minimal domain expertise, and often minimal supervision when coding C4I business logic. Anyone can write code to implement functionality. Anyone can write interfaces to capture data. But very few have the domain expertise (operational and engineering) to create comprehensive rule-sets and even fewer to effectively implement complex rule-sets into a C4I system.

This is not a technology issue and seeking technology-focused solutions will squander resources and fail to deliver on the promise. The introduction of SOA will not change these dynamics, and improved governance will have little impact unless it drills deeply into the engineering tank and provides guidance at a level of detail that is meaningful to coders. Indeed, the development of C4I business logic is a staggeringly complex and time-consuming endeavor under the best of circumstances, such as found within the well-defined, pre-defined, and constrained environment of today's legacy systems.

In developing C4I capabilities within an SOA environment, we seem fixated on the mechanics of creating atomic functions (i.e., web services), managing them through governance, and gluing them together via an Enterprise Service Bus (ESB) and workflow constructs. The real value in C4I capabilities springs from the business logic that fuels the evolution of data-to-information and information-to-understanding, and then

empowers decision-makers to assert effective command and control across the battlespace.

Lack of attention to the preeminent role of C4I business logic will be largely responsible for the continued dependency on legacy systems over the next 10-15 years…..and the business logic embedded in these legacy systems will tenaciously resist efforts to extract, decompose, and reconstitute the rule-sets into an interoperable SOA environment (interoperable with legacy systems at the deep level of business logic). During this period, promises to transform C4I through an infusion of new SOA technology will be costly, problematic, and delayed, though strong governance will help mitigate the easy problems.

Fear must be attenuated as a key motivation for rushing into SOA, and high-frequency marketing bluster like *"Are you agile or fragile? Are you leading or failing? Are you growing or fading?"* do not advance the discourse. The rigors of responsibility must temper the stampede to deploy SOA. Yet, in spite of the lack of real-world success in deploying SOA-based C4I systems over the last 5 years, we are harried into immediate action to embrace – are you ready for this - the next wave of SOA technology (called SOA 2.0, or Web 2.0 or maybe Enterprise 2.0), affording even more benefits and efficiencies…..please make it stop!

So how do we avoid the excesses of SOA thinking and moderate the hype surrounding SOA technologies? It is a profound mistake to view SOA as a solution, especially while ignoring the hard problems in C4I discussed in this paper. For every contrarian view of SOA, there are hundreds of briefs, white papers, webcasts, conferences, case studies, etc. extolling the virtues & benefits of SOA. SOA success in areas such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), payroll, and supply chain are possible because organizations have a reasonably firm understanding of the necessary business logic, with sufficient granularity for implementation. This is not the case with C4I as can be seen by reading any Request for Proposal (RFP) for a C4I system. We can avoid some of the sensationalism of SOA by focusing more attention on what the software is suppose to do rather than the mechanics of how it will be done.

Of course, progress in SOA will continue to be made, but near-term expectations for advanced SOA-based C4I capabilities will remain unfulfilled. Instead, legacy C4I systems will continue to be the load-bearing foundation for C4I in real-world operations, with point-to-point web services bolted on to evince the success of SOA.

Dr. Lee Whitt is currently a Northrop Grumman Technical Fellow involved in the design and development of C4I systems. He is one of the original developers of the first desktop-based C4ISR prototype called JOTS (Joint Operational Tactical System), initially deployed in 1985 on Carrier Battle Groups and Ashore Command Centers in all theaters of operation. Throughout the 90s, he focused on the Pacific Theater, supporting ashore and afloat commands with the evolution of C4I systems from JOTS to JMCIS to

GCCS. He led the development of the first web-based C4I system (ELVIS = Enhanced Linked Virtual Information System) in 1995 and the first Java-based C4I system (ELVIS II) in 1996. More recently, his interest is on the challenges of developing network-centric capabilities.

He earned his Ph.D. in mathematics from Yale University (1975) and worked in academia for several years as an assistant professor on the faculty of Cornell University and Texas A&M University.