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

Why is C4I Software Hard to Develop?

Track 1: C2 Concepts, Theory, and Policy

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

Abstract

SOA technologies promise 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 (e.g., the GCCS family of systems) will continue to prosper and evolve during this period, with the most visible "SOA" evolution 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 you can't even find a viable SOA-based C4I prototype. So what's the problem? The problem is that the truly hard problems 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 that selects winners and avoids losers?

Developing C4I software is significantly more difficult than stock market software, because the business logic is far more complex. This complexity is the starting point for this white paper…..so get use to the presence and prevalence of legacy C4I software.

§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, but it has so far failed to deliver on the promise. Other past contenders for the silver-bullet technology award include C++, CORBA, CASE (Computer Assisted Software Engineering) tools, and IDE (Integrated Development Environment).

Most recently, Service Oriented Architecture (SOA) constructs, based on web services, have been anointed as the next technology silver-bullet for C4I. SOA governance has also ascended, complemented by the obligatory management and artifact-generation processes that are now institutionalized as CMMI (Capability Maturity Model® Integration), Six Sigma, and DODAF (DoD Architecture Framework). 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 business logic of C4I.

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 demand urgent action or face dire consequences:

- 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 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

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 key 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 to balance the SOA mantra that everything is easy as long as we avoid a few common mistakes (e.g., not enough governance, too many web services, not enough web services, lack of standards compliance, lack of infrastructure such as an Enterprise Service Bus). To help the reader calibrate his thinking relative to this paper, I suggest the reader take a moment and mentally compile of list of what is easy in the design & development of C4I systems and, much more importantly, what's hard.

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 embedded rules (equivalently, business logic). SOA is not about C4I business logic.

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. I've heard SOA evangelists (representing prominent SOA vendors) assert that, with their products, 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 quickly replaced, often in less than 18 months, once 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 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?*

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, I will argue 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 have some examples in mind. 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, TAB37, TIBS, M-Series, J-Series, COP messages) from various sources/sensors (e.g., Radar, Sonar, EW, JSTARS, GPS, TADILs, overhead). Correlation logic is responsible for comparing the track information contained in an incoming track report against a database of tracks and, based on various factors (e.g., attribute "weights", source confidence, correlation thresholds, user configurations), the correlation software makes a decision on the track report vis-à-vis the track database, and takes an action 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 software 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 flight missions, UAV missions, weapon fire zones, controlled ingress/egress routes). Creating and managing safe-havens for 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.

Yet another example is the business logic embedded in 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. Furthermore, the software implementation of the rule-set must be validated through a long certification process that has its own measure of complexity. Of course, the previous examples also require software development and certification to verify proper implementation of the rule-sets, but data guards have a unique place in C4I due to their role protecting security enclaves.

In many legacy systems (such as the family of 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 will be discussed later 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 higher level of complexity that is inherent in the today's legacy C4I systems and, again, not amenable to decomposition and distribution in the form of services.

§3 A Relevant Perspective – The Role of Business Logic

Consider the challenge of writing software designed to optimize an investor's stock portfolio. The 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 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 the technology for software design & development and the hardware for performance & scalability are not obstacles. Under theses conditions, will it be hard to create the software?

By assumption, it is straightforward to collect streams of information on markets and economies, and to write software to perform simple manipulations (e.g., extract and compare elements), but it is not at all clear what business logic is needed to "intelligently" process and analyze the information to select winning stocks. Furthermore, the rule-set must be tested for consistency and completeness, edge cases and anomalous conditions must be handled, and the software implementation must be verified to ensure proper execution of the rule-set. A good modular design will make it easier to build the rule-set and to isolate changes (when needed), but domain expertise in disciplined harmony with software engineering is the key to defining and implementing the rule-set.

In a similar fashion, it is relatively straightforward to collect streams of information across the battlespace, and SOA technologies will likely improve collection and facilitate distribution, but C4I systems are still responsible for information processing and analysis and C4I domain expertise, combined with disciplined software engineering, is the key to implementing mission rule-sets.

From a systems engineering perspective, C4I planning/execution and investment planning/execution have many similarities, particularly in the collection and processing of information, and subsequent analysis and presentation 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) much 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 simply characterization. In this respect, the business logic for C4I software will necessarily be more complex than investment software.

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 (in C4I and investment software) and effectiveness will suffer, perhaps rendering the system unusable. It is the central importance and pivotal role of business logic – surpassing any other system characteristic or attribute – that define and differentiate software applications. Business logic is the genome of effective software.

This perspective, along with the extraordinary complexity in crafting and implementing C4I logic, is the theme of this paper and the case for why C4I systems are hard to build. 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.

This represents another similarity between C4I and investment software, and even highlights 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 larger 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 near domain boundaries. Even experts can disagree on the nature of the condition (boundary or mainstream), what it means, and how to respond, as evident to anyone who has watched experienced financial analysts 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? 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, since the automation of business logic in software means that type 2 errors become bad system decisions occurring at CPU speeds, resulting in a high volume of processing errors. Type 1 errors require user intervention to complete a process or task that the system failed to finish, possibly leading to an overload of manual tasks queued for user resolution. In the latter case, an audit trail or "resolution" log (along with prioritization) may be necessary to avoid overwhelming the user. In both cases, these errors can quickly degrade system utility and usability, leading to delays in decision-making and/or incorrect 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.

In addition, the delivery of data in an SOA environment (including publish & subscribe repositories) will suffer from the lack of context. While it is easy to expose data on the network through web services, the business logic responsible for the creation and fusion of data is not exposed, hence not available to client applications. This affords freedom of action to these clients in terms of data manipulation, data display, data routing & distribution, etc. As a concrete example, if an F/A-18 route is created by mission planning software that faithfully models F/A-18 flight characteristics (e.g., turn rate, climb rate, maximum flight time), then it is a simple matter to post the route (e.g., as an XML document defined by the schema CRD – Common Route Definition) into a pub/sub repository for applications to find and pull. However, once an application receives the XML flight route, there is no guarantee that the application will respect – or even know - the business logic inherent to the original mission planning application. If the route is modified, then it is possible the changes will result in an F/A-18 route that is "illegal", according to the original rule-set (e.g., the modification might have the F/A-18 stop in mid-air, fly into an obstacle, or stay 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. Reviewing one of the main motivations for objected oriented programming, the object construct bundled data with functionality, so that applications pulling data "objects" would also implicitly pull the functionality to manage 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 important the scalability to ensure 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, AIS, TBM). 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, loosely-coupled web service will offer much value in satisfying the performance and state management requirements for correlation.

C4I legacy software has been generally developed to support the 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). Efficiencies have been employed to optimize performance in this network 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, since 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 capabilities, 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 business logic. As such, the ROI of the next-generation SOA enterprise 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 it when we see it. Much progress has been made over the last 10-15 years to field interoperable C4I systems, particularly across the COE-based family of systems which has achieved an unparalleled record of success in delivering interoperable C4I capabilities to US and coalition forces. These systems, and their shared embedded business logic, have evolved through many years of operational use and through an extensive test and evaluation process to verify operational effectiveness across the battlespace.

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 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 points of entry 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 internal 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 (e.g., an unintended consequences).

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 lead to difference advice and different investment strategies, even given access to the same information and the same initial conditions. Furthermore, efforts to define an "average" strategy by aggregating and smoothing results derived from various web services, accessed across different financial institution, will not yield a viable investment strategy (e.g., it is not reasonable to "average" a buy and sell recommendation for the same stock).

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 (e.g., conflicting business logic), reversing a decade-long trend toward improving C4I interoperability. Technology is not the main impediment to achieving C4I interoperability (recognizing the important role of technical standards). C4I interoperability is a deep concept, penetrating far below the technologies that permeate system interfaces, data formats, and network protocols. C4I interoperability at a global level requires a consistent and common methodology for data processing, management, and analysis across all systems…..and this means shared business logic employed across the network by C4I systems to provide C4I mission capabilities.

------------------ end of "smooth" draft -----------------------
------------------ beginning of "rough" draft --------------------

Discussion topics are bulletized under each section below.

<u>§7 Coalition Operations and Security Boundaries</u>

- The US Navy's Numbered Fleet Commands (C2F, C3F, C5F, C7F) annually collect and prioritize their top 10 operational C4 requirements to drive the acquisition process. In previous years, coalition and multinational C4I interoperability has been the number one priority, and for FY07, it remains the number one priority.

- There are significant challenges in supporting coalition operations and, specifically crossing security boundaries – it is hard enough with today's legacy systems, but supporting SOA protocols and standards (e.g., SOAP, UDDI, WSDL, WS*) through data guards will resist our best efforts.

- A data guard is the implementation of a rules engine, applying embedded business logic to a data payload to remove/modify specific contents in order to sanitize data as it passes from a higher security level to a lower security level.

- MLS, MSL, MISL, and other similar constructs solve the easy problem of crossing security boundaries, namely allowing user access to information at and below the user's security level. The associated business logic is straightforward, consisting of a directed graph with nodes corresponding to security levels and a rule that walks the graph, starting at the user's security level, to identify the lower security levels available to the user.

<u>§8 Why is it hard to develop C4I capabilities</u>

- "For every complex problem, there is a solution that is simple, neat and wrong"….H. L. Mencken

- For every problem there is a solution which is simple, obvious, and wrong"-Albert Einstein
- The easy path is often the quickest route to the wrong place

- Summarize above points to succinctly organize why business logic is hard to develop (and certify), highlighting its dependency on state & context, the need for commonality, consistency, and scalability, and the pernicious problems of boundary effects and anomalous conditions.

- Discuss bandwidth constraints and, specifically, the lack of business logic that optimizes the use of available bandwidth. Many bandwidth management tools are simplistic, using IP and port assignments to provide quality of service and, perhaps, looking somewhat deeper into the TCP/IP packets. The emergence of IPv6 will improve some of this management, but none of these tools have a deep understanding of the data payload, its mission context, and its relationship and priority vis-à-vis other packets on the network. Network management tools can easily intercept every packet and inspect the contents, but these tools have only a few simple rules (based on a small set of fields in each packet) for making decisions. The optimization of bandwidth is not a technology issue, so IPv6 offers no solution. This is a very hard problem in defining and implementing business logic, providing flexibility to change rule-sets as conditions change.

- Even if we could address many of these (very hard) problems in the evolution of next-generation C4I systems, there is an even deeper problem, perhaps the basis for a "Grand Challenge" in C4I. 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 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 OVs, SVs, and TVs provides various design patterns for architectural views of a C4I system. Unfortunately – and here's the Grand Challenge - we have not been able to find general design patterns for C4I mission logic….and this in spite of decades of work and volumes of lessons learned. Perhaps we haven't 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 domain experts working long hours over many years.…..so it will continue to be hard to develop C4I systems (and new advances in technology will be little help).

- In developing C4I capabilities with an SOA environment, we are too focused on the mechanics of creating atomic functions (i.e., web services) and gluing them together via an ESB and simple workflow constructs. The real value will be derived from the business logic that orchestrates data & functionality, in concert with user interaction and environmental drivers (e.g., bandwidth, fault recovery, IA, access controls). Google is today's information powerhouse precisely because of the business logic that it has developed and implemented in its search engine…..and no one doubts that this was very hard to do. Advances in technology certainly helped Google achieve success and Google is now exposing some of its capabilities as services, but it was their innovations in defining the rules for information analysis that lifted Google to dominance.

- Today, C4I projects, project 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, management, and analysis) to coders who often have minimal operational experience, minimal domain expertise, minimal, and (perhaps worst of all) minimal supervision. The introduction of SOA will further exacerbate this problem, and improved governance will have little effect because it doesn't go down deep enough into the engineering tank and it isn't at a level of detail that is meaningful to coders. The development of C4I business logic is a complex and time-consuming endeavor under the best of circumstances, even within a well-defined, pre-defined, and constrained environment.

<u>§9 Concluding Remarks</u>

- Lack of attention on the hard problems of C4I (business logic) in the develop of C4I will have the unintended consequence of keeping legacy systems around for at least another 10 years (along with the support costs). The migration to SOA will also result in bring other unintended consequences, some of which have been discussed in this paper.

- Be wary of who offers to transform C4I through an infusion of new technology; be wary of anyone that that offers an easy path forward; be wary of anyone that asserts success is guaranteed if certain basic practices are followed (e.g., governance).

- Fear must be removed as the key motivator for rushing into SOA. And fear-mongering bumper stickers don't help the discussion: *Are you agile or fragile? Are you leading or failing? Are you growing or fading?* The rigors of responsibility must temper the rush to deploy SOA. And, in spite of the lack of real-world progress or success in deploying SOA-based systems over the last 5 years, we are now harried into immediate action because the next wave of SOA technology (called SOA 2.0, or Web 2.0 or maybe Enterprise 2.0) is upon us, providing even more benefits and efficiencies…..make it stop!

- Rules provide for management of data and for management of functionality. The separation of data and functionality (the antithesis of object-oriented thinking) bodes poorly for SOA. To be clear, nearly all object-oriented implementations were not fully object-oriented, but the client/server model maintained a close & tight connection between rules, data, and functionality. Anyone can write code to implement functionality. Anyone can write interfaces to capture data. Very few have the domain expertise (operational and engineering) to create rules and even fewer to integrate diverse 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.

- 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. Why isn't there a conference on "SOA failures...a case study in disaster"? Or, why isn't there a conference on "Software bugs....SOA problems that cripple". The answer is obvious: there is no market for negativity in SOA, particularly now that it is firmly anchored in the collective consciousness of DoD.