SERVICE ORIENTED AQUISITION

Evolving how Services Acquire Software in a Net-Centric Age

Version 0.9c <u>DRAFT</u>, 09/17/2008

Authors

Michael Behrens, R2AD, LLC (Editor) , Eugene Luster, CSC

Abstract

Many software paradigms have come and gone. More are on the way. All purport to be the silver bullet that has eluded the software development industry since its inception around the middle part of the last century. One thing experience has taught us is that unless the acquisition process keeps up with the dynamic nature of software development it could become a stumbling block that prevents best practices from taking hold.

This paper covers a myriad of topics that the authors have noted which affect the entire software life-cycle process with a focus on acquisition. With each issue raised, recommendations for improvement are provided which hopefully can be implemented so that the U.S. Taxpayer can purchase the best DoD software systems for the U.S. Military and its allies.

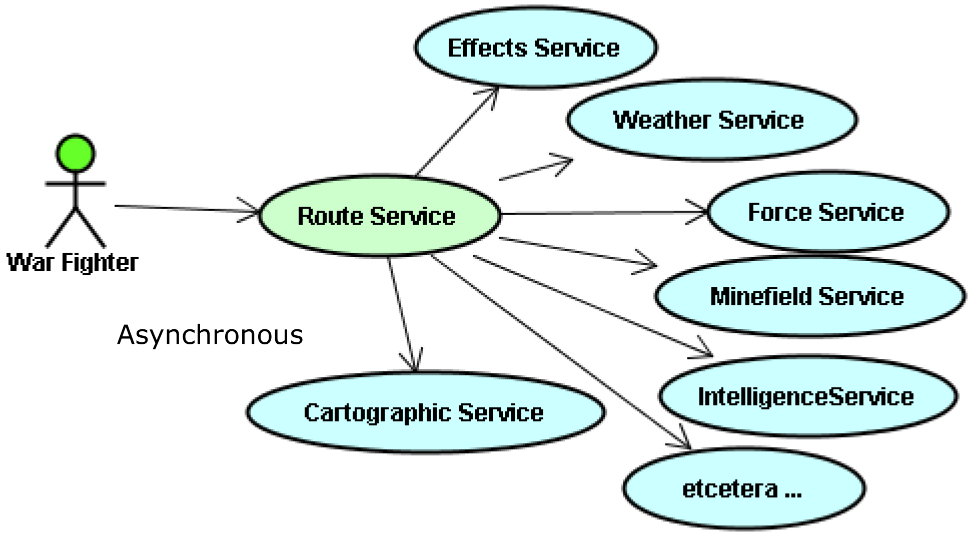
A goal of this paper is to describe how acquisition of enterprise software systems can promote interoperable interfaces, it is necessary to combine Net-Centric or Service Oriented Architecture (SOA) concepts with software development processes. The papers takes a holistic view and considers impacts of acquisition theory to the entire software life-cycle.

Table Of Contents

I. Service Oriented Architecture

Thinking Service Oriented Architecture[1]

If a minefield were available as overlay, then the graphics and perhaps a name would be displayable as part of the battlespace visualization in a C2 system. The operators that created the overlay would know what it meant and would include it on their briefs along with verbal or textual information about the situation.



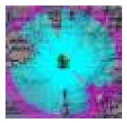


Figure 1 - Cascading Services

With our net-centric hat on, that overlay could become available as an overlay web service and made available to a wider community and even available using simple HTTP requests as part of mashups[2]. However the meaning and importance of that shaded area would be lost outside the context in which it was created if only the graphical representation was disseminated. A logical question is raised: should there be a service for every type of overlay, in this case, a minefield web service? It could provide the graphics using Geographic Markup Language (GML) as well as providing useful data about the minefield – the data behind the overlay (type, timing model, status, country of origin, pedigree, etc).

Taking this a bit further, higher order services would then use this service as they perform their functions such as route planning. A soldier in the field could ask the higher-order geo-spatial route service to generate a route between two points for given parameters. This route service could then invoke other sub-services for roads, minefields, blockades, checkpoints, enemy positions, sensors, weather, etc. and provide an appropriate route back to the soldier.

In general, everyone (architects, designers, PM) should think about how "data" can be used by others outside the enclave. The services should be self-describing, self-contained, and modular. Data should be capable of being externally referenced with some lifecycle guarantees (meaning, the service should be backwards compatible). All these in turn can then be choreographed to make the network the computer.

[1] Extracted from the Thinking SOA paper v1.2 available at http://www.r2ad.com/papers.
[2] http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

An Example High Order Service

Consider an end-user on a wireless thin client requesting an evacuation route from a route service. Many back-end services are invoked automatically to ensure that the route provided is secure and fast. There is a hierarchy of services at work being invoked by the route service using the weighting factors provided by the end-user (e.g: security is more important than speed). Some of these might include weather and the location of enemy and friendly forces. These lower level services are also being invoked by other services independently in a stateless manner.

Many assumptions are made in the diagram, however it is important to understand a few of them. Consider the minefield service for instance. It would need to expose a operation which returns a list of minefields within a geographic box or along a multi-part path or corridor. Presumably, along with that information is an ability to obtain geo-rectified vector graphics for those minefields, in the case where the user is able to request visualization in a coordinate space.

Now suppose a new service is created which provides information about biological or chemical hazards. Can the system ingest that new data source and convey it as part of a command and control presentation service and be incorporated into the route service? Can this be done "without" any code changes to the existing route or presentation services?

If we can achieve that sort of dynamic data integration, then we will have true information integration. For the engineers, this means that the design must be robust and agile, taking into account extensibility and versioning concerns.

This example scenario is useful from many perspectives. It helps one to look at net-centricity in a more abstract manner. One can "design" many such hierarchical dependent services in depth, in order to reveal many patterns which should be applied to all service implementations (naming, security, auditing, priority of invocation, pedigree/providence, updates, etc). From the warfighter point of view, the importance is the timeliness and accuracy of the route. These are part of the requirements which need to be specified in the command and control requirements document.

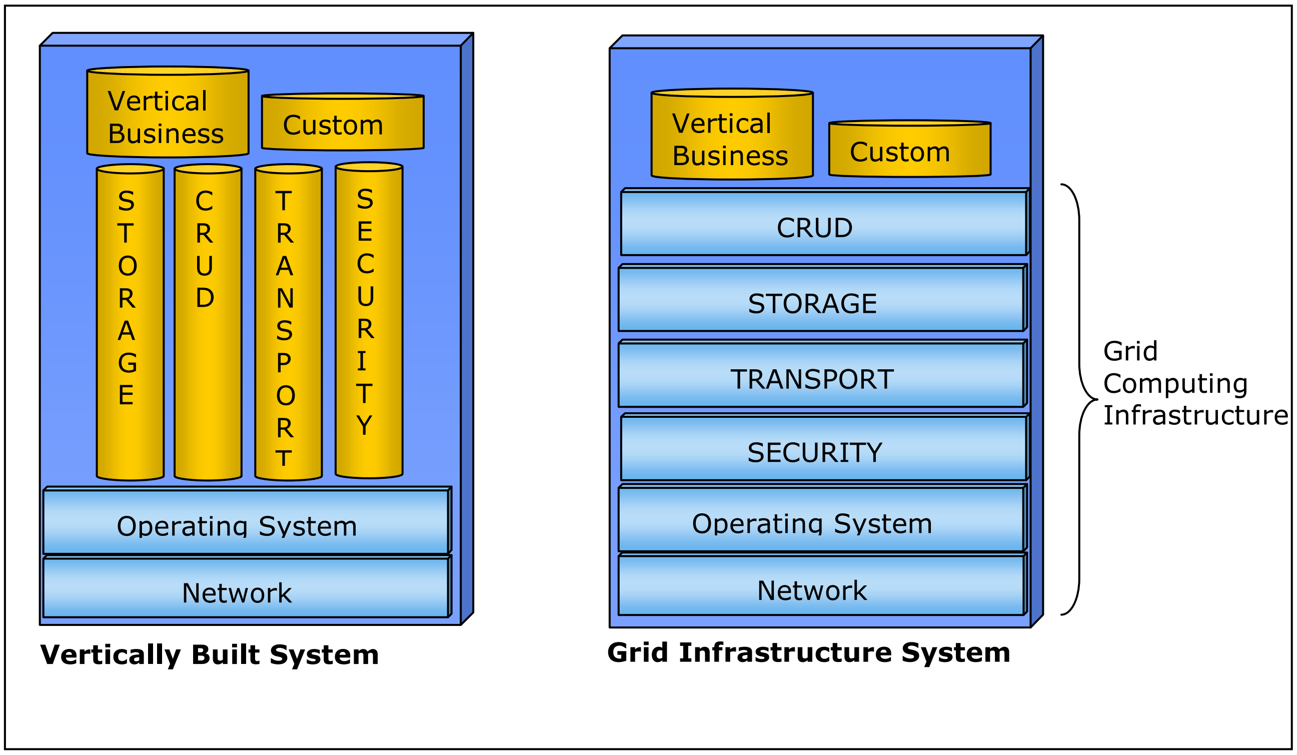
Common Services

The general consensus is that stovepipes are bad. One of the forcing functions which create stovepipes is acquisition policy that do not encourage common infrastructure. Contracts and task orders that focus solely on a single capability or function end up creating a stovepipe. This raises the overall cost of the system because many dollars are spent on the integration and interoperability down the road, simply because using common services were not employed.

Even within the execution of a contract, management tends to break things apart into different divisions or groups or based on the wrong categrory, thereby creating a stovepipe potential. An example of this would be creating a group based on the branch of Service (Army, Air Force, etc). Instead, groupings would be better if organized functionally, such as Security, Data, Transport, etc.

Acquisition must think horizontally. This means that contracts need to be let out which take advantage of and provide for horizontal layers with less reliance on vertical layers. The more functionality that can be derived from the horizontals in the architecture, the more cost effective the system is and the more re-use is attained. The project then can focus on its main business (warfighter) purpose.

Consider two separate areas: Intelligence and Medical. Currently, both systems would be under different contracts and different vehicles. What should happen is a total system engineering approach that acknowledges that all systems share many of the same functions

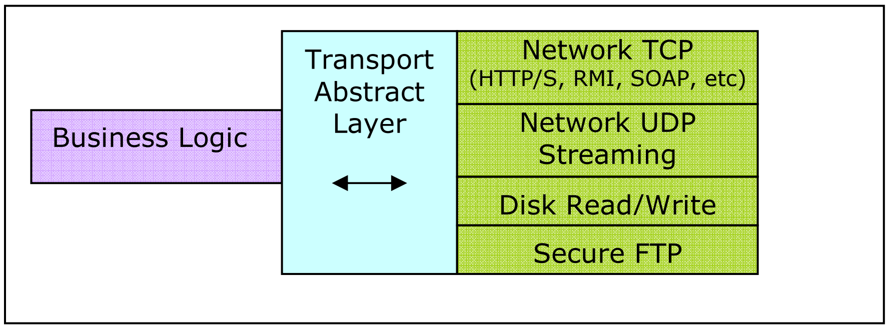


and patterns.
Every system has many things in common:

- store, retrieve, update, and remove data (standard CRUD operations)
- Transport the information to different systems
- display the information in standardized formats (word processing, spreadsheets, presentations, accounting, etc)
- secure the information with transactional integrity and auditing
- provide for access controls
- configuration management of updates and patches
- help and training
- and many more

As another example, examine at how many different ways there are to distribute data between sights: Oracle Replication, Sybase Replication, Proprietary synchronization messages, network appliances, and others.

Having different implementations of these common services is not necessarily bad, however having a standardized interface for them would shield developers and the necessary vertical components from having to develop their own. These standardized layers is what the Grid Computing field is attempting to create and supporting standards organizations such as the Global Grid Forum is one way to achieve horizontal layers which all systems can benefit from.



Common or cross-cutting services therefore help reduce duplication of service implementations and help increase the level of interoperability. Developers and the Acquisition process must help ensure standards and published specifications (DoD or from a standards body) are actually used and not just referenced. The DoD should insert itself strongly into the standards process to help ensure that the products meet the critical needs of the department (such as disconnected or low-bandwidth operations.

If governance does not advocate standardized interfaces for these basic functions, then developers will discover the newer technology and incorporate it in proprietary fashions which cultivate lack of interoperability. Advertising these interfaces is important for adoption.

Service Level Agreement Transport

Along the lines of using shared infrastructure is the use of a standardized transport for our data that is exposed as services or other means. An inherent pitfall in following standards however is that the specification behind the standard is open to interpretation, leading to implementation variants. For example, we suggest that the DoD take up the cause of developing a specification for a reliable, robust, and secure communications service. This would entail much more detail that simply stating that an Enterprise Service Bus (ESB) should be used. This transport service should be engineered from the ground up with the requirements that we know we must have such as guaranteed delivery and distributed data policy. Standards such as Java Messaging Service (JMS), which is a huge part of ESBs today, already offer guaranteed delivery, however what they mean by that is that if the network is down, they simply queue up the message for later delivery. This simple queue methodology causes performance bottlenecks and delivery of stale content. What a robust transport layer would do instead is to understand the network and use it to make the delivery of data to include using other means such as email, secure FTP, or satellite, even sneaker-net if need be to accomplish high priority communications. The transport service would be designed with priority channels, built in integrity such as those used in satellite communications to enable reassembly of data if the stream is interrupted. The creators of information would also indicate the pedigree of the data along with time oriented descriptors which can be used by information dissemination algorithms.

An application developer would have to simply specify a delivery level of service, much like we do when we purchase something and choose between overnight or ground delivery requirements. This would enable application developers to also provide a data policy which can be referentially attached to the data or data stream to help ensure access rights and distribution control via embedded tagging.

II. Service Development

Service Inventory

A service registry is an inventory or catalog of all DoD software listing services under development and in the field. Information such as the program sponsor, development, time period, and other factors would be available to the many authorized individuals. This inventory would be accessible on-line, open, and searchable. It can be distributed in that each Service could stand up their own inventory which can be searched and indexed by others.

An active inventory of DoD software is needed to help promote awareness of services (net-centric software components). A federated registry will promote re-use and to reduce the amount of wasted tax payer dollars on duplicative non-interoperable development.

Currently the Federated Development and Certification Environment (FDCE) process in the Net-Enabled Command Capability (NECC) program is an example of how services can be registered during different test phases and how a common development environment can help ensure interoperability though code-sharing. This aspect was most successful as part of an NCES collaboration environment known as the GiG Enterprise Development Network (GEDEN) based on Collabnet. It has been shutdown, however a new and better site should be available soon called DoDForge.

The key change for Acquisition is to ensure that the contract language supports the sharing of source code and build scripts. Furthermore, it is important that development efforts first look at what has already been developed before re-inventing the wheel. Therefore it is important for the registry to also show programs/services/tools/etc which are under development across the enterprise. Otherwise, when it comes time to use a service, the warfighter will have too many choices which do not interoperate. Even the government team which writes the contracts must first examine and use this registry to ensure they do not become the reason for duplication of effort.

This is the exact reason why governance in a SOA is so important. If the services are not interoperable then the point of the SOA is lost. This is why it is important for architects to be involved in the development cycle and it is important for organizations to create governance boards to create policy and make sure services conform to those policies before being put into operation. While the FDCE concept can help, it would most likely be better for each COI to have their own FDCE and a parent FDCE to govern cross-community services.

The products that are built need to be advertised and promoted in order to increase awareness and incorporation of them. Word of mouth is not very efficient. Creating a slick-sheet or a product web page with features and screen shots, for example, can help prevent the re-invention of the wheel. Search engines, such as Google, should be able to pick up that information. An emailed newsletter and email announcements would also help distribute knowledge about existing and forthcoming capabilities.

Many systems today are so huge, that over time, parts of them are forgotten and end up with duplicate functions implemented in different ways over time. Creating smaller component services can help prevent the cost of "dead" code. GCCS-J, for instance, has far more capability within it than anyone realizes (i.e.: Intel and COP Web Services, interactions with Google Earth, etc).

This paper proposes a service catalog (not UDDI) for Architects, System Engineers, and management/developers to go to in order to find out what all the current services are and also, what the planned services are. This would help reduce the wheel reinvention syndrome.

Therefore this catalog or service white pages and yellow pages would need to provide developer point of contact (POC) information, release schedules, draft and final (if fielded) WSDL files, and perhaps multiple endpoints which can be used for testing and for actual use.

Contracting agencies should be able to reference this registry to ensure that developers are not reinventing the wheel. Developers need to use this registry in their design and proposal documentation. Furthermore, contractors and agencies should be able to advertise their services as well (yellow pages), creating a marketplace with a little competition.

Software Repository

The Ada Information Clearinghouse [AdaIC] is a good example sharing software to promoting re-use. This is similar to SourceForge, Java.net, or the Java Community Process (JPC)[3] on the internet.

Currently, most systems are developed over a period of time, tested and evaluated over a period of time, then fielded and patched over a period of time. This paper proposes a new approach that would provide field offices the ability to perform their own maintenance by the qualified F&M teams. This would move to the "edge" the software change process making it more responsive to the regional needs. At the same time, changes which affect the greater community need to make its way across regions. Good configuration management processes can help ensure communication and acknowledgement of changes.

One benefit of this approach described earlier in this document is that the original developers would become totally devoted to the next generation system. Another benefit is that the F&M teams could work closely with the end user to affect positive change in the system. For example, one site might want a new button added to a GUI to help filter objects from a map or to help with printing. Since all the code would be accessible to the F&M teams, they would be able to add this sort of minor functionality. These modifications would not require massive redesign or retesting efforts. Minor bugs could also be fixed and recurring problems could be analyzed near their source.

There are some security concerns; however they are mitigated by establishing Regional Maintenance Centers (RMC). These centers would collaborate with the other RMCs and with R&D teams. Everyone with access would have access to the repository which would be under strict configuration management. Only those changes which have been tested and verified with the embedded security engineer would be allowed to be executed on operational systems. This model is actually already being used, however not officially and not in a controlled manner. The establishment of RMCs would help bring more control the system and along with it more security.

Once a standard repository is adopted, such as Application Content Services (ACS)[4], then maintenance can be performed on code in the repository in a controlled and secure manner. ACS is a trusted software repository which is a new specification which leverages the OASIS specification called Solution Deployment Descriptor (SDD) which provide a function similar to the COE installer and the DISA DII COE Integrated and Runtime Specification (I&RTS) descriptors.

Source Code

Currently, many contracts do not require that the source code be delivered. This is terrible. As contractors come and go and software systems age, it is important that the systems used by the war fighters be capable of being maintained in an efficient manner. Furthermore,

[3] Java® Community Process homepage: http://jcp.org
[4] ACS Web Site: https://forge.gridforum.org/projects/acs-wg

source code, when delivered, can be used to verify functionality, validate security, and to gather metrics on efficiency and compliance to standards.
At delivery time, the contractors should be asked to not only deliver the code, but also the scripts and tools necessary to create the binaries and the entire delivered media. Delivering functioning virtual machines or zones that contain the full development environment would help as well. Furthermore, the government agency should only put into use the binaries that have been created from that source using the instructions and tools specified by the developers.
This concept is not new and not radical. This concept has been successfully employed on numerous contracts such as the CMTC[5] and JRTC systems from TRADOC via STRICOM.

Every developer that builds software systems has the tools that create the software. If that capability were also at the "edge", then last minute or emergency fixes could be made close to where the capability is deployed. New features could be added in critical or controlled environments. Taking this concept a little further, the creation of Regional Software Maintenance Centers whose responsibilities would include the configuration management and maintenance of software is in order. This would free up the production world to focus on real advancement. Many times engineers have become tied down in maintenance that prevents "re-factoring" from occurring. Collaboration between the centers would ensure that fixes are shared and that new features can be employed if needed in other regions.

The DoD should establish a requirement to analysis all source code. Usually, the U.S. Government (DoD in particular) has only usage rights. Having the source code supports automated testing and analysis, and long term maintainability. There is movement in this area though the use a federated development environment in the NECC arena. The serious DoD contractor concerns are expected to be their right to secure their intellectual property and not to lose their competitive advantage when contracts come up for re-compete. A radically new way to develop systems for the DoD can be the use of software appliances which enhance retains corporate value while at the same time provides the best quality system for the warfighters.

This sort of model was successfully employed in at least one system developed for the U.S. Army at the Combat Maneuver Training Center (CMTC)[6] which has been maintained on site for over 10 years.

Code Re-use

Code re-use is achievable if there is a will to do so. Reuse can save money which can then be spent to build newer and better capabilities which can be delivered sooner. How many word processors are needed in the field today? The answer of course is one. How many ATO viewers or Target systems or message processors are needed? Why do we have so many different variations of software which perform essentially the same function? How many millions of dollars could be saved if there were a single authoritative council that directed the architecture and design of our command and control systems?

A single authority could be created by an act of Congress or internally within the department. This might evolve out of the current JFCOM efforts to unify the Services, however there are too many cooks in the kitchen so to speak. Each service (AF, Navy, etc) has their own budgets for systems and this causes conflicting efforts and waste. Attempts in the past to unify the services have for the most part failed. The failure to agree on a single way of doing the same things is covered up with the notion of "family of systems". Even today's Service Oriented Architecture notions are being twisted at the protocol and semantic level which creates minimal interoperability. Instead of every service and agency being in a race to come

[5] http://www.jmrc.hqjmtc.army.mil/
[6] Entity Force Structure: http://www.r2ad.com/papers/ForceStructure-R2AD.pdf

out with a SOA solution, lets all take the time and work together to first lay out the requirements, then a specification, then a standard from which to implement against. This would yield a set of standardized network interfaces at a minimum which is the goal of NECC. There could then be a marketplace that has multiple instances of a service created by different providers. Communities of interest would be free to determine which the best service is and therefore which get ongoing support. Because these services would all conform to the same interface the developer or end-user all use the services in the same way.

The Java Community Process (JCP) is a good model to follow. Another would be the IETF or the GGF[7]. What if all command and control software were like Java? There would be a single package for creating a URL and making a network connection and establishing a secure context. Changes to the system would be exercised in small focused JSR-like working groups consisting of stakeholders. These groups would be created with the approval of the board. The forward progress would not be duplicative and as wasteful as today's software development.

As a side note, perhaps there is too much money in the DoD for software development. This presumably stems from the way each project is funded: No direction and community policy enforcement to encourage the re-use of specifications, architectures, designs, and code. Without top-down direction, there is no reason for individual projects to use anything that already exists. Why should they, if there is funding for them to re-invent the wheel? They can work and develop and be happy without consideration about other applications and how they interact and are managed. On the other hand, with so much duplication, there exists competition. So perhaps the challenge is to officially hold competitions which focus on interoperability and to reward achievement.

Program Utilization Metric

Another aspect of acquisition that can change to positively affect net-centric software development is the creation and use of a new measurement which shows the effectiveness and adoption of software. This is related to code re-use in that it is a measure of the success of a program to have active applications deployed and used.

Far to often, program management or higher-level agency policy stop funding on a project without knowing the full effect of that decision. For example, consider an Army project for terrain analysis that is being developed by the Army which they also provided to other services and agencies. What would the impact be if Army decided not to fund the delivery of the capability to the Joint Program office just because of a misalignment in technical architecture? The application would soon be pulled and then become unavailable on systems which warfighters use. If a metric were collected which indicated that the terrain analysis tool was actively being used at 58 sites around the world (in addition to the Army specific sites), then the Army task monitor would be more likely to continue funding the Joint aspect of the project because the return on investment is huge. Army, in this example, should be allotted "credit" for providing capability beyond their Area of Responsibility.

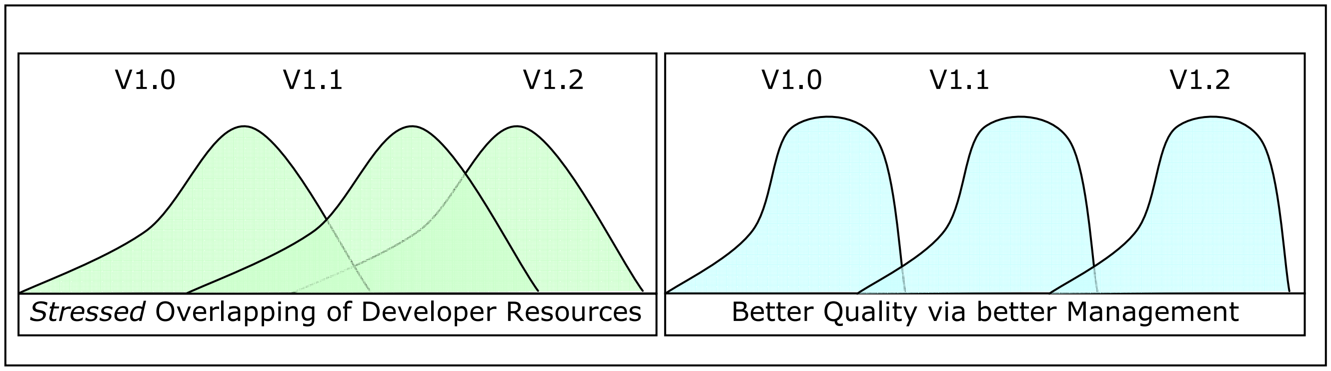
Separation of Concerns

Software developers can generally handle more than one task at a time, however the efficiency factor declines as more tasks are added. The acquisition process must ensure that the time lines that drive the schedules and fielding are taking into account how they affect the quality of the software systems being built. In some systems of record, the author is aware of developers having to be responsive to problems in the field for version 1.0 while delivering

[7] The Internet Engineering Task Force (www.ietf.org) & the Global Grid Forum (www.ggf.org)

and testing 1.1 while working on design materials being presented at PDRs and CDRs for 1.2 while preparing for the research and programmatics of 1.3 or 2.0.

The diagram below depicts in green (on the left) how development teams are constantly responsible for many versions at one time (overlap). The graph on the right separates fielding from development and there is minimal overlap of versions.



The question remains how the fielding concerns can be separated from the development concerns. One method typically employed by commercial software houses is to create two distinct departments: Manufacturing and Production. Manufacturing can be considered a Research and Development (R&D) Integrated Product Team (IPT) and Production can be considered a Fielding and Maintenance (F&M) IPT.

In the diagrams above, the overlap in the first graph and the larger gaps in the second workload graph would be filled by the F&M team. R&D teams are more expensive than F&M teams, so by reducing the workload on the R&D team, the cost savings can be applied to the F&M budget. The cost model of the software develop should look then look more like a bell curve instead of a flat line of constant IT spending. This model was successfully employed as part of STRICOM's training system acquisition.

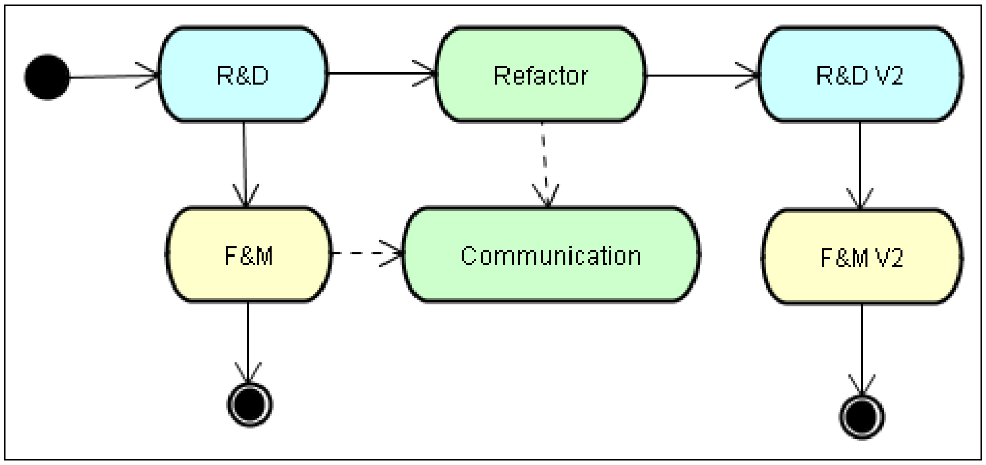


Figure 3 - R&D and F&M interactions

The breaks between versions then provide for a "refactoring" period during which the lessons learned, newer technology and training can be brought to bear for the next release. This period is a time of enlightenment. The developers are brought out of their development labs and can recharge, like coming out of a cave into the light.

It is vital that management ensure that the vital communication between the R&D and F&M is fostered. It is recommended that during each cycle, a small portion of the R&D team actually become part of the F&M team for a short period. The lessons they learn can be brought back into the R&D fold. Those developers that gain good experience with testing and fielding generally build better quality software.

Keep the procurement model distinct from any particular technology choice used by the developing contractors is a way to ensure that the same acquisition process can be used regardless of whether SOA or Client-Server is employed as a means to meet the requirements. Maintaining good communication between all involved is extremely important.

Another factor which could be healthy is periodic budget crunches where management is forced to prioritize funding. An historic example of this occurred during the Gramm-Rudman-Hollings law which forced a ten percent cut across budgets.

Without a solid governance model, SOA will be chaotic resulting in extra dollars being spent later to address shortcomings and worse, money would have been spent on capabilities that never get used.

Cyclic Waterfall Software Development

One benefit of the waterfall software development cycle was that it had a clear beginning and ending. Schedules might slip, as they tend to do, however when the project was over, it was over. The resultant system could then be in use for years with only periodic maintenance and needed enhancements. The quality of the system was greater in the end, mostly because everyone knew the stakes involved at each step of the way. All stakeholders examined the output of each step thoroughly to ensure it was done right the first time. Process oriented projects ensured that there was a cleanly defined mechanism to insert changes, such as new requirements, which spawn a miniature speedy cycle to catch up to the main development with solid impact analysis and reviews. Without that mature process, requirements creep from multiple sources and the real end-user requirements are not fully captured. The budget model typical mirrors a bell curve.

On the other hand, the more recent cyclic refinement methodologies present no reward for success and support failure. If there is a bug or a missing feature, then it is fixed on the next go around. Production software has a tendency to become "good enough". A special project manager and development team is required to have good experience in order to be able to listen to the customer input and respond with agility to prioritize the changes. The budget model tends to be constant over time or worse, increases at a steady rate.

Therefore, what is needed is a combination of the two models, referred to here as the cyclic-waterfall, or the "Seven Pools Method", after the Seven Sacred Pools of Kipahulu on Maui, HI.

It is important to get a working system out the door periodically (the waterfall). While in the pool, cyclic processes can be used to specify-build-test. In-between pools, developers and managers have an opportunity to examine newer technology and other competing implementations. During this breather, complete re-factoring can occur as needed. The older development team can be downsized if needed or put on different projects all together. As the project matures, the pools become deeper meaning they have more capabilities, are more robust, have a higher quality, and are closer to being in production (fielded). The budget model has ups and downs and becomes less over time.

III. System Architecture

Architecture Gap

The majority of today's developers are not aware of the architect's work. They develop their systems and as an after thought, the "architecture" might be updated or created to reflect the current state of the system. This process is backwards from the science of software engineering. This is a trend which was noticed by us in the late 1980's. The gap has grown some since then. As technology advances and developers adapt, the gap space changes, sometimes getting better sometimes worse, based on this author's observations.

In any event, there is a real gap, as evidenced by lack of metrics showing how architecture is actually being used with the DoD[8]. In order to reduce this gap, it is important for architects to be a part of the actual software development lifecycle. While many programs have an official architecture group and/or a least a lead or chief architect, it has become commonplace that they are often away from the development activities and show up at the Preliminary and Critical Design Reviews (PDRs & CDRs) in time to make some minor adjustments. However this practice tends to place the architects on the defensive and at odds with the software designers. If the architect could appoint junior architects in each major development shop, then perhaps the communication lines could remain open through all the phases of development. Furthermore, it is recommended that Architecture Reviews (ARs) be established before the PDR to ensure that the architecture being adopted by the designers is inline with the vision and policies needed to achieve interoperability, shared standards and designs.

While some argue that it is not possible to architect and design the enterprise, we believe that a unified architecture is possible. This unified architecture, much like the architecture stack of the world wide web, would exist for the purpose of server applications and sharing resources such as storage and processing power. This new web would evolve from the grid computing technology and standards allowing application logic and data to be distributed across a dynamic mesh.

Architecture Patterns

Just as there are object-oriented programming patterns, there are also architecture patterns. They become very evident when looking at the data flow diagrams, for instance, of the many command and control systems. They all ingest data, store the data, and provide query mechanisms for processing and display.

We need to begin to re-use architecture in the same way it is desirable to reuse code libraries. This can help reduce cost and increase interoperability. Analyzing the various systems for these patterns would yield enough information to present a single architecture for the DoD. Given more time, the authors of this paper would like to perform this task and publish the results on architecture patterns in the DoD. Efforts such as NCOW, NESI, and others would be analyzed with the cooperation of their government bodies and the DoD Architecture Framework (DoDAF) and the Federal Enterprise Architecture Framework (FEAF).

Albert Einstein desired a unified field theory to join the mechanical laws with the theory of relativity. A common architecture can become a corner stone upon which to build many systems in an efficient manner. The Object Management Group (OMG) introduced Model Drive Architecture (MDA) which supports the abstract of architecture away from a specific language implementation.

[8] GOA review: http://www.gao.gov/new.items/d04731r.pdf

The ongoing logical follow-on would be a unified architecture where constructs such as persistence, cache, transaction, audit, authenticate, etc. would all carry the same detailed semantics making implementations of them compatible between systems.

Software Model Relativity

Model Driven Architecture (MDA) promises that all code might be generated from the "model". While this can be true, the quality and depth of the code generation component comes into question. While the idea of a model is good, let's consider the concept of the model for the military. Where is the C2 or C4ISR Model for the U.S. Military? There have been attempts to rally around models like C2IEDM or others, however they are limiting and different groups interpret them differently, abuse their meaning, or more often extend the model to fit their needs. This of course causes problems when communicating. Can there be a single model for all C2?

Applying Object Oriented methodology, one would say that since in the real world there is only one of each thing and that all attributes and behaviors exist in the real world, it is therefore just a matter of writing them down for our problem space (battlefield, etc). Of course it is not that simple as the perceptions of reality for different individuals play a role on what each person or organization or country might believe the model should be. Essentially, Einstein's Theory of Relativity has a role in software development. The traditional "is a" or "has a" relationship all depends on the point of view – it is relative. Does a tank contain a bullet or is a bullet in a tank. What about time? Einstein would insist that we bring that up because not only does it depend on our point of view but also of time. What are all of the relationships for the round when it is being manufactured, or warehoused, or transported, or loaded, or fired?

These "forces" in software development create a never-ending cycle of change that will cost billions and trillions of dollars over time (black hole?).

It is therefore important to convey/communicate to all developers the semantics of interfaces as well as the syntax. The architectures of the net-centric era need to bring a common language to the systems of records to ensure interoperability happens.

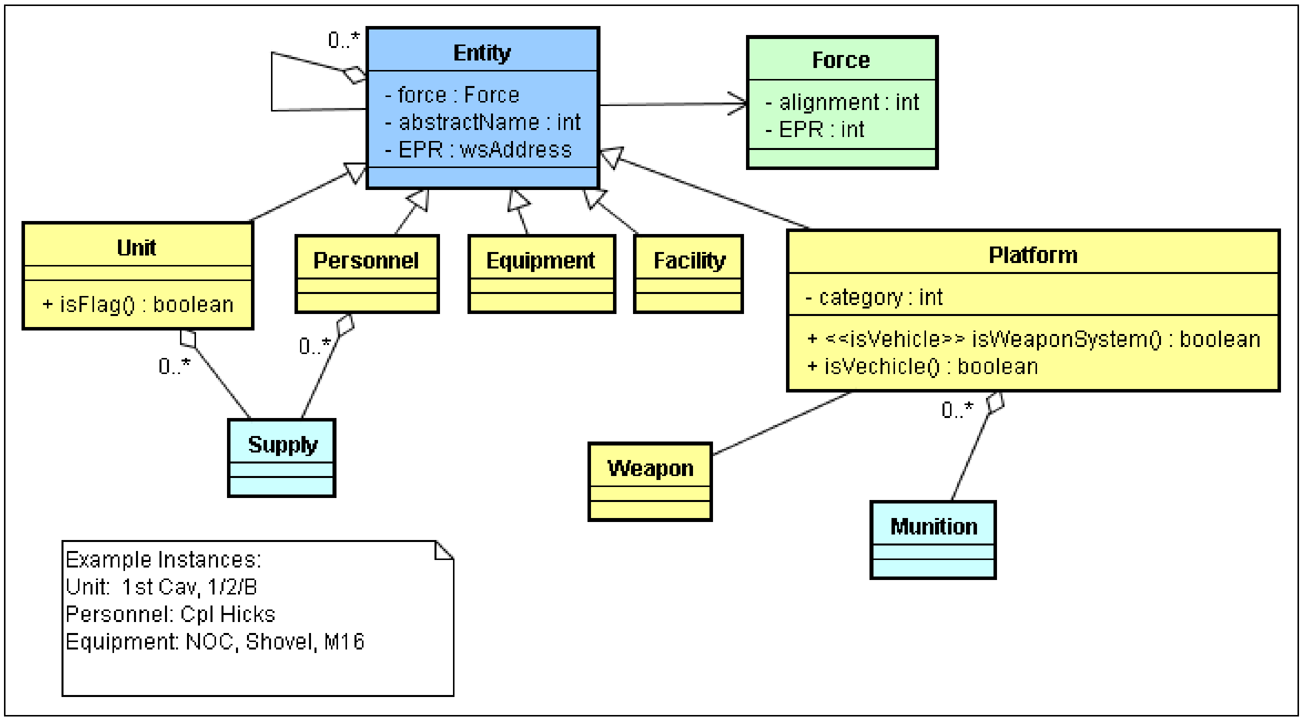


Figure 4 - Entity Structure Pattern

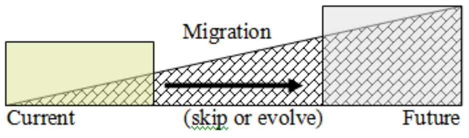
Today's best and evolving standard for conveying and depicting architecture and design is MDA which is built on the shoulders of Unified Modeling Language (UML). Figure 4 shows an example of a well understood model, Task Organization or Order of Battle. It is not complete, however it conveys that a battlefield entity can contain other entities (hierarchy) and that these entities are associated with a force. Even though we generally understand this model, it is difficult to have a consensus on what it should be exactly. This dilemma creates an interoperability challenge. Either there are multiple models which can't be exchanged, or the model becomes too huge and complex that it is a barrier to implementation. Perhaps the answer might be to have models of models which can extend each other.

Net-Centric or Data-Centric?

Data is important, no doubt, however centering on data or any one aspect of net-centricity (services) causes net-silos wherein only those services hard-coded to understand the semantics of the data are able to use it. To overcome that, many "bridges" end up being created manually or using tools (essentially programming translators). Instead, it is important to follow how one can create a service to exchange information without human intervention. This is perhaps wishful thinking and too far off into the future, however at a minimum more emphasis is needed on the standards which enable standardized cross-domain communication and cross-CoI communication.[9]

IV. Migration from Legacy Architectures

Migration Paths



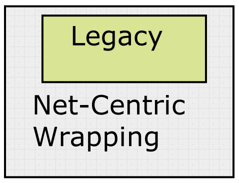
An abstract and detailed view of the system architecture is needed to understand the full impact of decisions which impact the course of the future system of record. There are many possible migration paths which fall into two major categories/ philosophies:

- Complete Overhaul and Re-Design
- Stepwise Cyclic/Spiral <u>Evolution</u>

There are pluses and minuses to each approach, however while the future systems are in the process of defining themselves, it seems that the older systems are evolving and proving themselves to be worthy of retention. For example, GCCS-J is already using Web Services and will be even more so in the near future, essentially catching up to capabilities displayed in the NECC Pilots and in some cases surpassing them.

So it seems that new development should certainly be reviewed for net-centricity (along with security, GUI design, standards, etc) while at the same time legacy systems should strive to expose a net-centric interface where external customers exist. Waivers should be permitted for legacy systems which work just fine (email, for instance), which do not expose fully compliant net-services for others to use. Thick clients will always be needed as well, so plan for them (i.e.: Google Earth).

Wrapping



The concept of creating an abstraction layer on top of more complicated elements is a familiar pattern in software engineering. From a migration

[9] See white paper on a Semantic Exchange Protocol: www.r2ad.com/papers

standpoint, this is a critical tenant on which more effort should be expended to ensure that this pattern is used for external interfaces.

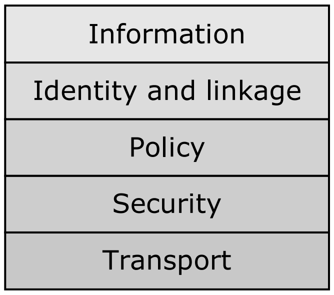
V. Distributed Computing

Power to the Edge

A migration path must take into account the advances made in distributed computing and the specific subfield of grid computing. Basically, grid computing can be defined as the harnessing and sharing of computing power (CPU, memory, and storage) of networked computers to solve problems and perform tasks.

First must be a recognition that net-centric computing is distributed computing and likewise, grid computing is net-centric. It is logical to state this, however many do not fully appreciate the vision and concept of grid computing and therefore dismiss it with the unintentional side-effect of ignoring the advancements and contributions made by these fields by academia, governments, and industry. Just as database technology once evolved from 1st-4th generation technologies, management must be aware that web services are also evolving with grid computing leading the way.

Key Technologies



Regardless of how standards evolve, it is certain that several key technologies are required in order for a complete victory over stove pipe non-interoperable systems:

- Universal identity (WS-Naming perhaps, Handle.net, WSRT)
- Enterprise Management (WSDM, WS-Management, WBEM, WSRF, SML, etc.)
- Policy (WS-Policy, WS-Agreement, perhaps)
- Search (Google, perhaps)
- Geospatial (GML, KML)
- Notification (Alerting, CAP from OASIS)
- Authorization (Policy agreements)
- Configuration (WS-Management or WSDL or WSRF)
- Data Distribution (caching)
- Provisioning (secure application distribution, ACS perhaps)
- End User and Machine trusted Identity (PKI, WS-Naming)
- Basic Execution Environments (BES)
- Advanced grid containers such as Globus

Secondary to these are other various technologies which help make things easier (transformation engines, accelerators, languages, etc). These others naturally occur and become available without much effort (they are easy). The main ones however are harder to establish and require leadership direction, and authority to ensure interoperability.

Data to the Edge

The user might go in and out of communication so a network stateful caching session should be considered early as part of the infrastructure.

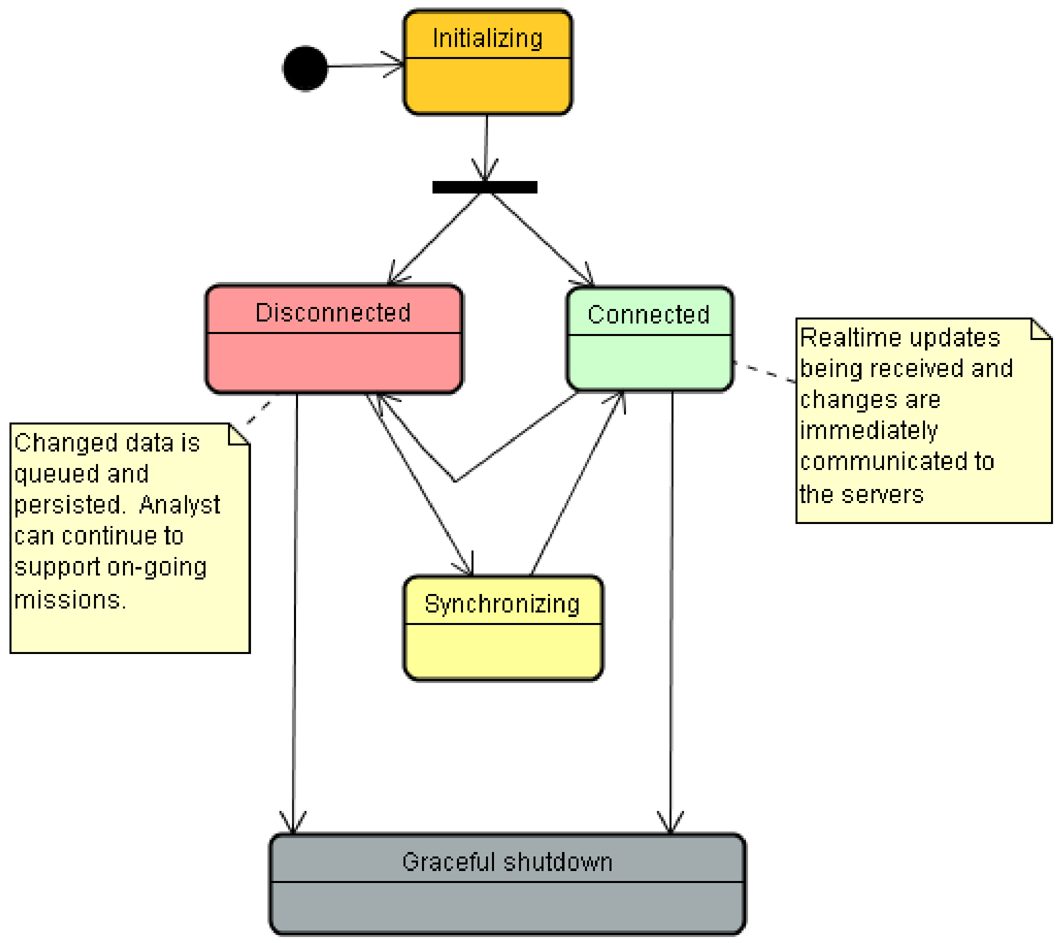
There are different meanings of the term "disconnected ops". I've described two definitions below. There are probably others and variations….
Perhaps arriving at a consensus or different some different terms might help ensure a positive software/system design. In either case, it is important to maintain transactional integrity of the information.

Definition One: When a client's LAN is disconnected (either intentionally or unintentionally) the application software automatically enters an off-line state. The user is visually made aware of the disconnected state. Changes made to any existing data or additions of data are queued and persisted locally. When the connection is established, the changes are communicated to the server and the client receives current state information. Any entered or changed items made while off-line are checked for coherency and any conflicts are communicated to the user for resolution.

Definition Two: The user synchronizes the client and then intentionally unplugs from the network or directs the software to enter into a stand-alone mode. Data can be queued in a similar manner as stated in definition one. The client, upon reconnection with the LAN, directs the software to perform a re-synchronization. Conflicts are again communicated to the user for resolution.

All services and capabilities should be designed with the limitations of the network in mind. If these considerations are not part of the design, then a huge burden is put onto the latter stages of system development where it is the most expensive to effect change.

Incorporating a distributed data design into a common service, perhaps as part of transport or storage, would provide benefit to all participants.


Initializing
Disconnected
Connected
Realtime updates being received and changes are immediately communicated to the servers
Changed data is queued and persisted. Analyst can continue to support on-going missions.
Synchronizing
Graceful shutdown

Principles of Net-Centricity

John Garstka describes the principles of net centricity in this way:

> *The tenets of Net Centricity are as follows: end-to-end communications that supports connectivity, interoperability, security, and discovery. For IETF purposes across the Internet. For First Responders or 911 support within a Metropolitan network with access to some form of command control center, and for Defense these principles should work and apply across their Global Information Grid (not Grid per OGF or the SOA from Father of that Grid Dr. Ian Foster, but rather a network Grid).*

See: http://www.dtic.mil/futurejointwarfare/concepts/netcentric_jfc.pdf

VI. Odds and Ends

Standards Compliant

Beware of the term "Standards Based".
Compliance to a published and adopted specification recommended by a standard body results in interoperability. Just being "Standards Based" accomplished very little in terms of realized interoperability. A system could use C++ and claim that it is standards based.

Compliant software can also adhere to published profile(s) and thus enable agencies to deploy solutions that interoperate even when based on different open source and/or commercial software vendors' implementations. Two examples of profiles are the WS-Interoperability (WS-I) [WS-I BP 1.1] and the OGSA WSRF Basic Profile.

Patriotism & Rewards

DoD software developers want to know that they are doing their part to help the country. They are patriotic. Government program managers should bear this in mind as they issue work orders and prepare program schedules. Software developers gain great pride in knowing that their work is fielded and is making a difference for the warfighter. Feedback is important. Just the simple act of a congratulatory letter from the PM to the project staff is a great motivator. A letter from the field is even better (especially from high-ranking officers). The opposite is also true. Lack of fielding and lack of acknowledgement is sure to reduce morale and performance.

Accountability & Requirements

Contractors that are late or deliver code that has errors in it should not be paid (at least as much as they would otherwise receive). The government must be able to hold money back from the contractors that do not deliver a product that meets the requirements. If the delivered software, even though on time, has bugs which impact capability and performance, then the contractor should be required to fix the problems on their dime. The government should not be in the business of paying for code which does not work or is not in compliance with the agreed upon criteria.

In order to enforce this, the contracting agency must have an excellent understanding of the specifications and must be able to understand what it is that the developer is producing. A clear understanding of the detailed requirements is required in order to enforce the quality of the product. It is therefore important that the Government be provided a good specification, and has allowed for sufficient time and resources for integration and testing. Generic and vague tasking should raise the red flag.

The U.S. Government should not reward contractors that deliver buggy software with more money to fix the bugs they wrote. Independent Verification and Validation (IV&V) is an important tool that the government needs to employ. IV&V teams need to be driven by a good set of requirements which can be used to validate the system.

However, it is important to understand that requirements can change. Every software engineer is well aware of the pitfall called "requirements creep". The answer to this dilemma is two fold:

- Develop Requirements that are complete as possible
- Ensure the development process can handle new requirements

Spending a decent percentage of the product life-cycle on requirements analysis is healthy. It prevents "requirements-creep". Likewise, spending a decent amount of time and energy on design can help create a robust product which can withstand new requirements. An example of this is the use of abstractions in design. Instead of designing many code modules around a specific database schema, a generic approach can be taken which isolates code from items which are likely to change in time, such as schema. While binding eventually is needed, applications can use current technology to mitigate the cost of changes by using tools such as Hibernate in the DB world and using XPath and XQuery in the XML world. Mediation technology and virtualization technology (grids) offer a cost effective solution.

Certified DoD Software Engineer

Software developers for the DoD today generally have a security clearance which they must renew periodically and receive interim training updates. However they do not receive any certification on how to develop DoD software. Microsoft, Oracle, Sun, and many other companies offer certification training and testing.

The DoD should do the same to ensure that developers are aware of the security guidelines that they MUST follow. Furthermore, such a certification process could help make developers aware of the software registries, architecture documents, and other policies of the DoD. This can be a tool to help reduce cost of development and to make engineers aware of existing policies and processes.

Certification and Accreditation of Services

Most systems today have a "type" accreditation. Any deviations from the set of software components (including the OS in many instances) require the approval and authorization of the local decision authority. This poses a problem when considering the net-centric world where a service is stood up without complete knowledge of all the interactions possible in the production environment.

We recommend that a policy be developed between the project management, Certifier and DAA that says each distinct service or capability be separately certified and accredited, with end to end testing to the maximum extent possible of the superset of services intercommunicating as a consistent part of their functions.

Discrete units of capability (OS, web server, app, etc) are tested individually, with the conglomerate findings used to develop a risk assessment. Taking this a step further, separate testing could be conducted on the risk assessment of a set of orchestrated services.

Another important facet which pertains to this is that in order to meet Service Level Agreements (SLAs), services need to be capable of being deployed dynamically into various "service containers". Grid computing offers this technology today. This sort of environment should be approved by a knowledgeable security team. The end result will be a robust and secure command and control environment which brings "power to the edge"[10].

Service PitFalls

The pitfalls known so far should be documented. The lessons learned from the pilots could be documented in a way to help legacy systems move ahead. Industry should be consulted, being wary of the sales language and motivations of companies.

One such pitfall is defining the network services at the wrong level of detail or with the wrong focus which can lead to many nearly identical services (specialized) which do not interoperate to share real information.

Another might be that networked services become overly dependent on single points of failure so that, like a rack of dominoes, when ones goes down, they all go down.

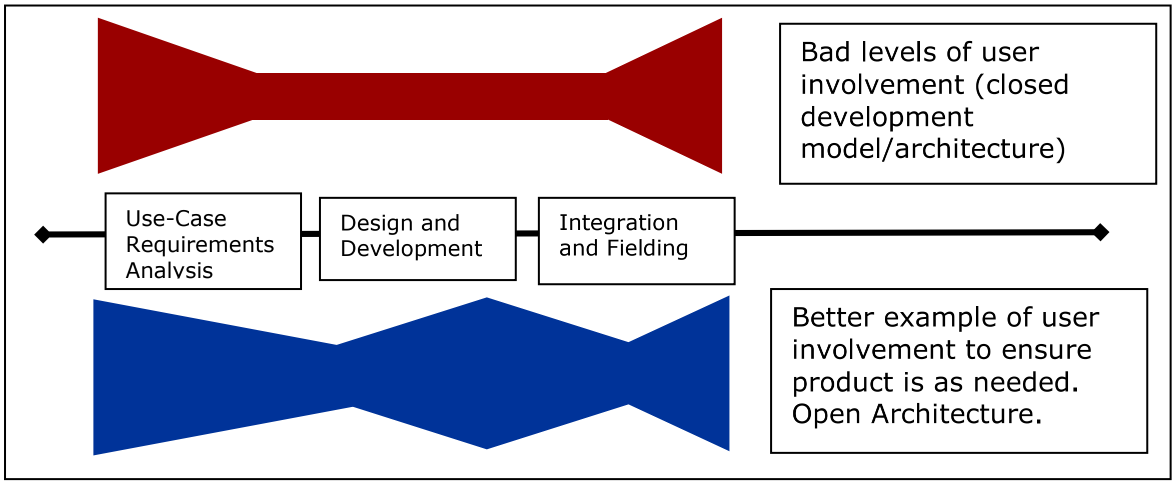
Still another, might be the SOA Silo effect which creates a beautiful net-centric system which does not interoperate with others because of proprietary, complex, or non compatible external interfaces.

Lastly, another pitfall to avoid is versioning. Since services are used by many folks on the network, to include external groups or programs, it is important to ask the question of what is impacted when a service definition changes. What will ensure backwards compatibility? How will a consumer of a service function if the service is updated with a newer version that does not match the previously established and coded interfaces? There are two ways to solve this:

- One is to build a service with versioning part of the protocol to access it (such as how browser can render HTML created from different versions of the specification by examining and acting upon header information).
- Another way would be to introduce a mediation/translation layer which re-directs calls. This is not as efficient and carries with it a heavy maintenance burden on that service broker component.

Customer (End-User) Input

To many times the end user never quite gets what they need or asks for. One reason for this is that while they may be involved in the beginning of the software lifecycle, the are rarely in the middle (preliminary/detailed design and Unit level testing).


Bad levels of user involvement (closed development model/architecture)
Use-Case Requirements Analysis
Design and Development
Integration and Fielding
Better example of user involvement to ensure product is as needed. Open Architecture.

[10] Grid Study for DISA: http://www.r2ad.com/papers/Grid-Study-R2AD.pdf

In the diagram above, the width of the graphic indicates the level of external involvement throughout the software lifecycle by end-user (warfighter) as well as others such as other programs, testers, auditors, and other developers from other projects that might be inclined to collaborate during the process to share ideas and best practices.

We propose that by keeping projects open throughout their entire lifecycle can greatly improve the overall quality and capability of the systems. Shared development environments such as SourceForge provide a team accessible web based development and collaboration space. End-users could then join the project team and provide feedback when it is needed most. Take advantage of the network to provide remote access to software in the various stages of development and test to get feedback.

Additionally, the end-user (the real customer) should be involved in all phases. Representation is critical at the key milestone events to include final testing and installation. Software Engineers would benefit greatly if they can spend time in the field helping to train, install, or solve issues with the end-user.

C2 XML Document

Ideally, all C4ISR systems would be capable of ingesting a single document format for all battlefield elements. It would behave much like a spreadsheet file does for Microsoft Excel. This concept has been around a number of years and many attempts have been made to create it, however without standardization adoption, the attempts have failed.   Recent attempts might be more successful, however it makes sense that eventually something will be adopted and therefore should be part of the migration strategy.

The XML document would have to meet certain requirements:

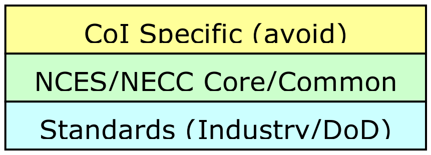
- Human and machine readable/parseable (XML tagged)
- registered MIME extension ("*.c4i")
- standardized/published and clear format description found normally in the Interface Control Document (ICD).

The document should also be capable of being very small, meaning it should not have very much overhead and could be as simple as <x:note>Insert note here</>. Embedded MIME data should be supported for attachments (such as SOAP with Attachments, etc). However these should be only really used for registered standardized formats (JPEG, NITF, MPEG2, etc).   External references should be allowed for both human and machine transversal.

A modified version of Microsoft Word and perhaps Adobe Acrobat should be capable of producing this document[11]. Word should be capable of opening the document and examining its contents. It should be capable of being emailed and also reachable on web sites (like PDFs).. It should be capable of being signed and encrypted. Bridges should be able to sanitize them with zero loss of referential data, meaning the pedigree should be encoded using a one-way token.

Standards

The migration path must include standards. Where standards fall short of requirements, they must be identified and involvement in a standards process (inside or outside the DoD) must focus on creating adoption via implementation and piloting.



[11] XML supported in Office 2003 and Adobe products

Cost-Benefit Analysis

COTS software is not the answer if the COTS does not follow the standards. It would "COTS" too much to keep upgrading and retro-fitting other code each time the vendor changes course to account for competitive, economic, or other commercial forces. Code re-use coupled with open-source initiatives (a form of code-reuse) along with the overall life-extending benefits of standards need to be weighed to maximize the tax dollar potential.

Examine alternatives before spending tax payer dollars to ensure that they money is spent wisely. Many times, the "Not-Invented-Here" syndrome causes extra money to spent without real reason. Try to adopt and re-use rather than always invent. Take the time and energy to create synergy between departments and programs to help share the cost. Be aware that this also creates dependencies and this has to be considered as part of the analysis. Sometimes, it is better to be independent.

VII. Terminology and References

The following outlines the key concepts and terms used in this paper.

IV&V: Independent Verification and Validation (IV&V) – unbiased review and testing
UML: Unified Modeling Language from the Object Management Group (OMG)
JCP: The Java Community Process by which Java technology is advanced.
NECC: Net-Enabled Command Capability, previously known as JC2.

Acknowledgements

Thanks to Chris Brown of Mercury Interactive, Peter Ziu of Northrop Grumman, for their constructive feedback and valuable input. We recognize David Watjen of the Defense Information Systems Agency (DISA) for the professional technical editing. Thanks to MITRE JFCOM for hosting the SOA Symposium, July 2007, at which many of these ideals were presented and discussed.

References

[WS-I BP 1.1] WS-I Basic Profile 1.1, web services stack
http://www.ws-i.org/Profiles/BasicProfile-1.1.html

[OGSA Arch] Foster, I., Kishimoto, H.,. Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J., and Von Reich, J.: The Open Grid Services Architecture, Version 1.0. GGF OGSA Working Group (OGSA-WG), 2005-2006. **http://www.ggf.org/documents/GWD-I-E/GFD-I.030.pdf**.

[AdaIC] The Ada Resource Association (ARA) maintains the Ada Information Clearinghouse (AdaIC) website. http://www.adaic.org/

[SDD] OASIS Solution Deployment Descriptor (SDD) Technical Committee web site
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sdd

[OGF] Open Grid Forum, web site: http://www.ogf.org