SSA Building Blocks –
Transforming Your Data and Applications into Operational Capability

Diane D. Buell
The MITRE Corporation
L. Shayn Hawthorne
The MITRE Corporation
James C. Higgins
USAF Electronic Systems Center 850 ELSG Advanced Development

CONFERENCE PAPER

The Electronic System Center's 850 Electronic Systems Group (ELSG) is currently using a Service Oriented Architecture (SOA) to rapidly create net-centric experimental prototypes. This SOA has been utilized effectively across diverse mission areas, such as global air operations and rapid sensor tasking for improved space event management.

The 850 ELSG has deployed a working, accredited, SOA on the SIPRNET and provided real-time space information to five separate distributed operations centers. The 850 ELSG has learned first-hand the power of SOAs for integrating DoD and non-DoD SSA data in a rapid and agile manner, allowing capabilities to be fielded and sensors to be integrated in weeks instead of months. This opens a world of opportunity to integrate University data and experimental or proof-of-concept data with sensitive sensors and sources to support developing an array of SSA products for approved users in and outside of the space community. This paper will identify how new capabilities can be proactively developed to rapidly answer critical needs when SOA methodologies are employed and identifies the operational utility and the far-reaching benefits realized by implementing a service-oriented architecture. We offer a new paradigm for how data and application producer's contributions are presented for the rest of the community to leverage.

1. BACKGROUND

The 850th Electronic Systems Group (ELSG), headquartered at Peterson AFB, Colorado acquires, fields, and sustains the warfighter's global sensing, communication and decision-making capabilities, including missile warning and defense sensors, global command and control systems, space control sensors and battle management systems. Headquartered in Colorado, there are also operating locations at Offutt AFB, Nebraska, Hanscom AFB, Massachusetts, and Dahlgren, Virginia. The Group administers a multi-billion dollar contract budget over DoD's fiscal year defense plan. The Advanced Development Division (XR) within the 850th ELSG identifies technologies relevant to Group capabilities and helps transition these technologies into current acquisition programs. An ongoing thrust area is the development and deployment of netcentric capabilities through the implementation of Service-Oriented Architectures (SOA), and the Advanced Development division has created various experimental and operational prototypes as demonstration tools.

A Service-Oriented Architecture (SOA) is a design model that encapsulates functionality within services. Services interact with a common communications protocol and are exposed via a standardized interface for others to use. A service is a well-defined unit of functionality that communicates via a standardized interface to other services. [1] A SOA is an architectural style that is intended to create an agile, integrated Information Technology (IT) infrastructure that can rapidly respond to changing needs by employing loosely coupled and dynamic applications (i.e., services). A SOA enables integration by loosely coupling the service implementation technology and platform with the interface that is used to invoke the service. A SOA uses standards-based interface components, removing the need for developers to understand the technologies used by individual services. This makes IT more flexible because standard interfaces enable services to be composed as business processes and to be reused by multiple processes. A SOA also simplifies integration across the IT systems by hiding the complexity of underlying service implementations. This increased flexibility and simplicity enables efficiency in integration efforts across organizations and across an organization's modernized and legacy applications. [2]

This paper describes the technical aspects of the SOA developed by XR and the capabilities demonstrated via prototypes (section 2). Section 3 describes the proposal to use the SOA to build operational Space Situational Awareness (SSA) capability in an experimental environment. Section 4 details our experience and lessons that we learned while implementing the SOA. We plan to apply this experience to implement a SOA for SSA data and applications.

.

2. TECHNICAL ARCHITECTURE

The XR SOA has demonstrated the ability to deploy a net-centric application using web-services, adapters, semantic processing, and an enterprise service bus (ESB). This SOA application loosely integrates a variety of disparate systems through rules-based event processing, mission visualization, and alert generation. The first implementation of the SOA created a service that was demonstrated during Joint Expeditionary Forces Experiment (JEFX) 08 – the Global Space Effects Information Service (GSEIS). Fig. 1 depicts the GSEIS architecture.

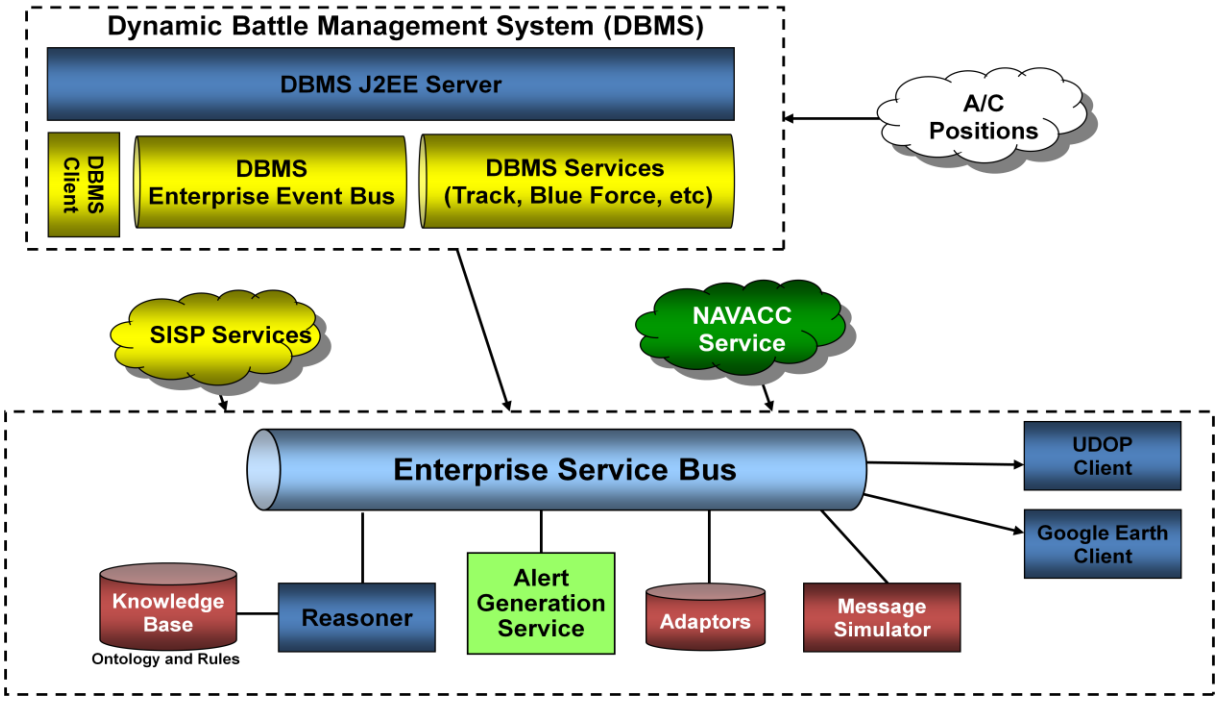

Fig. 1. GSEIS Service-Oriented Architecture

GSEIS provides a network-centric architecture, composed of agile web services and loosely coupled components. An Enterprise Service Bus (ESB) provides the backbone of the application, integrating adaptors that "speak" Extensible Markup Language (XML) - based messages. GSEIS also implements a "reasoner" that integrates heterogeneous data sources using semantic technology. New data sources are easily integrated into this flexible architecture, providing a path toward future joint capabilities. Multiple web patterns are supported, including Simple Object Access Protocol (SOAP), Representational State Transfer (REST), synchronous request/response, and event-driven messaging. Any XML-based client can be easily integrated; therefore, GSEIS is not limited to air and space synchronization alone. With GSEIS, we were able to: show how a loosely coupled architecture contributes to an agile operational environment; and investigate how ease of integration can lead to new operational concepts.

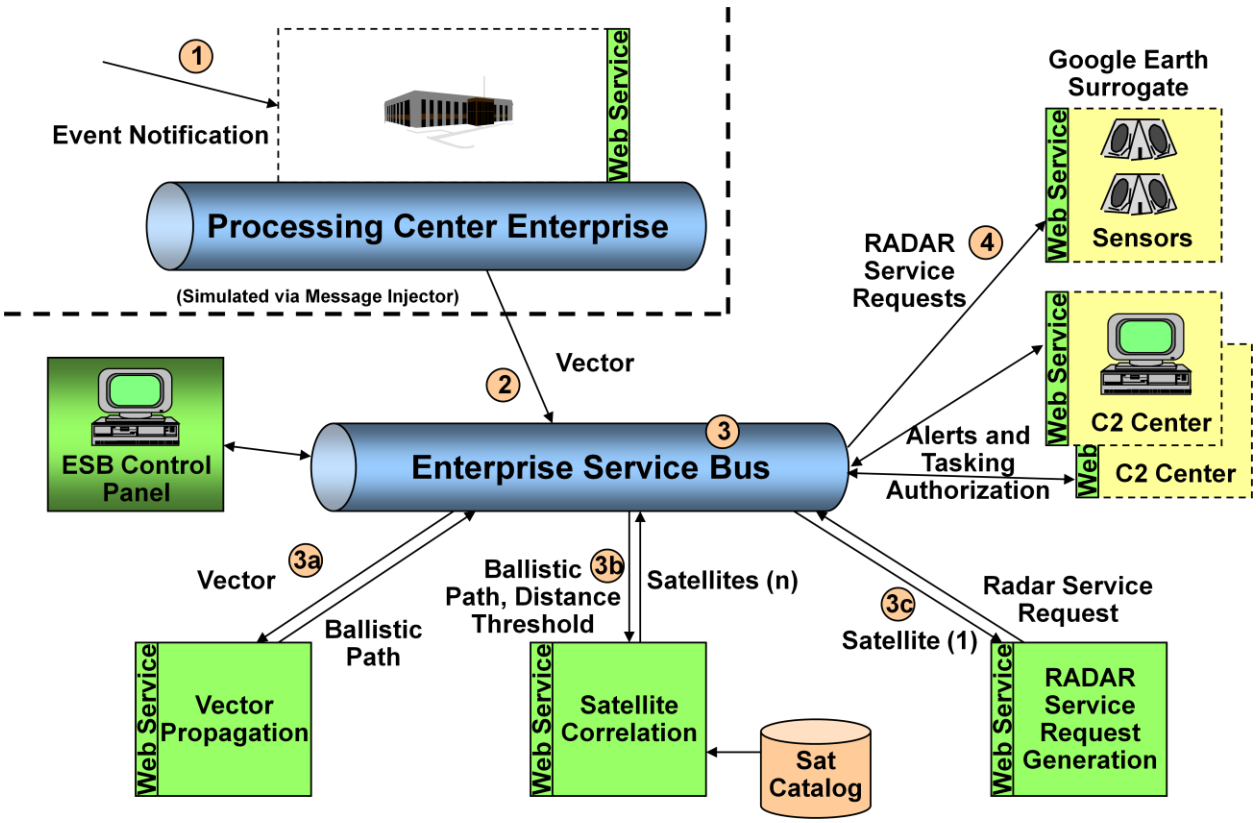

Fig. 2. SOA Reused for Rapid Sensor Tasking

The second implementation of the XR SOA, shown in Fig. 2, was used to create a rapid prototype to demonstrate how services could be orchestrated for efficient sensor tasking. The numbers in Fig. 2 show the chronological flow that is orchestrated by the ESB.

The same Enterprise Service Bus (ESB) was used in both instances, with the bus endpoints traded out for mission-specific services. By using this method, the Rapid Sensor Tasking prototype was developed in a matter of days. If services are available, they can be 'plugged in' to the ESB. Many applications are not yet available as net-centric services, i.e., the data they produce has not been standardized in format and content. There are currently several communities of interest working to define appropriate formats and schemas. For the current set of non-standardized applications, adapters need to be developed that translate the application's output into a standard format for ESB consumption.

In the same manner, the XR SOA has been used for an Integrated Missile Warning/Missile Defense prototype, and it is currently being used for the JFCC-Space Integrated Prototype (JSIP) and the Gapfiller Joint Concept Technology Demonstration (JCTD). The capabilities demonstrated through use of the same SOA span a variety of mission areas, including global air operations, rapid sensor tasking, integrated space surveillance, and homeland defense. Reusing the same building blocks for five disparate applications has enabled XR to appreciate the power and flexibility of a well-built service oriented architecture.

3. SSA BUILDING BLOCKS

The 850th ELSG would like to collaborate with SSA data and application providers to guide the implementation of a SSA SOA that enables C2. This will be the sixth successful repurposing of the architecture discussed above. Developing and deploying a net-centric application involves many of the same challenges as developing a "traditional" application, including capturing Concept of Operations (CONOPS) and requirements, high and low

level system design, software development, integration and testing, and finally customer acceptance testing. However, deploying a net-centric prototype involves a number of unique challenges. These include technical and information assurance issues related to developing software in an unclassified environment, and later deploying it to a classified network; relying on external systems for data critical to mission success; requirements management in a prototyping environment; and the impacts of data sources going offline at inopportune times. We will apply the experience gained, as described in section 4, to guide the development and implementation of the architecture.

We encourage SSA data producers and application providers to take the necessary steps to create services. We can help the producers to understand what this entails. The ideal world we are moving toward is one in which everyone will follow standardized conventions, but the near term work around is to accept the lack of standards implementation and develop adapters to bridge any disconnects.

We plan to 'plug and play' these services into our ESB in an experimental environment. We want to experiment with different ways to put the pieces together, similar to using building blocks to obtain many diverse creations. The variations that produce effective operational utility will be demonstrated to the user community. In this way, the investment necessary to operationalize the services and SOA will be narrowed down to those combinations that maximize operational usefulness. Additional benefits include allowing the SSA user community to experiment with changes in CONOPS and Techniques, Tactics and Procedures (TTP) that will be necessary as they transition to netcentric operations. A key benefit to the SSA data and application producers is the exposure of their work to the world. Once the services are created and registered, any authenticated user can access the service. This can result in a proliferation of available information for other SOAs to use.

In order to accomplish this, we propose the following:

- 850 ELSG/XR establishes a SSA SOA internet presence
- SSA providers contact 850 ELSG/XR to participate
- 850 ELSG/XR provides SSA providers web service design assistance and guidance, in the form of service design, interface specifications, and data schemas
- 850 ELSG/XR establishes each SSA service as an ESB end point when the service is ready for use
- 850 ELSG/XR orchestrates SSA services to explore benefits and publishes the results.

4. SOA IMPLEMENTATION

While implementing the SOA for each of the prototype efforts described in section 2, the team gained valuable experience that can be applied in future SOA implementations, for both prototypes and operational systems. This experience base has been captured in a set of 'lessons learned' that we will apply to the SSA SOA. This section describes the lessons we learned through the development and deployment of service-oriented architectures. A description of each pitfall is followed by several recommendations on how to avoid it.

Lessons Learned

1. Development Environment - When developing applications to be hosted on a classified network, the availability of an unclassified analog (that replicates the interfaces found within the target classified network) would greatly reduce integration risk. During the software development life cycle, depending on the model employed, approximately 50-70 percent of development resources are dedicated to software design and development. This is followed by an integration and testing (I&T) phase that requires the remaining 30-50 percent of available resources. In an environment where development takes place in an unclassified environment, and I&T takes place in a classified environment, this balance can be substantially upset. Problems with regression testing, incompatible interfaces, information format changes, and unanticipated information security requirements can lead to considerable re-work of already developed software. Examples include web services on a classified system that don't match their unclassified Web Service Description Language (WSDL) descriptor, or data interfaces/standards that differ across multiple networks. These problems are complicated by the fact that software "development" is typically not permitted on a classified network, due to Information Assurance (IA) constraints.

   Recommendation 1: To facilitate true net-centric solutions, especially for Service Oriented Architecture (SOA)-based applications, programs should develop unclassified analogs that duplicate the interfaces found

within the live, classified networks, so that developers can use them to support their development activities. Ideally, this unclassified analog exposes an interface that developers can leverage, thereby reducing I&T risk on the target operational network(s). This "development analog" concept needs to be funded by programs-of-record (PORs), and be subject to applicable service level agreements (SLAs). In addition to producing this development analog, the POR must also commit to supporting Operations and Maintenance (O&M) by keeping that analog online and up-to-date with regard to data interfaces and standards.

Recommendation 2: Construct classified development analogs that duplicate the actual interfaces found on the live, classified networks. These classified analogs would be similar to the unclassified ones mentioned above, except that they would have the ability to incorporate classified design specifications, data standards and interfaces, and other items that might be unavailable in an unclassified environment.

Recommendation 3: Creation of a classified development environment, separate from live, classified networks. A development enclave such as this could host a multitude of classified services and interfaces, allowing designers to effectively develop, test, and integrate their service(s) and application(s). Since this development network would not have access to a live, classified network, the IA accreditation process should have policies and requirements that are less rigorous and complex as the current IA standards a system must comply with to obtain an Interim Authorization to Test/Operate.

2. Information Assurance (IA) - IA places heavy technical and schedule risk on applications developed for classified environments. It is important to account for the appropriate resource allocation and schedule impacts in order to accommodate the IA Certification and Accreditation (C&A) process. Information Assurance (IA) consists not only of a government approval process, but also plays an important system design role. On the development side, IA expertise is critical to the development process in the understanding of impacts to software and hardware design. IA C&A often requires "locking down" certain operating system features, features that may be required for the application being deployed to work properly. ("Locking down" refers to the process of disabling certain features/services of the operating system in order to protect the system or network from malicious software, hacking, or memory leaks. In some cases, however, those operating services are necessary for an application to function properly.) In addition, the process of conducting "gold disk" scans requires application technical expertise to address issues that arise, and to understand the implications of system configuration changes that may be required by IA. These "gold disk" scans examine the hardware and software, including the operating system, application-specific software, data, and configuration files for security vulnerabilities that could compromise the integrity of the system or hosting network. Once the C&A process has been completed, the system has attained its Interim Authorization to Test or Operate (IATT or IATO). It can then be transitioned to the target environment, where IA expertise continues to play an important role. Inevitably, software configuration changes and security patches will be required to remedy problems encountered during I&T. In all likelihood, these changes will involve source code changes, configuration changes to Commercial Off-the-Shelf (COTS) products, and possibly configuration changes to the operating system itself. Each of these may have a serious impact on the application's IATT. As Lesson 1 documents, IATTs were not designed with the prototyping environment in mind. Furthermore, development practices, such as the ability to view, modify, and compile source code, are not allowed in practically all cases after a system has attained its IATT. This lack of flexibility suggests the need for a new type of IA accreditation that better facilitates software development, particularly in the emerging net-centric arena. Indeed, one of the sought after benefits of net-centricity and service oriented architectures is the ability to rapidly adapt an application to changing requirements. Without some kind of Interim Authorization to Prototype (IATP), the full potential of net-centricity and SOA will be unrealized.

   Another IA area of concern is user and system authentication. User and system authentication requirements must be designed into the system with detailed knowledge of the target environment. The Government is implementing Public Key Infrastructure (PKI), and progress varies from site-to-site. Some sites have little to no PKI infrastructure at this time and, therefore, will not be able to accommodate PKI requirements. Others may have a relatively mature PKI service and will not allow an unknown system, service, or application to interface with their trusted enclave without a certain level of PKI functionality in place. As such, deploying a single solution that works in one environment may not work in others. In a network-centric environment, the PKI challenge is complicated by the fact that many users may access the

application being developed, and those users may be distributed across multiple sites with differing PKI implementations. An example of the challenges of implementing PKI, specifically on a classified network, involves the distribution and transportation of user PKI certificates. While the certificates themselves are not classified, they are usually disseminated to users via classified email or on classified media. In an experimental environment like JEFX, it is necessary for users to transport their PKI certificates to deployed locations, but processes associated with couriering classified material (the media containing a PKI certificate) make this impractical. On the surface, it would seem that downgrading the PKI certificate would solve this problem, but once a CAC-like card holding the certificate is read on a classified system, that card now becomes classified, and the problem starts all over again.

Recommendation 1: Employ IA expertise on the application development team, beginning in the earliest design phases, and throughout I&T. Also, maintain a good working relationship with the C&A representative. This individual has seen numerous systems and applications go through the C&A process and can provide valuable guidance to successfully develop and evolve a system's security posture so that it satisfies the strict C&A criteria.

Recommendation 2: Establish software modification processes what will allow reasonable access to source code, compilers, and software debugging tools in the target environment. These processes must concurrently support the protection of the system while allowing software engineers to mature the system.

Recommendation 3: Develop prototype-specific IA accreditation that takes into account and supports software development. The creation and implementation of a prototype accreditation policy would allow software developers to more effectively design, test, and integrate net-centric applications and services in classified environments.

Recommendation 4: During the design and development phases, determine the PKI requirements for each site, so that the application can be designed, developed, and deployed accordingly.

3. Service Level Agreements (SLA) - SLAs must be employed in order to solidify technical relationships between systems and operational relationships between organizations. The identification of implemented standards is critical.

   The vision of net-centricity, and SOA in particular, implies the ability of applications to access and share services and information across a network. The information and service providers share (and consume) products of value to a wide variety of users, including unanticipated but authorized users. In order to provide a useable product, the interface to those products must be carefully defined, often in the form of a Web Service Definition Language descriptor, or WSDL. These WSDLs describe the technical aspects of a service interface, but they don't capture the "contractual" issues related to system availability, fee-for-service mechanisms, backwards compatibility, system reliability, system availability, system performance or throughput, change notifications, or system load support. These characteristics are formally captured in a SLA. In the absence of a formal SLA, a project has little or no recourse when a service or information source fails to function as expected. This can result in cost, schedule, and technical risk for the program, as well as operational risk to operators and other end users that rely on that service. For a prototype development, there may not be enough time to develop formal SLAs. However, it is important to spend time understanding the contractual items listed above for each consumed service and documenting them as much as is practical. One often overlooked element of an SLA is a statement of the conditions under which the agreement is to be terminated. While the termination of an SLA may be stated in simple terms such as a calendar date (which may be applicable for a limited duration prototype or proof of concept), operational programs of record may not have a known end date. Instead, vague verbiage like "until superseded" may be employed. Unfortunately, such a situation can result in poorly aligned operating schedules where a service can't be relied upon by other high availability, high reliability systems over the course of their system lifetime. In addition to the technical SLA just described, there is also a need for an organizational SLA. An organizational SLA, which is often implemented in the form of a Memorandum of Understanding (MOU) or Memorandum of Agreement (MOA), should be employed to formally capture roles and responsibilities of each party as they pertain to a net-centric solution. As an example, a MOA could be written to describe the teaming relationship between a prime contractor and sub-contractors, or between mission partners. As

they pertain to deploying net-centric applications, such an agreement should include a discussion of which party has specific responsibility for technical support and maintenance for each component of the system, and must also consider the burden associated with unanticipated users, software version changes, compatibility with end-user equipment, and evolving mission requirements over the lifetime of the system. Other issues such as staffing changes, availability of personnel with appropriate security clearances, and ever changing priorities by each signatory should also be addressed. Ideally, a comprehensive agreement would prevent the occurrence of "not my responsibility" disputes.

Recommendation 1: Implement comprehensive technical SLAs with all external systems providing services and information, including conditions under which the SLA will be terminated.

Recommendation 2: Implement organizational SLAs with all parties providing services and information, to include technical support obligations and accommodation of staffing turnover.

4. Requirements – Even for prototypes and experimental applications, formally capturing requirements and CONOPS, and vetting them through the user/operator community is crucial. Documenting changes will not only ensure that customer requirements have been accurately depicted, but that their translation into actual application capabilities matches the stakeholders' expectations.

   The requirements gathering and management process has a shortened timeline for an experimental prototype, as compared to a program-of-record. In our case, for JEFX, our team began with a list of identified capability gaps, generated an idea for an initiative that would address a set of gaps, and then proposed the idea to an evaluation board comprised of stakeholders and user representatives. Once our initiative was selected, we did not have the time or resources to gather operational requirements. We assumed the derived operational requirements based on the stated capability gaps in order to develop the first release of our prototype. The demonstration of this first release during JEFX Spiral 1 resulted in operator feedback that allowed for a refined set of operational requirements. In a full-scale development environment, a formal process for managing changes to requirements is typically employed. This process would normally be handled by a Change Control Board or Configuration Control Board (CCB). In a prototyping environment, a formal CCB is often impractical, but the process should still be performed, albeit on a smaller scale. Another topic related to requirements and CONOPS, specifically in a net-centric environment where end-users may be distributed around the world, involves the need to communicate the capabilities of the system to the user community. Much like the Internet, a SOA-based environment could soon generate new applications and capabilities to the warfighter at such a high pace that it may be difficult to formally train users fast enough. In these cases, it will be increasingly important to field applications with comprehensive context-sensitive help, plus on-line user guides and training material. Doing so will not only help the user community to understand the capabilities being provided, but it will also ensure that operators use the system in the most effective manner. In an experimental environment like JEFX, where deployed systems are often immature, this is especially true.

   Recommendation 1: Take the time to clearly document operational requirements, and make sure that these requirements are effectively communicated to the software developers.

   Recommendation 2: Frequently (1 time per month) demonstrate developed capability to users and/or user representatives. This will allow for requirements refinement based on development constraints and limitations. During these demonstrations, track which operational requirements are met fully, partially, or not at all.

   Recommendation 3: Prior to a formal release, verify that all operational requirements planned for the particular release have been met.

   Recommendation 4: Implement on-line user guides and context sensitive help to complement user training.

5. Redundant Data and Connectivity Sources – Net-centric applications rely on data feeds from external systems to satisfy their mission requirements. The robustness of those external systems, internal networks, and the networks of the partner systems is of paramount concern.

Ideally, any net-centric application would rely only on authenticated data sources from official programs-of-record (POR). However, there may be times when these PORs may not be available, due to planned and unplanned system/network outages, or when overburdened by high loads from unanticipated users. This loss of data/services could be compounded during a time of crisis when heavy system loading and network fragility exposes weaknesses in those systems. In order to assure system functionality, alternate data sources can be employed, even if they are not direct connections to the authenticated data source. "Third party" applications that act as information clearinghouses can be employed to share the connectivity burden, and to offload the authenticated data source, thus insulating it from potentially catastrophic failures.

Recommendation 1: Many data types have multiple potential sources. With the potential fragility of any network-based application (due to latency and offline data feeds, for example), incorporating the ability to have backup/failover data feeds is recommended. This kind of "A/B" switch provides necessary redundancy for mission critical systems.

Recommendation 2: Net-centric applications should have redundant paths to the backbone network (NIPRNET, SIPRNET, and JWICS) upon which they are dependent. These redundant routes should be taken advantage of to ensure that communications to the outside world are kept open.

Recommendation 3: Net-centric applications should employ data caching to minimize user impact of varying application/network loading and communication delays.

5. SUMMARY

Our team learned many valuable lessons regarding the implementation of SOAs. Our participation in JEFX gave us the opportunity to demonstrate a netcentric capability to five distributed operations centers and to serve many unanticipated users, allowing us to obtain valuable user feedback. We are applying these lessons learned to the programs within the ESC 850 ELSG portfolio to aid the implementation of netcentric solutions. We plan to apply this experience to the development of a SSA SOA for experimentation use.

6. ACKNOWLEDGEMENTS

Bygren, S. et al, Development and Deployment of Network-Centric Applications - Lessons Learned from JEFX 2008, MITRE Technical Report, MTR080041, May 2008.

7. REFERENCES

1 Erl, Thomas, Service-Oriented Architecture, Prentice Hall.

2 Semy, et al, MITRE's Perspective on Emerging Industry Service-Oriented Architecture Best Practices, MTR 080088, April 2008.