MTR06B0000055

MITRE TECHNICAL REPORT

Principles of Interoperability and Integration Volume 1: Fundamentals

August 2006

Harvey Reed, D520

Sponsor: 554th ELSW
Dept. No.: D520
Derived By:

Contract No.: FA8721-04-C-0001
Project No.: 0305400C-GC
Downgrade To:
Declassify On:

MITRE
Corporate Headquarters
McLean, Virginia

Approved by:

Abstract

This is the second edition of the MTR originally titled “Principles of Interoperability and Integrations”, now publicly re-released as :”Principles of Interoperability and Integration, Vol 1- Fundamentals”.

As work progressed after the original release, Net-Centric Conversations (NCC) was greatly expanded upon. In addition, Nodes combined with Discovery Scopes to become Sub-Enterprises. These two topics are the basis of “Principles of Interoperability and Integration, Vol 2- Net Centric Conversations and Sub-Enterprises”. This new 2nd MTR will be published early Fall 2006, and will also be publicly released.

The promise of net-centric warfare is based on the assumption that services (especially web services) can freely converse amongst themselves, and exchange data via messages. The reality is that this is a hard problem, due in no small part to the realities of how we organize services to run within Nodes. In addition, DISA NCES efforts have encouraged us to view the world through nine service types, and much of the discussion has been stove-piped, focusing on one service type at a time.

This paper brings together net-centric elements of messaging, discovery and security and examines the minimum coordination to achieve a net-centric conversation between services. This coordination becomes the limiting factor in how quickly we can create, update, or delete a net-centric conversation. In fact we can define an ***agility metric*** for the enterprise as: “the highest sustainable rate of creating, updating, and deleting net-centric conversations”. An example can be Weather service. If your organization wants to use it (their web service) do you expect to be able to make this connection within months? Weeks? How about minutes or hours? That is what we need to strive for to meet an agile, small adversary.

1 Executive Summary

Interoperability and integration is the exchange of messages[1] between systems. Previously, systems interoperated by constructing unique interface agreements. This created a field of entrenched stovepipes with three properties:

- The field of stovepipes is not agile – they cannot form new relationships easily.
- It is hard to map "mission threads" through the field of stovepipes because of the unique nature of each interface.
- Increasing overall connectedness creates "n-squared" point-to-point connections.

Recent innovations concerning sharing information[2] and particularly "Communities of Interest"[3] enable groups to regularize the process of creating and maintaining a shared vocabulary that all participants can use. In addition to enabling broader human understanding of community topics, these shared vocabularies can be used to constrain the payload of "machine-to-machine" messages exchanged between services so that all participating services can understand and process the payloads.

The "net-centric conversation" is the key enabler of interoperation and integration between services in nodes. A net-centric conversation is more than just exchanging messages in a "mission thread". It is also a set of enabling context required for the conversation to mechanically take place, as well as a set of dynamic processes to maintain the context. Moreover, the exchange of messages can span the enterprise, not just between two services.

Agility of the enterprise is defined as the ability to change and/or form new net-centric conversations rapidly. We need to be able to add/subtract participants to existing net-centric conversations, as well as create new net-centric conversations. The processes to create and maintain the enabling contexts of net-centric conversations depend on organizing nodes around services and other nodes, and the interplay with various types of discovery services. The end result is a scalable view of an enterprise architecture, with dynamic processes to support agile net-centric conversations.

This paper concludes with appendices:

- Foundations – summary of the Enterprise Service Bus.
- Architecture Proposal – create an enterprise master capability list automatically as a side effect of the net-centric conversation enabling processes.
- Generic Data Center Proposal – use generic data centers a means to accelerate the velocity of net-centric acquisition.

---

[1] The choice of vendor or product used for the hardware or software is not relevant and is not a consideration. We assume the use of contemporary standards, such as JMS, SOAP, WSDL, BPEL, etc. that enable the technical exchange of messages regardless of platform software. All major messaging vendors can "bridge" between each other.

[2] DoD Directive 8320.2, "Data Sharing in a Net-Centric Department of Defense", 12/02/2004

[3] MITRE Technical Report, "COI Handbook: Practical Guidance for Communities of Interest (COIs) Implementing the DoD Net-Centric Data Strategy", December 2004, Dr. Scott Renner, Dan Hebert, Steve Rainer, John Wilson, authorized distribution only

2 Forward

This paper has evolved from the previous v0.3 version[4] internal paper which was the basis of the GCSS-AF Enterprise Service Bus to the current version which forms the basis of the future GCSS-AF direction of nodal interoperability, including the ESB "Extension Cord" (see Appendix – Enterprise Services). Much of the v0.3 paper has been updated, condensed, and re-packaged as an Appendix – Foundations.

This paper now incorporates much of the enterprise architecture "soundtrack" that I have used in the full-day ESB training sessions given at Maxwell AFB Gunter Annex, Wright-Patterson AFB, Randolph AFB, Air Staff, and other venues such as TEMs. The overall value of this paper is to collect in one place the current thinking of one of the few "generic data centers" in the Air Force, and perhaps DOD (see Appendix – Generic Data Center Proposal).

All of the principles discussed in this paper have general applicability and are not dependent on any feature of GCSS-AF. Occasionally a capability in GCSS-AF will be cited to illustrate a point, since GCSS-AF is a pathfinder for using an Enterprise Service Bus, and other innovations. Also I use the word Node to mean a "place to run services", and do not mean to denote large datacenters such as an AOC or GCSS-AF. A Node could just as easily be a computer in a backpack for an Army Ranger.

The biggest value of this paper is to expand and reinforce the principle of interoperability and integration based on "net-centric conversations"[5] via messages. The value of a net-centric conversation is the recognition of the importance of interconnectedness and patterns of interaction over a system optimized in isolation[6]. Much of the discussion in this paper serves to define the basis of "interoperability" for two or more Nodes (and their services) to work together, that uses "data interoperability" at the message level. This moves us away from the "tyranny" of n-squared highly specific interconnects[7], and provides "effective linking" which is one or the core foundations of Net-Centric Warfare[8].

There is still much work to do on this paper. Many people previously have attempted to define interoperability and integration, and I recognize that it is difficult to create a general theory of interoperability and integration that will be accepted throughout MITRE, and indeed throughout the engineering profession.

Thus, the approach is not to create a general theory, but rather to describe and constrain the problem, with a practitioner approach. In addition to posting on the USAF Portal, I am posting this on the MITRE SEPO portal to extend the collaboration and harmonize

[4] This version is posted on the "GCSS-AF ESB and Integration" Community of Practice in the USAF Portal, as well as the author's MITRE transfer folder.
[5] This is a term we use in GCSS-AF to focus discussions concerning integrating systems
[6] "The Agile Organization", Atkinson and Moffat, CCRP, 2005, in particular see Fig 7.4 and related text.
[7] "Power to the Edge", Alberts and Hayes, CCRP, 2003, in particular see Chapter 7 and Fig 16.
[8] "Net-Centric Warfare" Alberts, Garstka, and Stein, CCRP, 1999, p.91

Table of Contents

List of Figures

3 Principles of Interoperability and Integration

The most compelling reason to gather all of the principles in one place is to create a practical model of how to create and grow a net-centric enterprise. This net-centric enterprise has many moving parts and is complex. The biggest challenge from a human psychology point of view is to find ways to reduce complexities in ways that actually help9.

The goal is to outline a few important organizing principles. These principles reduce the complexity by means of scaling, identifying key points, etc. In short, this means reducing the net-centric enterprise to a manageable number of factors.

3.1 Tenets

The basic tenets of our principles of interoperability and integration consists of one big assumption, a set of things that we hold to be true, and a series of constraints that we impose an order to make this exercise tractable[10].

Our one big assumption is that we are not creating a general theory. Rather, we are creating a practical approach to construct new systems out of existing systems, by enabling net-centric conversations between the services of these systems. This forges a way ahead to a net-centric future.

The set of things that we hold to be true, come in the form of foundational elements. These foundational elements are well understood throughout the Air Force and DOD community and form the basis of our constructionist approach. These things are:

- IP messages are delivered via the GIG.
- Mission capability is delivered via services. These services may be modern[11] or legacy. A legacy system can be considered a service as long as its interfaces are well-managed and adapted into modern standards. This is a function of program management discipline, not a technology limitation.
- Our constructed architecture will be compliant with NCOW-RM.
- The enterprise services that we describe will be compliant with NCES.

[9] "The Logic of Failure", Dietrich Dorner, Metropolitan Books, 1996, p. 185. Dorner outlines basic failure modes in psychology experiments where subjects had to mange complex systems.

[10] This document aligns with, and goes into interoperability detail first outlined in "Net-Centric Enterprise Solutions for Interoperability (NESI)", Navy PEO C4I & Space RAPIDS Team, Air Force ESC C2ERA Team, March 2005

[11] The use of the term "modern" here refers to contemporary techniques of constructing services that comply with standards such as JMS, SOAP, WSDL, BPEL, etc.

A series of constraints make our constructionist approach tractable. The constraints are as follows:

- We only consider service-oriented architectures.
- We only consider Air Force and DoD IT intensive systems.
- We do not consider the problem of data interoperability and integration, except for the use of COI vocabularies in message payload[12].
- We use a system-of-systems approach to build larger systems out of smaller systems.

We also employ scaling techniques to reduce complexity. We envision that our constructionist approach to the principles of interoperability and integration will be a tactical complement to strategic net-centric initiatives such as NCOW-RM and NCES.

3.2 Realities

3.2.1 Social

Any system-of-systems construction technique must be in harmony with the social realities of the people and organizations behind the systems.

The first and most important reality is that people build systems to reflect themselves. The reason this is important is that we can examine social systems to determine typical grouping constructs.

For example the most common grouping construct for closely related people is peer-to-peer and hierarchical[13]. In fact, we see this organizational principle, extensively applied within the DOD. Of course within the DOD there is no single top absolute group; rather it is a forest of top groups. Indeed, we can look across the larger DOD enterprise, which includes other government partners, coalition partners, industry partners, and academic partners and see that this pattern still applies.

In addition, we need to keep in mind that the peer-to-peer and hierarchical grouping is constantly changing. This puts stress on tightly coupled systems that mimic an old grouping, but need to change to a new grouping and cannot do so easily. This makes the ability to form new net-centric conversations a high priority.

We also see this grouping effect influencing the vocabularies of the participants. In recent terms, we call these groups communities of interest or COI. The social fact that there could never be one complete single internally resolved vocabulary has a big impact on the physical realities of systems as we will see below.

When we look at the social function of funding and acquisition patterns of DOD systems, we note that much of the IT funding goes to specific, some would say stovepipe, systems. Thus, initiatives such as NCOW-RM are crucial, since they provide strategic, high-level net-centric patterns across stovepipe programs to move toward a net-centric future.

[12] Terry Blevins of MITRE is in the final process of publishing a data interoperability paper
[13] "Why Hierarchies Thrive", Harold J. Leavitt, Harvard Business Review, Mar 1, 2003

3.2.2 Physical

Any system-of-systems construction technique must also be in harmony with the physical realities of IT systems.

The first item to note is that most people would probably be more comfortable with one system rather than many systems. The reason for this is simple, a single system allows for more effective command and control.

Two factors work against having one system. The first is social as we have seen above. Groups that have authority delegated to them typically want their own systems with their own vocabulary or COI. The second is physical. There are limits to how big a single system can be. These limits have to do with memory, disk, span of database transactions, and the latency of user interaction. Thus, even in a single data center, we have multiple servers. We also divide data centers around centers of authority, and give edge servers to tactical users[14].

What we are left with is the social and physical reality that there can never be just one single system. Rather, we will always have many systems, organized similarly to the organizations behind them. Thus, our constructionist approach must be able to take this multitude of systems, and using organizing principles create agile higher level systems out of lower level systems, that are capable of participating in flexible net-centric conversations[15].

3.3 A Net-Centric Conversation requires Context and Process

NET-CENTRIC CONVERSATION: An exchange of messages with associated enabling context as well as a set of dynamic processes to maintain the context.

A net-centric conversation is the exchange of messages among services. A conversation can be weaved through many services, across many security domains, and have human-in-the-loop guidance at critical points.

The agility of our enterprise depends on being able to rapidly create and modify net-centric conversations, and thus the path of information sharing among participants. The robustness of the conversations depends on well formed message exchanges, particularly in the case of SOAP/WSDL. Even when following standards, there are interoperability gaps. We will not cover them here, and the reader is referred to best practices[16].

The net-centric conversation can only take place when all services that need to interact with each other share the enabling context, and can understand each others message context.

---

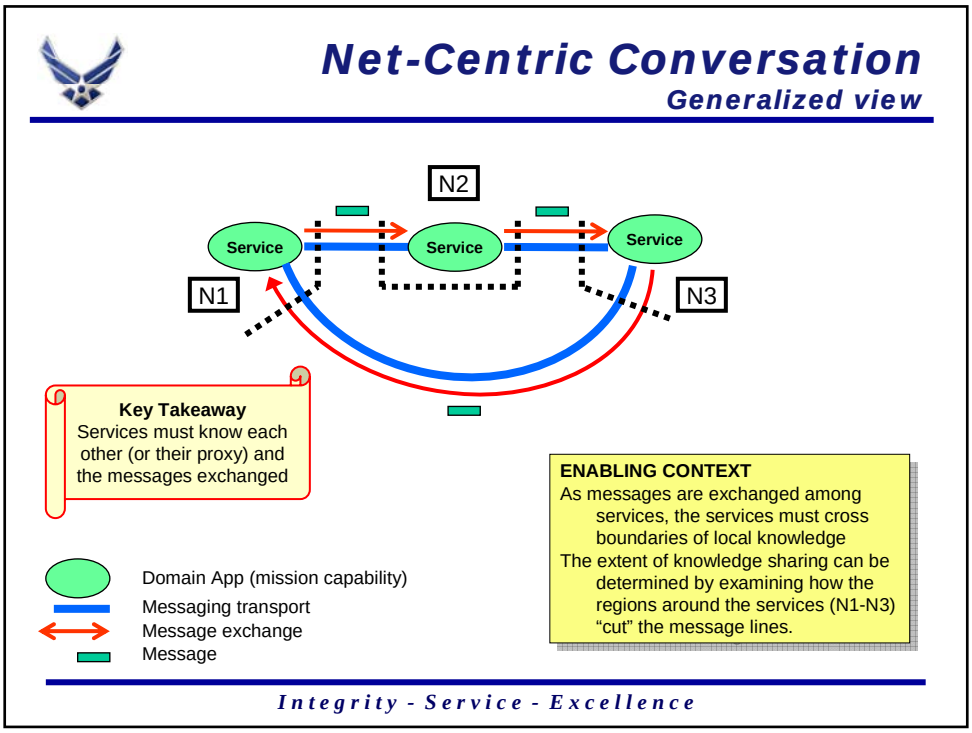
[14] A good example of this is the use of Akamai edge servers world-wide by GCSS-AF to complement the data center enclave that is distributed between Gunter Annex and Wright-Patterson AFB

[15] See Appendix – Generic Data Centr proposal for discussion about the economics of consolidation. GCSS-AF has seen tremendous economies of scale. That being said, there will never be one big data center for the whole DoD, and this paper explains how to glue everything together, whether it's a generic data center, or a more specialized system.

[16] "XML Schema Best Practices, Making Web services Interoperable", Jason Mathews, MITRE April 2004, MITRE Technical Report # 04-0508

3.3.1 An example net-centric conversation

Looking at a net-centric conversation from a topology perspective, we can see the services as the points in a graph, and the messages exchanged over the arcs. When we "cut" the message exchange lines entering or leaving a service, we see all of the required information that must be exchanged prior to being able to exchange messages.



3-1 Generalized View

3.3.2 Enabling Context for Web Services

The enabling context for the net-centric conversation is the set of mutual knowledge among the services of each other, and of the messages, especially the payload. The payload is the content (meaningful part) of the message; the rest is overhead for management by the enabling middleware software. The enabling context and message context are not identical for all net-centric conversation types (JMS, SOAP, BPEL), but there are enough similarities to start with the web service. The term "endpoint" refers to the "address" of the entry point of the service.

Item	Use	Notes
Called endpoint	The calling service uses this to send the SOAP request	Typically expressed as a URL. For inter-nodal calls, this URL is proxied
Callback endpoint	If using asynchronous messaging (preferred), the response will come back in a separate message, thus the caller needs to expose an endpoint	Typically expressed as a URL. For inter-nodal calls, this URL is proxied
Authentication	The proxy of the called service uses this to determine the identity of the caller.	Typically expressed as X.509 certificates. if the caller and called service are in the same node (not proxied), authentication will not be required on each call
Authorization	The proxy of the called service uses this to determine the identity of the caller.	Typically expressed as “roles”
Semantics	The called endpoint must understand how to write a message, especially the payload.	The payload is consumed in the WSDL, and described as annotated schema

3.3.3 Message Context

The message context is that part of the exchanged message, in addition to the payload, necessary for the cooperative sharing of messages. The issue of a consolidated list of URLs is handled in the Appendix – Modest Architecture Proposal. Below is an example of message context. The author suspects this context may vary from one Node to another, and thus the context must be transformed at runtime when conversing between Nodes.

Item	Use	Notes
Payload		See above enabling context
Calling tracer	A trace marking in the message denoting the calling service	Typically URLs to the service repository of the calling service
Payload tracer	A trace marking in the message denoting the payload	Typically URLs to the metadata repository of the called service

3.3.4 Dynamic Processes to Maintain Context

We employ a dynamic process to share enabling and message context between services to maintain net-centric conversations. Part of this process shares discovery registry information between Nodes, which happens before runtime, and is covered below. Part of this process will be to transform the message context from the conventions of one Node to another, which will happen at runtime.

3.3.5 Three Types of Net-Centric Conversation

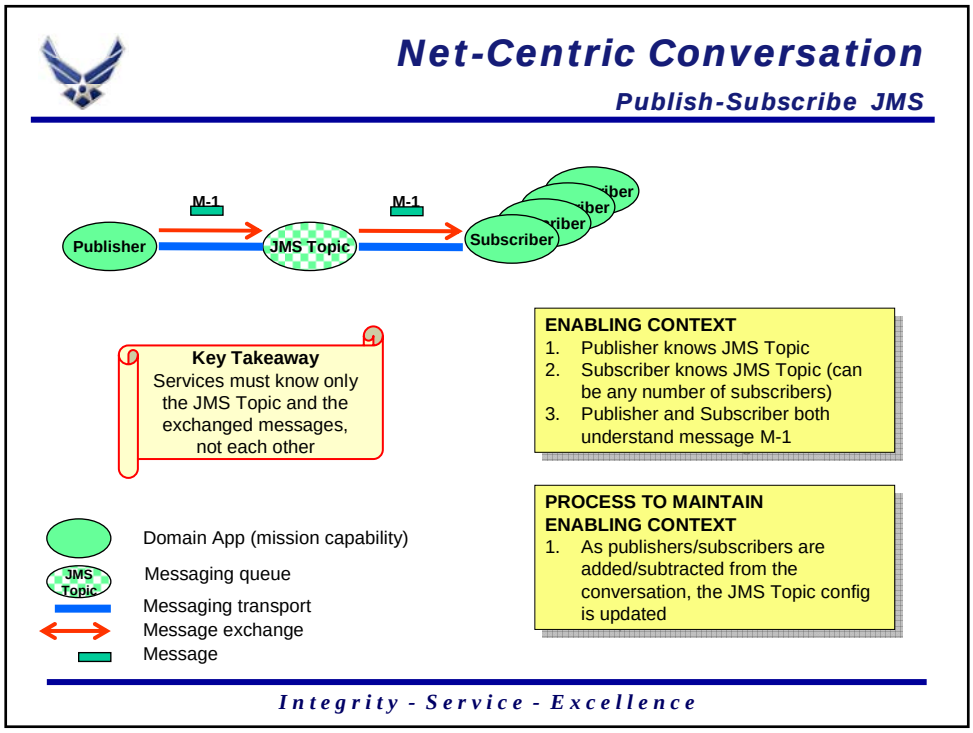
Before we examine the mechanics of sharing required information, and the organizing principles we can apply, we will review the three basic building block types of conversation:

- Publish/Subscribe
- Request/Reply
- Business Process

These are described in more detail as methods of integration in the appendix – Foundations/Enterprise Service Bus, at the end of this paper.

The first type of conversation is Publish/Subscribe using the JMS standard. A publisher sends a message to a JMS Topic which acts as a proxy for any number of subscribers. In addition, there are various levels of guaranteed message delivery. The subscriber receives the message from the JMS Topic which acts as a proxy for the publisher. There can be any number of subscribers. In this type of conversation we see that the knowledge sharing is focused on the JMS Topic. When a new subscriber is added/subtracted, the publisher doesn't have to know or change anything. The subscriber must assure it can understand the message when it subscribes, and the JMS Topic must be configured every time there is a change.

The error handling of this type is simple. The JMS Provider typically offers guaranteed messaging so that the publisher sends once, and doesn't have to test for successful delivery. Conversely the subscriber doesn't typically poll; it just waits on a queue for the message to be delivered with assurance of delivery.



3-2 Publish Subscribe JMS

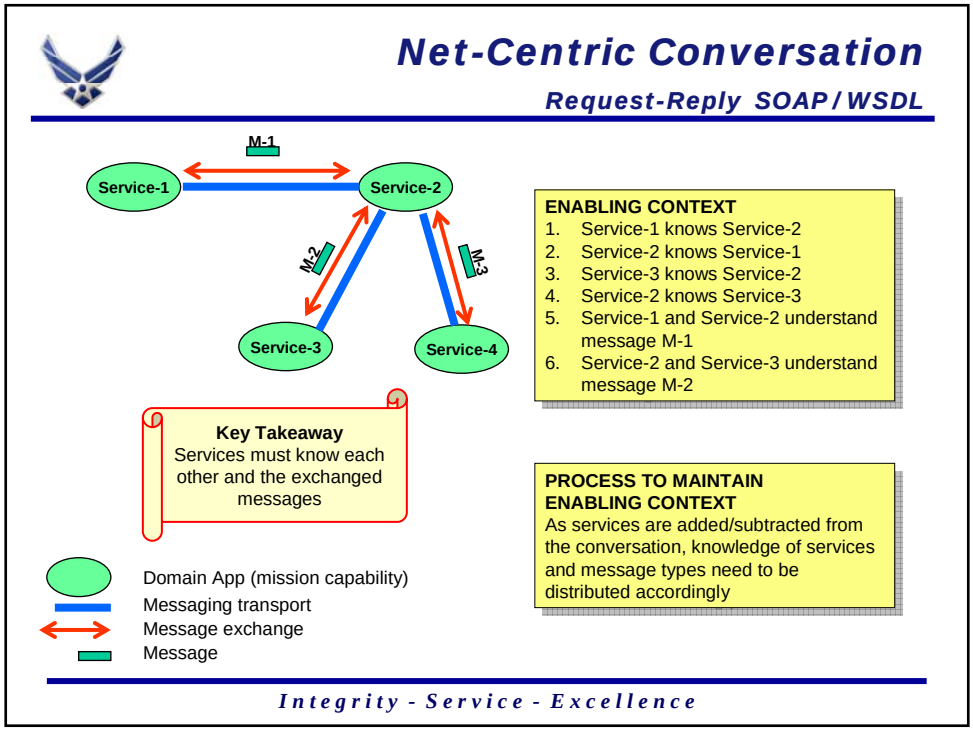
The second type of conversation is Request/Reply using the SOAP/WSDL standard. A requester sends a message directly to the replier. The replier can "block" and send an immediate response (synchronous), or send an immediate "ack" back, and send the content of the response back as a separate message (asynchronous). Asynchronous SOAP messages are preferred, and lead to a more loosely-coupled style of messaging.

In this type of message exchange we see that the burden of knowledge sharing is on each service. When participants are added/subtracted the mutual exchangers must share knowledge. Thus the effects of changing the conversation can be large.

The error handling of this type of conversation is complex. This is a serious weakness of this type of conversation. The error handling comes in two parts:

- Error handling related to protocol weaknesses – SOAP is usually transported over HTTP is weak and does not guarantee delivery of messages.
- Error handling related to business logic – consider that a single conversation can span multiple services, with splits (no joins usually) and that multiple development organizations are involved. There is no feasible way to assure uniform error handling across an entire conversation.

The transport error handling can be mitigated by using guaranteed messaging to transport the SOAP message at least across a WAN, where timeouts and other problems are more likely. Error handling in complex conversations with a lot of business logic can only be mitigated by using a business process engine, such as a BPEL engine, our third type.



3-3 Request-Reply SOAP/WSDL

The third type of conversation is a business process using the BPEL standard, which in turn relies on the SOAP/WSDL standards. A BPEL engine receives a message to initiate a business process, which is described in BPEL (an XML extension). The BPEL engine, reads the BPEL file, and orchestrates message exchanges with the participating SOAP/WSDL services. The orchestration expressiveness of BPEL is extensive, and a business process designer can express scopes,

sequence flows, splits, joins, waiting for a message, etc. All of the interaction external to the BPEL engine assumes web services via SOAP/WSDL.

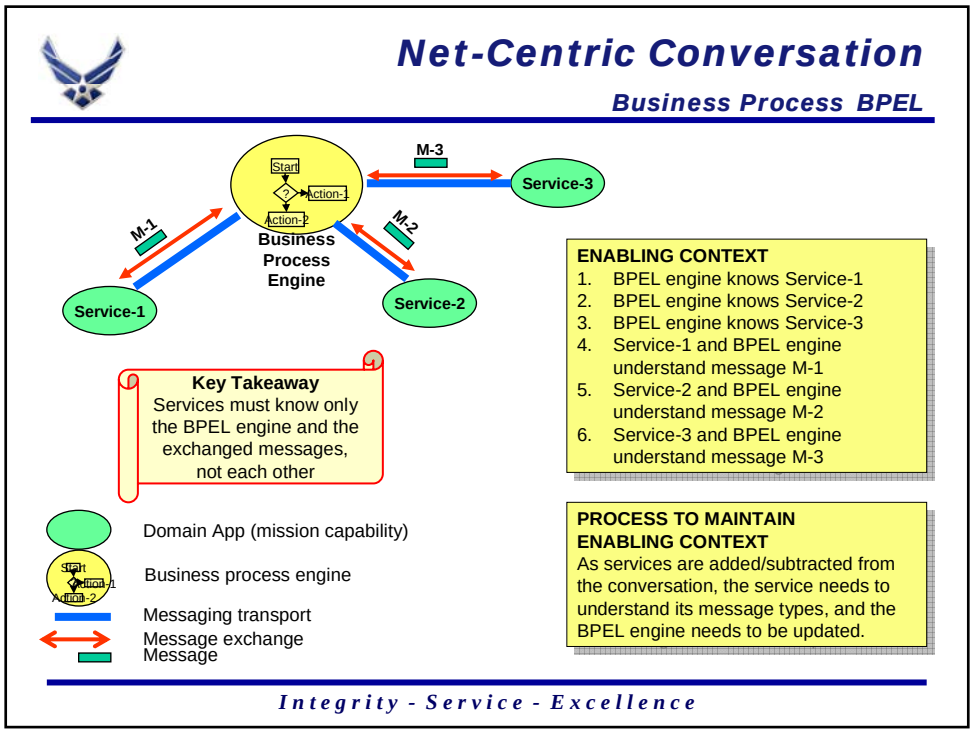
In this type of message exchange, the BPEL engine knows about the WSDL interfaces of the participating services, and the services know about the BPEL engine, but the services don't have to know about each other. This alone greatly increases the agility of our enterprise. When

participants are added/subtracted, it is easier to share the necessary knowledge in this type rather than with a pure web service SOAP/WSDL set of calls.

The business logic error handling of this type is superior to pure SOAP/WSDL calls. Built into the BPEL language are extensive provisions for fault handling and compensation. It is the most suited for long-running transactions of all the types, since it is based on the SAGA long-running transaction model[17].

[17] "SAGAS", Hector Garcaa-Molrna, Kenneth Salem, Department of Computer Science, Princeton University, International Conference on Management of Data,
Proceedings of the 1987 ACM SIGMOD international conference on Management of data

The transport error handling can be mitigated using the same technique as with SOAP/WSDL where messages can be transported via guaranteed messaging rather than HTTP.



3-4 Business Process BPEL

We can summarize the uses of the conversation types for building net-centric conversations as follows:

Conversation Type	Messaging Pattern	Ideal Use	Weakness
Publish/Subscribe JMS	1 Publisher → Many Subscribers	Disseminating periodic reports or events. Publisher doesn't have to know who is subscribing[18] As a convenience, subscribers can subscribe to a wildcard (*) set of topics, so they can get messages from new topics as they are added	Not meant for "chaining" together in a long conversation[19]. Subscribers however can respond to subscribed events by kicking off reactive business processes (some threshold was exceeded for example) as a form of BAM
Request/Reply SOAP/WSDL	1 Requester → 1 Replier A Replier can transitively make additional requests, although this is discouraged	Simple request/reply, no extensive chaining This is useful when there is too much data to publish everything, rather let other services query for what they want	HTTP weakness (no guaranteed messaging) Complex business processing makes error handling challenging
Business Process BPEL	Many → Many via BPEL engine	Complex long-running transactions orchestrated among a number of SOAP/WSDL services	Not suited for simple (one-hop) request reply[20]

[18] Its worth noting that early use of Publish/Subscribe JMS within GCSS-AF tells us that the people organizations want to know who is subscribing. This is fine, and doesn't detract from the agility of the conversation type. This paper focuses on the machine-to-machine aspects of net-centric conversations.

[19] This is notwithstanding the fact that the initial use of Pub/Sub in Operations Support is doing crude bulk message chaining as an emulation of current file handling. This gets participants on the ESB, where they can continue to evolve independently, and eventually exchange change records via SOAP/WSDL in preparation for the introduction of ERP BPEL driven business processes.

[20] A developer could make an argument that every web service call is a one-hop interaction. However if a SOAP/WSDL call can be correlated to an earlier active call (certainly the case for transitive calls), then we will

A service as a participant in a net-centric conversation only performs three functions:

- **Process data** – a service gathers data, compare, transform, merge, possibly sending an event if a threshold is exceeded. A typical example is a publisher or subscriber.
- **Answer query** – a service is an info provider. If there is too much information to publish everything, then answering specific queries is the preferred approach, and is far superior than directly querying a database via SQL, esp. over ODBC. This is a key point of the net-centric principles. Any data exchanged between services should be of the format of agreed message payloads, for example a COI XML change record message format, not the internal database tables of a particular instance of backing store. If you need to directly access a database table, then you are really part of that same service or database.
- **Perform command** – a service wants to update one or more other services to create an "effect" or update information. If the processing is complex, then consider using a BPEL engine and a business process to implement the logic.

Next we will review a set of organizing principles to organize services within nodes. This constrains and simplifies the process of sharing knowledge, which is a precursor to an agile enterprise.

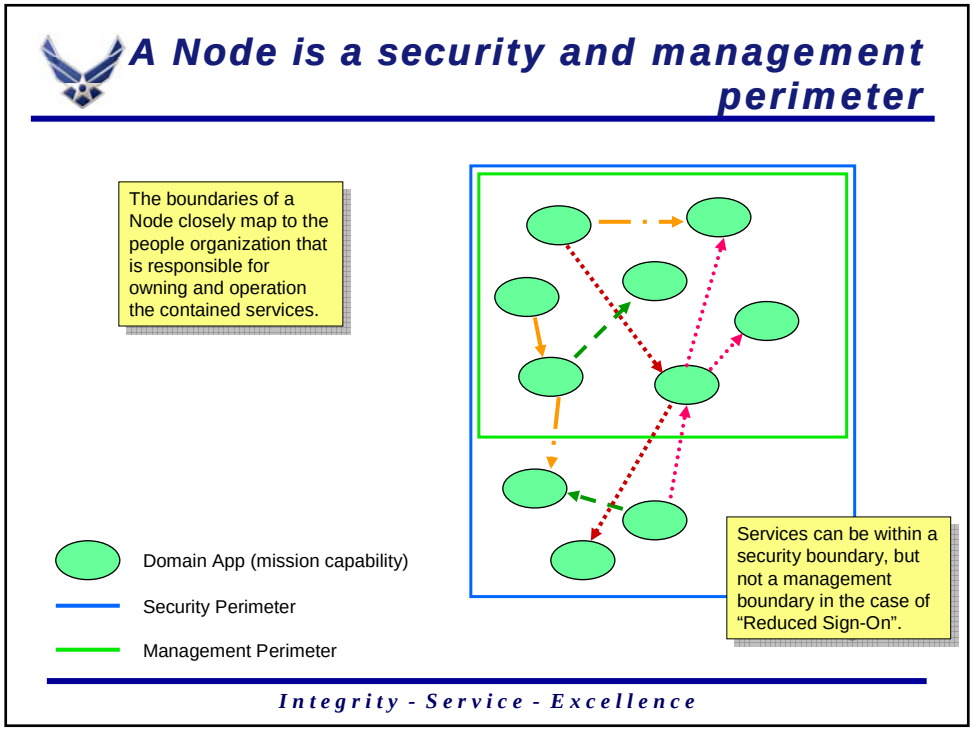
3.4 Nodes Integrate Services

INTEGRATION: A Node is a security and management perimeter which integrates a set of services.

This perimeter maps directly to the people organization that has the authority and responsibility to own and operate a set of mission capabilities embodied in the services. In simple terms, this organization will typically put the services behind a firewall (security), and also be accountable for the design, develop and runtime characteristics of the services (management).

---

say that is not a one-hop call, but part of a larger unit of preceding work, and must be correlated. This is the basis for the complexity of error handling.



3-5 A Node is a Security and Management Parameter

This definition is service-oriented and abstracts away how a node is actually constructed. This is important, because we want to include the hundreds of existing legacy systems that in effect, function as or within nodes. We will describe organizing principles within a node that can be used to modernize existing nodes, as well as construct new nodes.

The two organizing principles within a node are the enterprise service bus[21], and service management[22].

The enterprise service bus integrates services by providing three methods of integration: (a) publish/subscribe; (b) request/reply; and (c) business process or mission thread. Service management assures that each service is consistently managed through the acquisition, design/development, deployment, runtime, and sunset phases.

The enterprise service bus is an SOA equalizer. Any service (agnostic to type of service, modern or legacy) can "adapt" into the ESB and participate in one or more methods of integration. The enterprise service bus as an architectural construct is not at this time completely defined by standards. However, there is common industry agreement that an ESB supports:

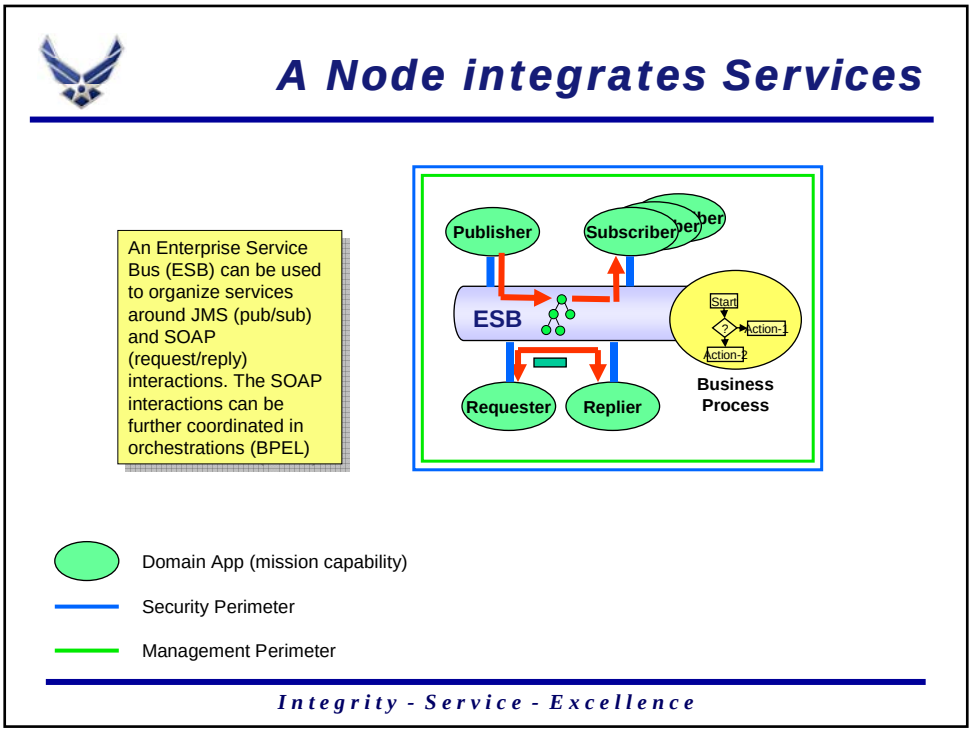
- **Messaging** – Guaranteed messaging between participants.
- **Routing** – Ability to route Publish/Subscribe and Request/Reply messages to their endpoint destination without sender knowledge of the ESB topology.

[21] See Appendix – Foundations
[22] See Appendix

- **Transformation** –Transformation between formats and protocols so participants can consume each others payload. This does not include large scale transformations such as provided by data warehouses.
- **Management** – Central management of the federated distributed topology of the ESB is necessary for scalability.
- Thus an ESB is a distributed and federated, yet centrally managed message bus that supports transformation of payload. We can think of an ESB as a guaranteed messaging plumbing system.

Service management recognizes the power of interfaces[23]. The interfaces are the unit of change management, and requirements. Service interfaces are also the unit of adaptation into the ESB. The implementation of the service is not important for using the ESB, but maintaining a constant service interface to the ESB is important. We recommend that the service interfaces are designed for exchanging small XML messages that reflect transactional changes, and answers to queries, as opposed to the current bulk file exchange common in Air Force operations support systems.

Below we see how a node integrates services. The two primary methods of integration are JMS publish/subscribe and SOAP request/reply interactions. Further, the SOAP interactions can be orchestrated in a BPEL business process.



3-6 Integrated Services

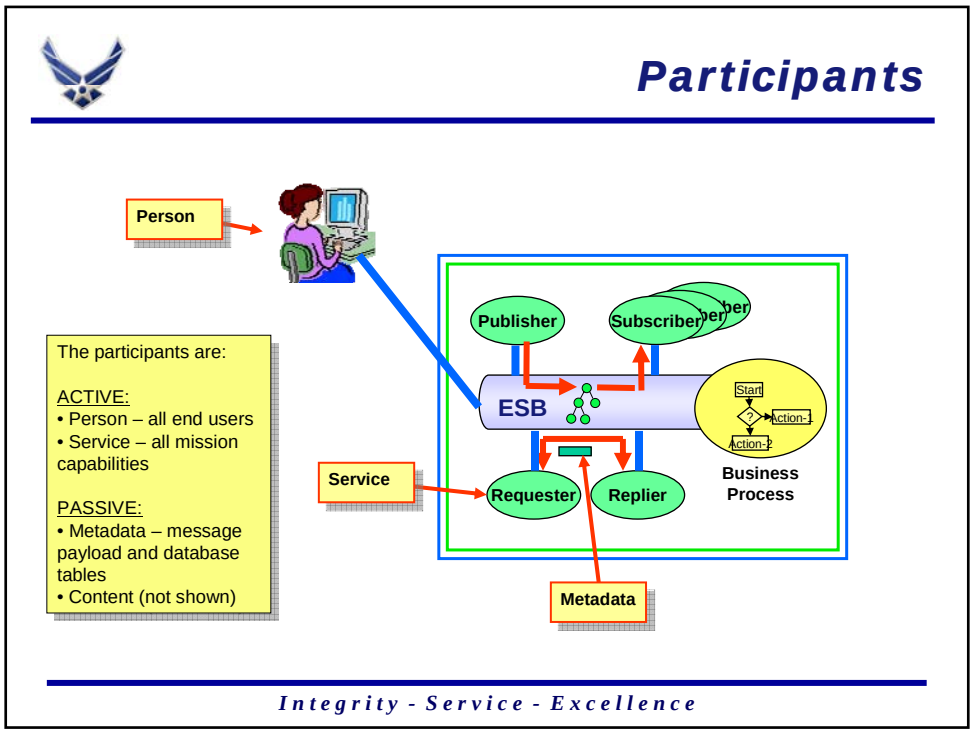
The net effect is that services that are integrated into the ESB can, within limits of authorization, freely and asynchronously exchange messages. For the purposes of integrating legacy systems,

[23] This is compatible with the JTF MOSA (Modular Open Systems Approach): http://www.acq.osd.mil/ats/opensyst.htm

the services that are integrated into the ESB can include adapters and private implementations that reach back to existing legacy systems. This in effect puts a modern service-oriented face on legacy systems. The focus on exchanging messages is deliberate. Prior attempts at defining an architecture based on platform software, such as DII/COE doesn't guarantee that pieces of software can be integrated. However, integration via message exchange will always work, because the major messaging vendors have messaging bridges and adapters to each other.

3.4.1 Participants

At this point, let's consider the active participants within a node. We have the services, the systems that the services execute upon, and the end-users of the services. All of these participants and their roles are known to each other and share a common trust model within the node. In addition, there are passive participants within a node[24]. These passive participants within a node are the messages. These messages are produced by one service and consumed by another service and are expressed in a vocabulary managed by a COI. To facilitate sharing, messages should be marked with tracing codes to denote the service that created it and the metadata of the payload (Discovery URIs of the respective participants).



3-7 Participants

[24] Content as a passive participant is not discussed in this paper.

3.4.2 Scope of Influence

Active and passive participants within a node have to be managed. The primary management tool of participants is the set of discovery services: services, metadata, people, and content. The set of discovery services will reside in some nodes[25], but not all. A Node may want its own discovery service in order to keep track of which services it wants to make public, and which to keep private. Nodes without their own discovery services will use the discovery services in a hierarchically higher-level node, and is said to be within that higher-level discovery services' scope of influence. This will be covered more thoroughly in the discussion on federation below.

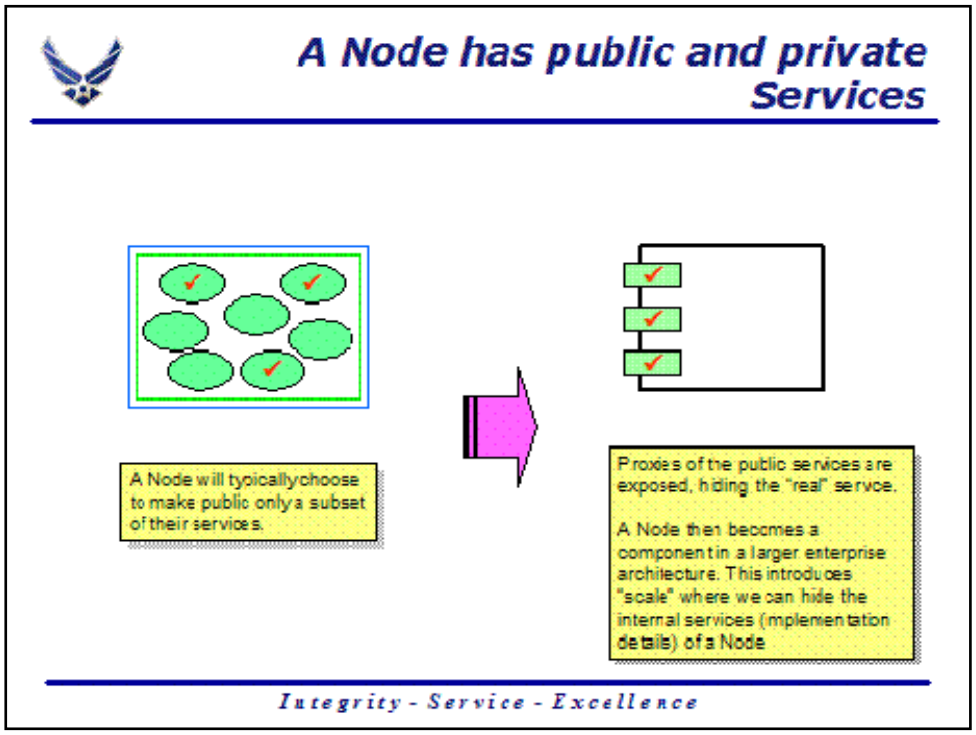
3.4.3 Opportunities

Having all of the active and passive participants within a node managed by discovery services presents unique opportunities:

- **Acquisition** – We have an opportunity to bridge the gap between stovepipe funding and the net-centric future vision. Currently, when a program is created and funding is allocated for a mission capability, a contract is created with an integrator with instructions to purchase hardware, software infrastructure, as well is to create the mission capability. With all participants managed by discovery services, the integrator can be instructed to reuse what already exists, both in infrastructure and other existing mission capability exposed as services. This has the potential to dramatically reduce costs, both in initial acquisition and maintenance.
- **Architecture** – We have an opportunity to use the information in the discovery services to, in effect, create a dynamic "Master Capabilities List" for all the Nodes in an enterprise. Please see the Appendix – A Modest Architecture Proposal.

The node then becomes a unit of scaling in the constructed enterprise architecture. If we "zoom out," we can see the boundary of the node and understand the aggregate mission capability without having to see all the individual services. This scaling, however, is not perfect since it is the individual services of one node that have to interoperate with individual services of another node. We will see more on that next.

[25] While major services in most nodes will be listed in high-level discovery services, such as the Air Force discovery, we do not want to exclude the possibility of nodes having their own discovery. This can be useful for example in the case of netted sensors. The netted sensors node may need to be able to dynamically find neighboring sensors as they come online and register them in a local discovery service.



3-8 Node with Public and Private Services

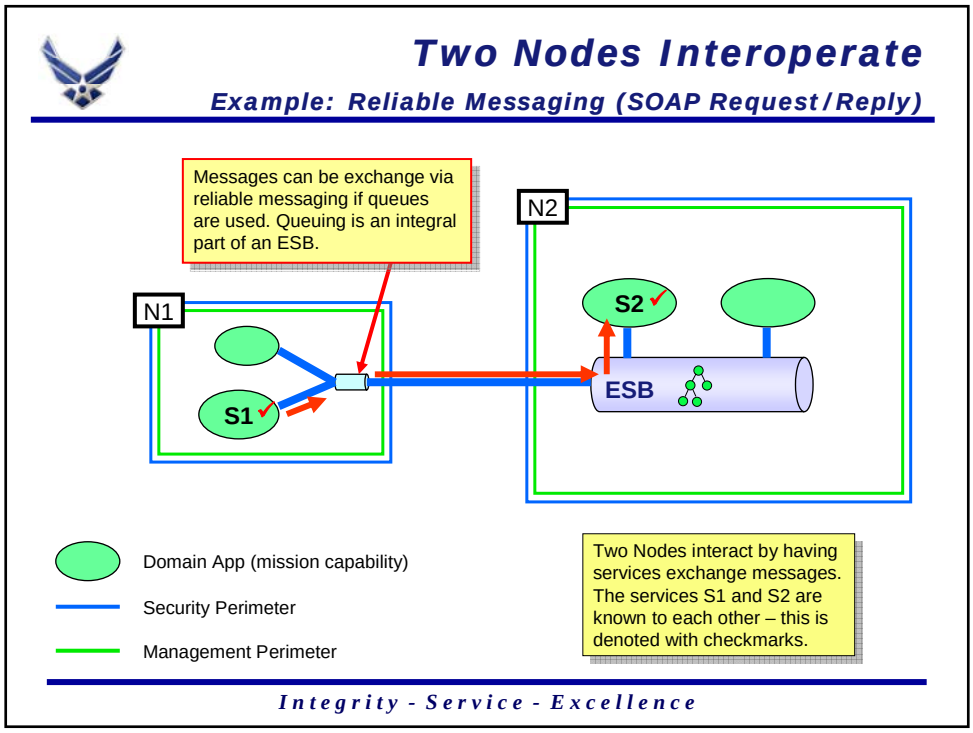
3.5 Nodes Interoperate

INTEROPERATION: Two Nodes that exchange messages interoperate.

In a net-centric conversation, a set of services exchange messages in order to complete a unit of work. The services each belong to one of the Nodes, and the Nodes have to cooperate and share enough information between them, so that the services can exchange messages. Thus, when two Nodes interoperate, they are sharing information with each other so their respective contained public services can exchange messages. The information that is shared is the “enabling context” for the message exchange. In addition, there is a dynamic process that keeps the context current.

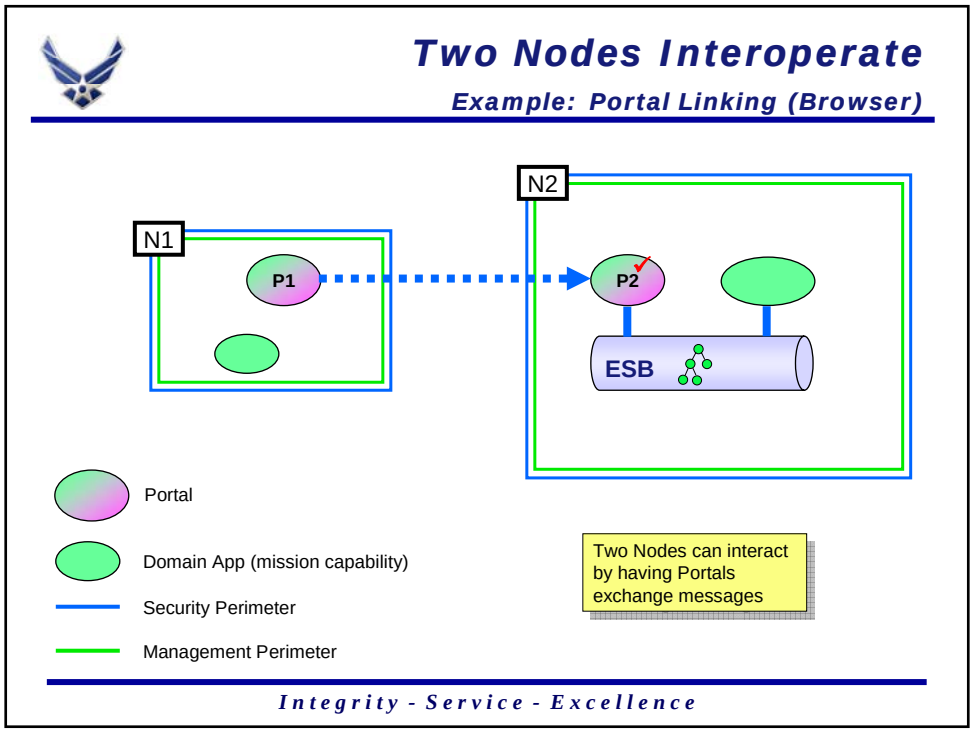
The cases examined below focus on the exchange of messages between web services (SOAP messages). JMS message exchanges are facilitated by the ESB, and Node to Node message exchange of JMS messages will be facilitated by ESB Extension Cords (below). The exchange of messages in a BPEL process is factored into individual SOAP message exchanges, thus the discussion below on SOAP also applies. For a summary, see the table – Summary of Methods of Integration / Intra and Inter Nodal in the Enterprise Messaging section below.

Since the exchange of SOAP messages between two nodes is always between two services, this means that at least one service in each respective node has to be made "public" public to the other node.



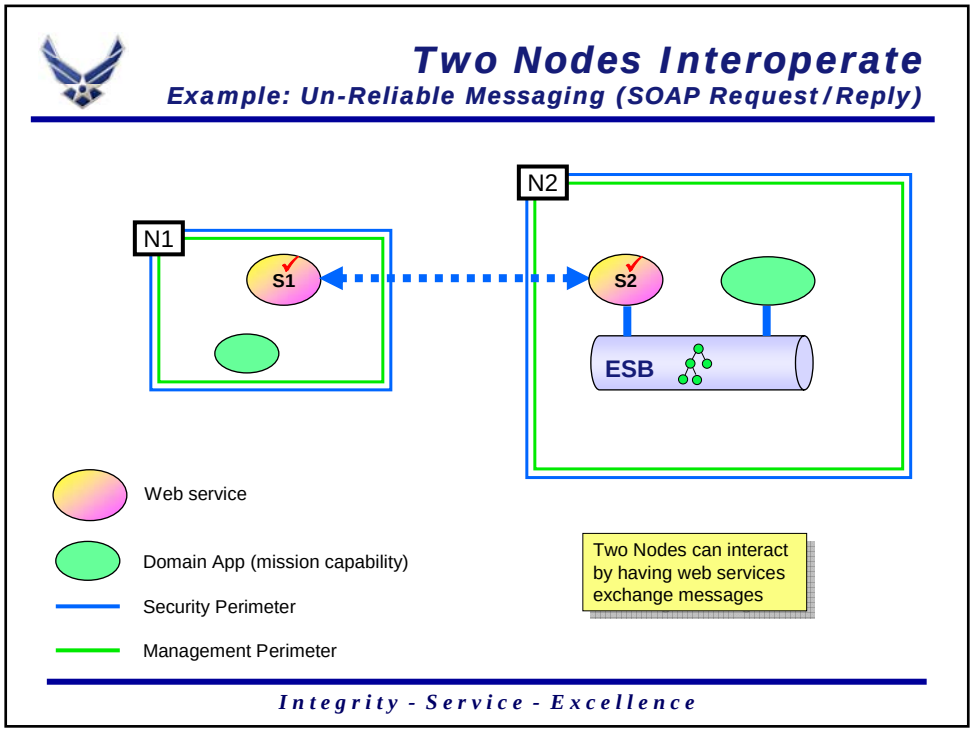
3-9 Interoperation between Nodes- Reliable Messaging (Soap Request/ Reply)

Other forms of interoperability are shown below. First, two portals in two different Nodes can interoperate by linking to each other. This is one of the ways that the Joint GCSS system will link to the GCSS-AF. In this case, the portal being called (P2) in Node N2 must be made known to the calling portal (P1) in Node N1. It is optional for the calling portal to be known by the called portal.



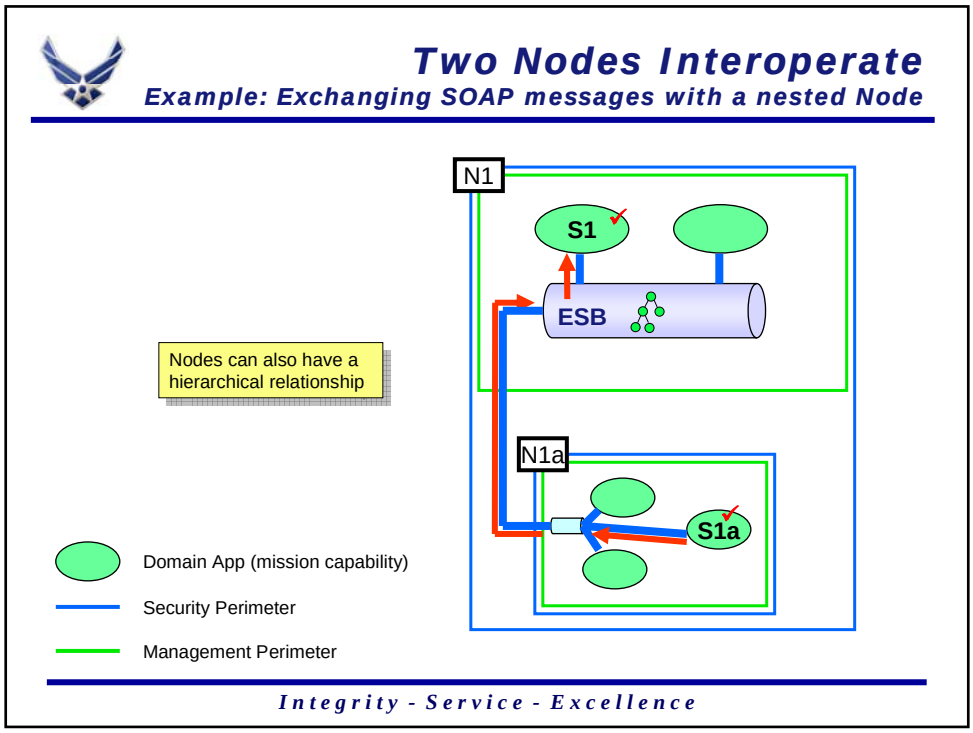
3-10 Interoperation between Nodes- Portal Linking (Browser)

Another method of interoperation is to exchange SOAP messages without reliable messaging. Service S1 in Node N1 and Service S2 in Node N2 must know about each other as in the previous SOAP example.



3-11 Operation between Nodes: Un-Reliable Messaging (SOAP Request/Reply)

Below, we see that Nodes can also interoperate hierarchically. Service S1a in Node N1a sends a message to Service S1 in Node N1.



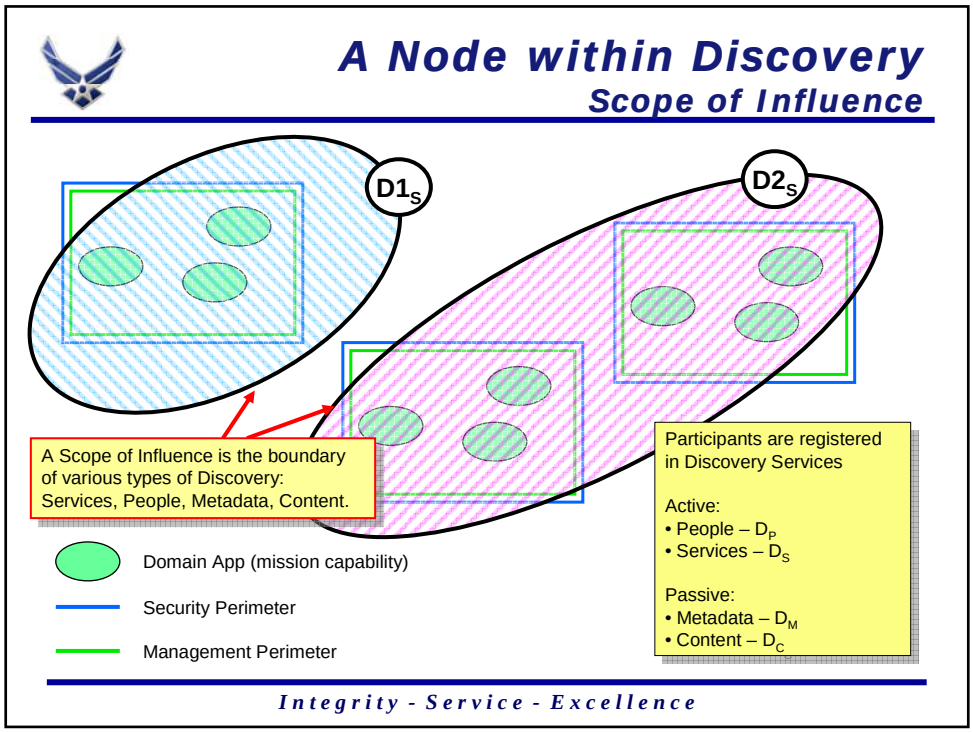
3-12 Interoperating Nodes- Exchanging Soap Messages (with a Nested node)

3.6 Nodes Interoperate using Federation

Two Nodes interoperate by exchanging messages. Exchanging messages requires that the participating services know each other and mutually share enabling and message context. This will usually mean sharing Discovery registry entries.

The organizing principle for participants is Discovery services.

There is one Discovery type for each participant type: Services, Metadata, People, and Content. Since interoperating Nodes is primarily a machine-to-machine effort, we will focus on the Services and Metadata types of Discovery.



3-13 Scope of Influence

Services within Nodes are listed in a particular Discovery service. The scope of influence of a Discovery service runs along organizational boundaries, so they will tend to encompass whole Nodes. For the same reasons we noted above that there can never be one of anything in an enterprise, there will be more than one instance of each Discovery type, such as a Services Discovery[26]. The diagram above is a notional depiction of Discoveries as stood up perhaps by Air Force and Army. They will each have their own set of Nodes, and keep track of their own active participants, such as Services, in a Services Discovery.

The smallest scope for a Discovery is one Node. Below we see a Discovery scope encompassing one Node.

Services within Nodes are listed in a particular Discovery service. The scope of influence of a Discovery service runs along organizational boundaries, so they will tend to encompass whole Nodes. For the same reasons we noted above that there can never be one of anything in an enterprise, there will be more than one instance of each Discovery type, such as a Services Discovery[27]. The diagram above is a notional depiction of Discoveries as stood up perhaps by Air

[26] For example, each of the branches Air Force, Army, Navy, Marines, as well as DoD level and Coalition partners will have their own Discovery. Also, for reasons we will see shortly, each Node may have their own private Discovery to keep track of private services that no other Node will see.

[27] For example, each of the branches Air Force, Army, Navy, Marines, as well as DoD level and Coalition partners will have their own Discovery. Also, for reasons we will see shortly, each Node may have their own private Discovery to keep track of private services that no other Node will see.

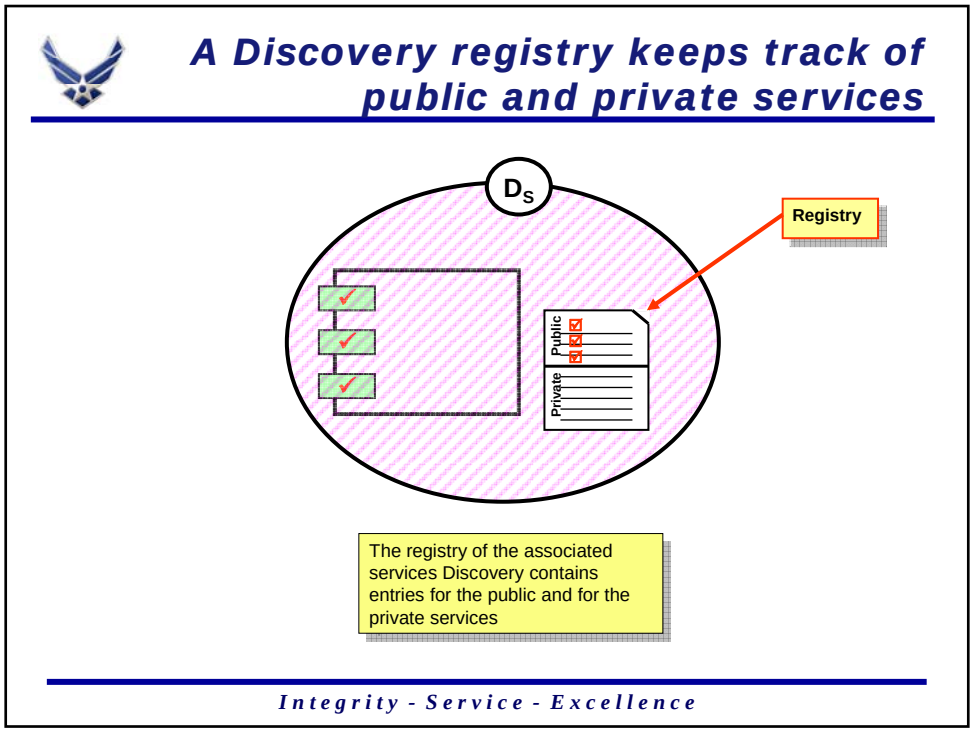
Force and Army. They will each have their own set of Nodes, and keep track of their own active participants, such as Services, in a Services Discovery.

The smallest scope for a Discovery is one Node. Below we see a Discovery scope encompassing one Node.

The scaling principle for a Node is the hiding of private services.

A Node will "expose" or make public only a subset of its services. The reason is similar to any encapsulation. Only a subset of the services will be useful outside of the Node. It is these public services that will participate in wider reaching net-centric conversations. The Discovery service for that Node will maintain a list of which services are public, and which are private.

These public and private lists of services within a Node must be shared amongst Nodes to enable a net-centric conversation.



3-14 Discovery Registry Tracking Public and Private Services

We see that a Node can be a component in a larger enterprise, hiding its implementation (private services) with an API for net-centric conversations (public services). Further, the public services should be exposed via proxy, hiding the physical location of the "real" service.

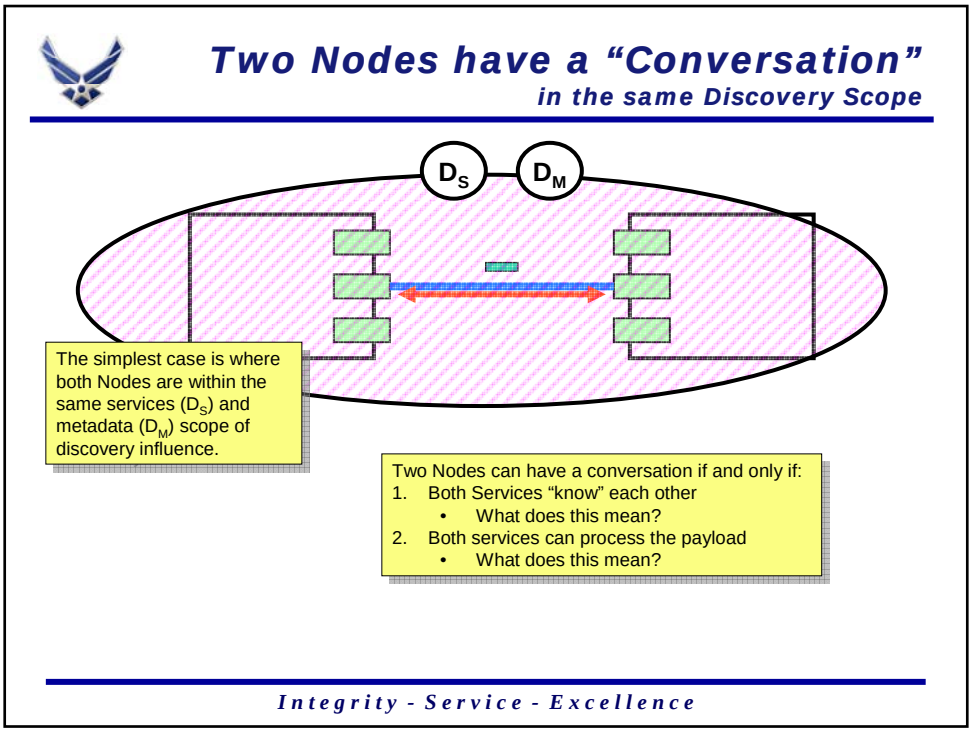
The primary organizing principle between any two nodes is federation.

Two Nodes work together by having their public services participate in a net-centric conversation. This requires sharing of enabling context as detailed in the table above. This sharing of enabling context is called federation. Federation means sharing enough information between nodes about each other's public services to effectively exchange messages.

Federation is the sharing of public service discovery information. If the two nodes are within the same discovery's sphere of influence, then the sharing has already happened. If the two nodes are in two different discovery's sphere of influences then we need a process to make the sharing happen.

In the diagram below, we see two Nodes in the same sphere of influence for their Services Discovery and Metadata Discovery. In practice the Services Discovery and the Metadata Discovery needs to have matching spheres of influence.

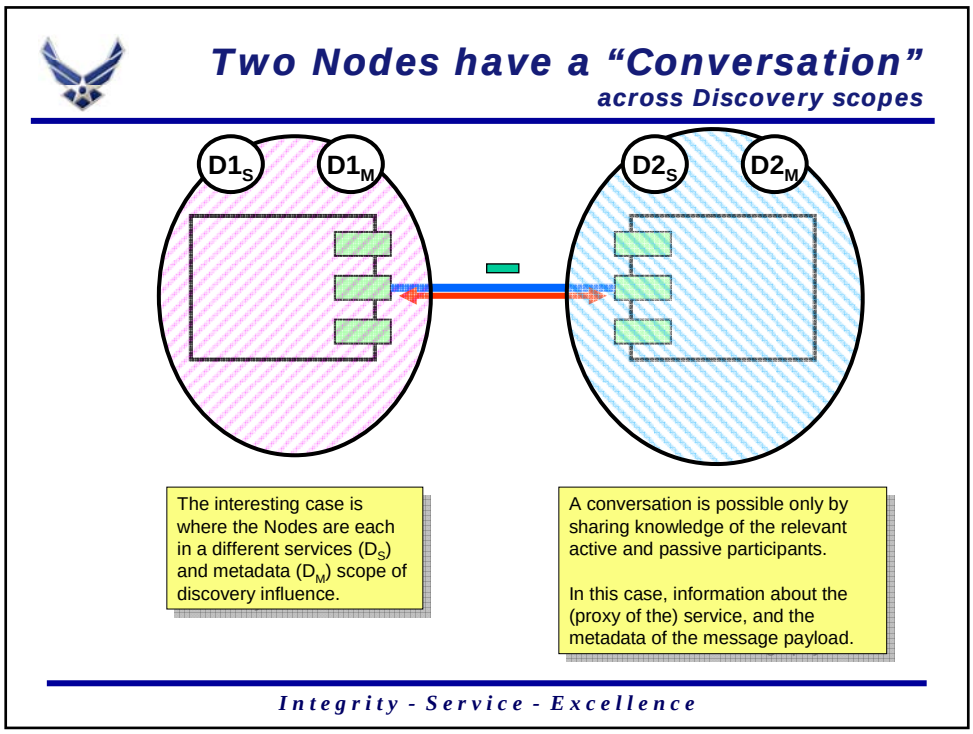
In contrast, the People Discovery service will likely be at a much higher level and encompass many Nodes. The reason for this is practical. Whereas for Nodes it reduces complexity to hide private services, for people who can be redeployed anywhere, it is simpler to have a few high level Discovery services for People[28]. Another practical factor is that with the advent of ERP systems, many legacy systems that expose services will likely be deprecated, and need to remain private, so they can evolve without disturbing net-centric conversations.



3-15 Conversation between Two Nodes (in the Same Discovery Scope)

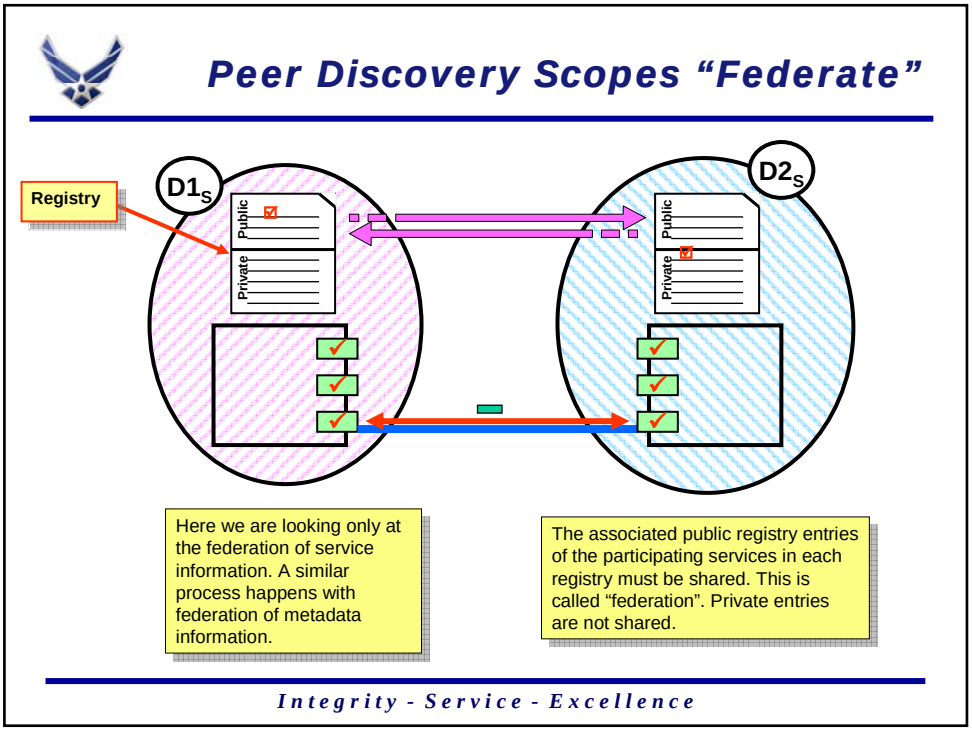
[28] For example, the author sees that the branches (Army, Navy, Marines, and Air Force) will have their People Discovery service, along with the DoD.

Next we see two Nodes that want to converse, each of which is in their own Service and Metadata scope of influence. Only information about their (proxies of) public services and message payloads need to be exchanged between the respective Discoveries.



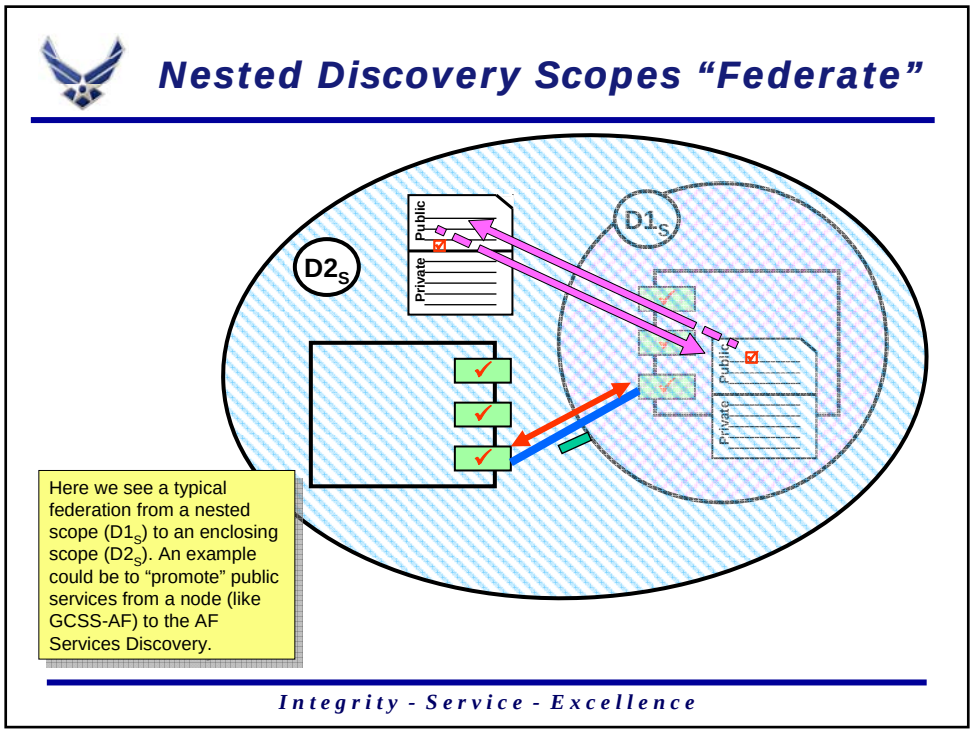
3-16 Conversation between Two Nodes, Across Discovery Scopes

Federation is the process of sharing public entries in the respective registries of the Discovery services. For service discovery this is sharing the proxy information (concrete WSDL endpoint) of the public services. The actual internal endpoint is private to the respective Node.



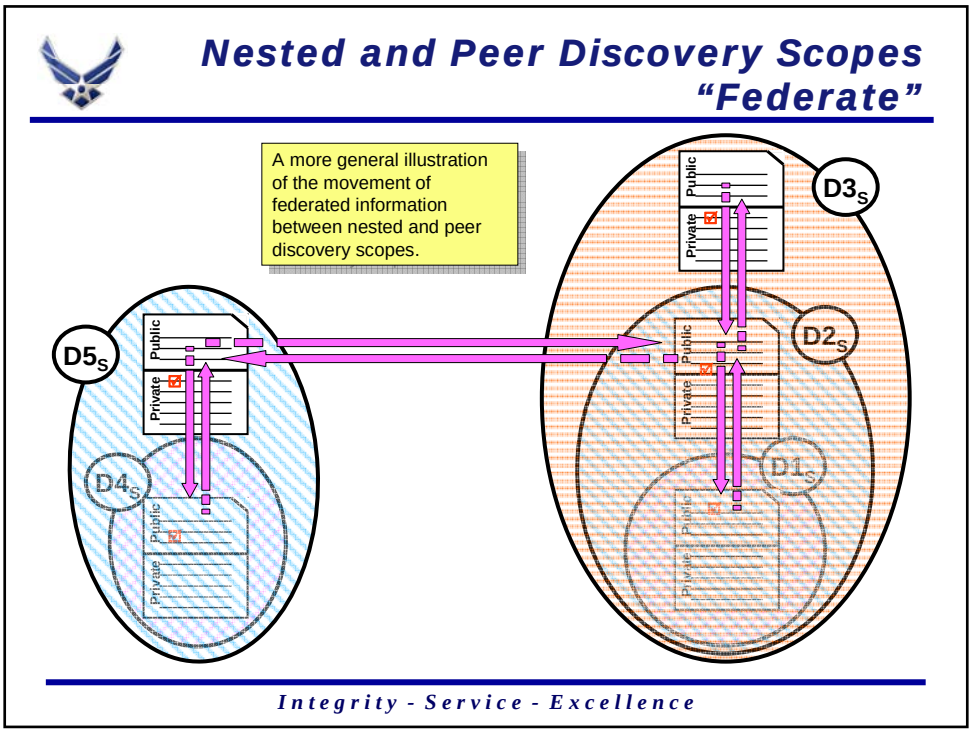
3-17 Peer Discover Scopes “Federate”

The next two diagrams illustrate more generalized use cases of Discovery federation. The first is a simple nested Node federating along hierarchical lines.



3-18 Nested Discovery Scopes"Federate"

The next diagram illustrates both hierarchical and peer-to-peer federation.



3-19 Nested and Peer Discovery Scopes "Federate"

With the interoperability and integration principles described above, we can now examine what an enterprise is, and what are enterprise services.

4 The Enterprise

ENTERPRISE: A forest of Nodes, possibly with no "top" or "root" Node.

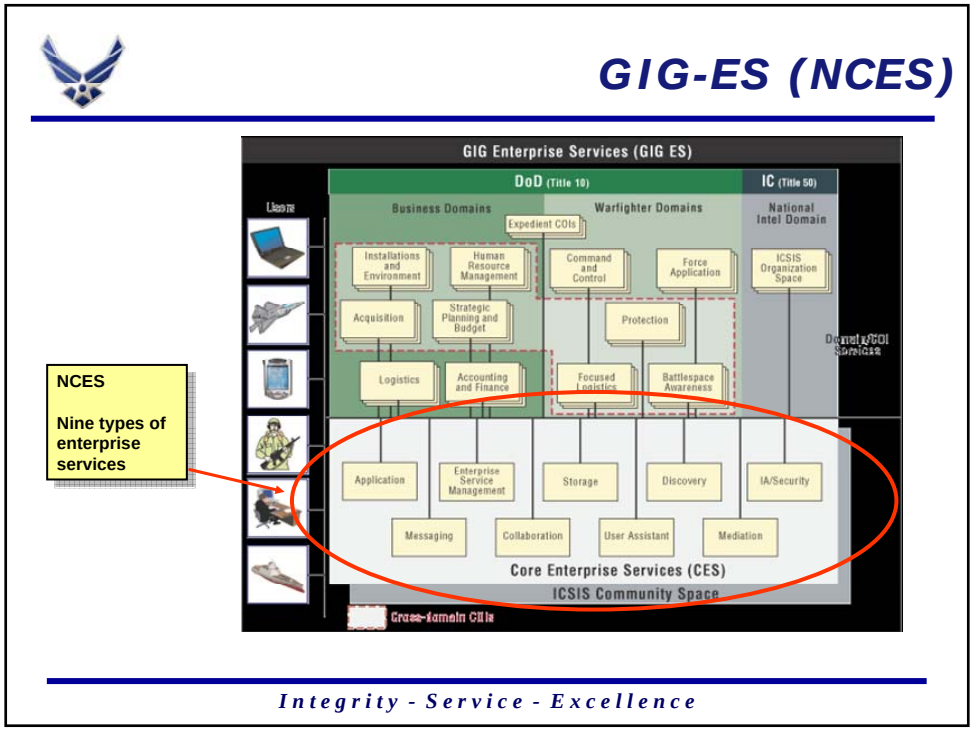
In a dynamic environment like the DOD, the enterprise will be constantly reorganizing, with mergers, acquisitions[29], and coalitions forming and dissolving. The need to form new and change existing net-centric conversations smoothly and rapidly is paramount. A major organizing principle is to group services within Nodes, and organize Nodes peer-to-peer and hierarchically. The net-centric conversations are enabled by federating the respective Discoveries, which was discussed above.

ENTERPRISE SERVICES: Those services which enable net-centric conversations across Nodes in the enterprise[30].

NCES and AFEITS efforts have described enterprise service types by outlining general characteristics. NCES calls for nine types as we see below:

- Application
- Enterprise Service Management
- Storage
- Discovery
- IA-Security
- Messaging
- Collaboration
- User Assistant
- Mediation

[29] In the DoD, mergers and acquisitions happen frequently under the label "reorganization".
[30] There is a class of enterprise services which enable human participation across Nodes, such as Collaboration. These are not discussed in this paper.



4-1 GI4-2G-ES (NCES)

Of the nine types, Discovery, Messaging and IA-Security are the minimal set of enterprise services that enable net-centric conversations:

- Discovery – Required to share enabling context for exchanging messages
- Messaging – Required to provide *reliable* messaging service
- IA-Security – Required for authentication and authorization of participants

The other service types add value to net-centric conversations, but are not required for minimal functionality.

4.1 Enterprise Discovery Service

We have discussed Discovery above in the "Principles" section. This service shares enabling context so that services of different Nodes can exchange messages with each other. The enabling context consists of knowledge of message payload and the identities and locations of the respective services.

KEY POINT: All Discovery services will be hosted in Nodes.

The Discovery service is for a set of Nodes, but will run in only one of those Nodes. The set of Nodes could be just one Node

4.2 Enterprise Messaging Service

Enterprise messaging is an enterprise service, because it enables *reliable* net-centric conversations between Nodes. The basic GIG provides an IP network with which we can use HTTP/S to send SOAP messages between public web services in Nodes. However, the HTTP/S protocol is not reliable[31], and we need guaranteed delivery[32] for exchanging messages over the WAN. The reasons are obvious. Without guaranteed delivery, each service will have to check for message delivery which in effect means added overhead, and the almost certain risk that this overhead will not be implemented uniformly. The approach we encourage and that commercial practice follows is to leave message delivery up to the middleware.

Reliable messaging between two points requires queues[33]. For example, in the WS RM, there are "message handlers" that the applications (services) on either end can use. In order to get persistence, these handlers must have queues with persistent backing store (such as a hard drive).

There are two competing standards for reliable messaging are WS Reliability and OASIS WS Reliable Messaging. There are fundamental differences between the two in terms of scope of QoS in message sequences, and whether the contract is considered a local matter. Co-authors of the WS-Reliability spec gave the OASIS WS RM group a requirements gap analysis in June 2005[34]. This author concludes there is much work to be done to arrive at consensus. Presently, reliable messaging is provided by proprietary products such as IBM MQ, TIBCO RV, etc.

In either case, proprietary or standards-based, two Nodes that exchange reliable messages must each have a queue that is compatible with the other. Until the reliable messaging standards are complete and robust[35], we are forced to use proprietary messaging.

Independent of proprietary or standards-based messaging, there are three options for enterprise messaging, discussed below. These options are complimentary, not mutually-exclusive, and the net-centric enterprise will make use of all three.

KEY POINT: All messaging services run inside of a Node. There is no such thing as messaging services between Nodes that run outside of a Node. All messaging services require a security and management perimeter, and are thus in a Node.

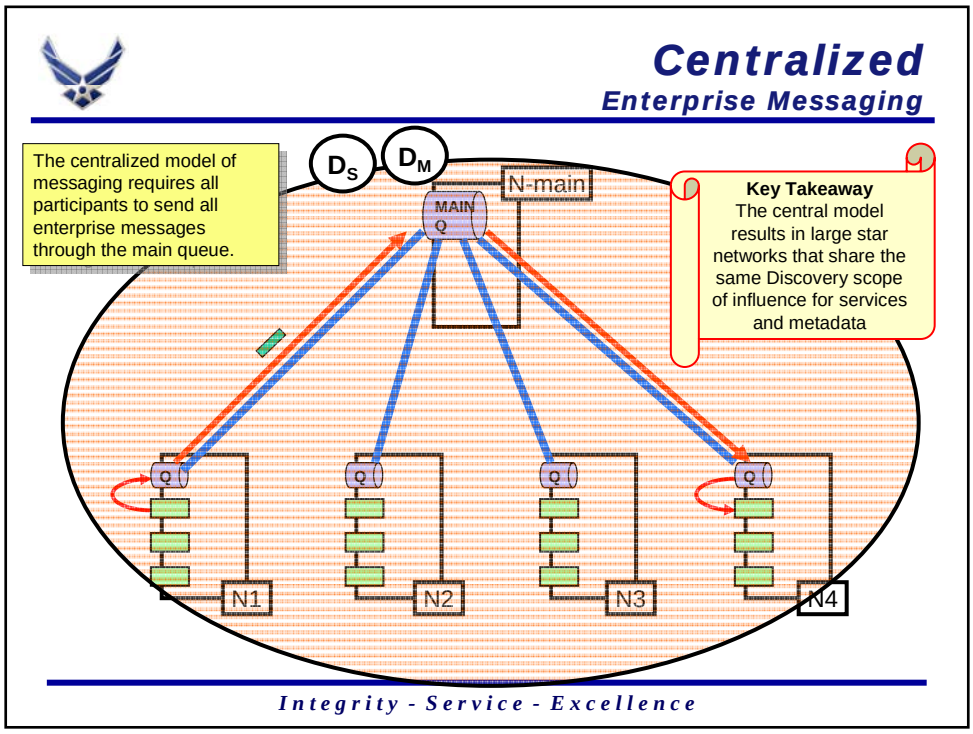
---

[31] HTTP/S does not guarantee delivery of messages.
[32] Some vendors (like IBM) call this "assured delivery". The distinction reminds us that if a Node is destroyed in an attack at the moment a message is delivered to it, the message may be lost. There are mitigation techniques for this (fail-over to a hot spare remote site) that are beyond the scope of this paper. For simplicity sake, I will continue to use the term "guaranteed delivery".
[33] Queues are similar to stacks of mail as it makes it way through the Postal system. At every Post Office, the receipt of your mail is recorded automatically, and the next Post Office destination computed. The receipt of your mail is the transactional "guarantee" that your mail isn't lost.
[34] See http://www.oasis-open.org/apps/org/workgroup/wsrm/download.php/13186/Requirements-Analysis-WS-RM_WS-Reliability-v08.pdf; this requires an OASIS account.
[35] The timeline for this is not clear.

4.2.1 Centralized

In this model, a set of Nodes install and use a queue owned and operated by a central Node as shown in the diagram below. The main or central queue must run inside of a Node.

The key point is that all of the participating Nodes send their messages through a central Node. This model requires that all the Nodes share the same services and metadata Discovery scope of influence.



4-3 Enterprise Messaging – Centralized

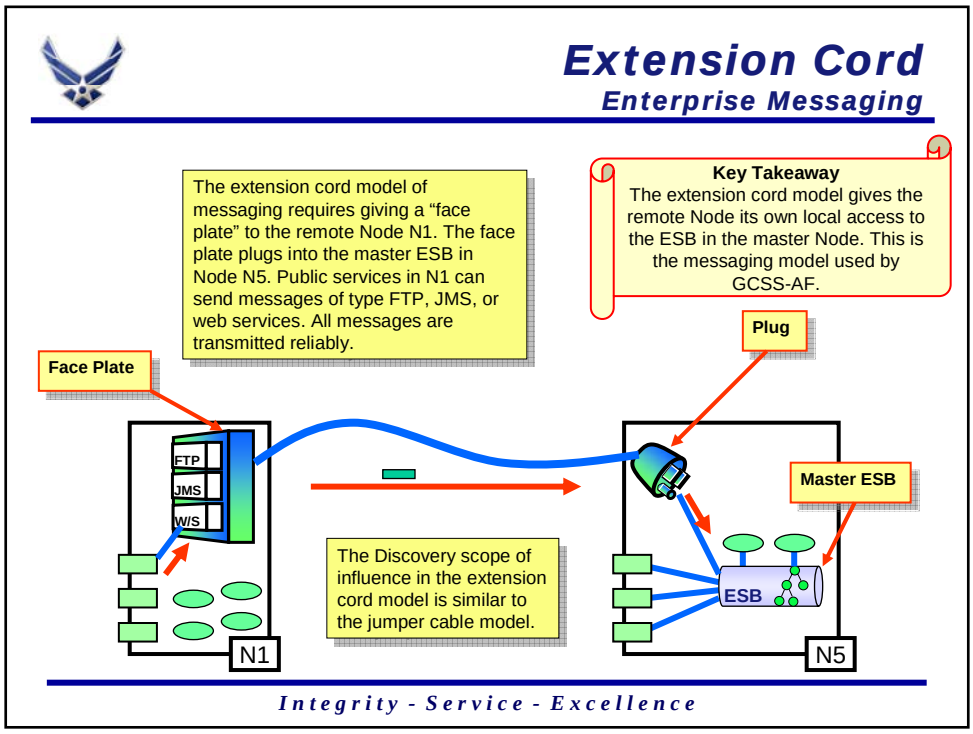
4.2.2 Jumper Cables

In this model, Nodes that exchange messages must have compatible queues, and be in the same Discovery scope of influence for the relevant public services. For example, in the diagram below, we see that a public service of Node N1 is sending a message to public service S1 of Node N4. Node N1 and service S1 of Node N4 must be in the same Discovery scope of influence for both services and metadata. Similarly public service S2 of Node N4 sends a message to Node N7, and they are both in the same Discovery scope of influence for both services and metadata.

The jumper cable between Node N1 and Node N4 can be owned and operated by either Node, or by a central agency. In the case of a central agency, the jumper cables would be of the same type and thus compatible. If the jumper cable was owned by individual Nodes, then at juncture points, such as Node N4, where a potentially two different jumper cables meet, standard bridges must connect them[36].

[36] All proprietary messaging products offer bridges to each other. For example BEA to IBM, Sonic to TIBCO, etc. This is standard equipment, and does not detract from the efficacy of this model.

In addition to the structural relationships from a messaging point of view, the jumper cable model also shows how scopes of metadata Discovery (COI regions) can be bridged. One Node can have public services, of which some are in one scope, some are in another. This discretionary sharing of public services would be part of that Node's federation processes. Inside the bridge Node, the internal services understand both sets of metadata (COIs).



4-4 Enterprise Messaging- Jumper Cable

4.2.3 Extension Cords

In this model, Nodes that exchange messages are given compatible queues by a master Node, and are in the same Discovery scope of influence for the relevant public services. This model is most suitable for master Nodes that use an Enterprise Service Bus (ESB) internally to integrate services[37].

This is the messaging model used by GCSS-AF. When NCES makes its enterprise messaging products available for use, then GCSS-AF will formulate a replacement policy.

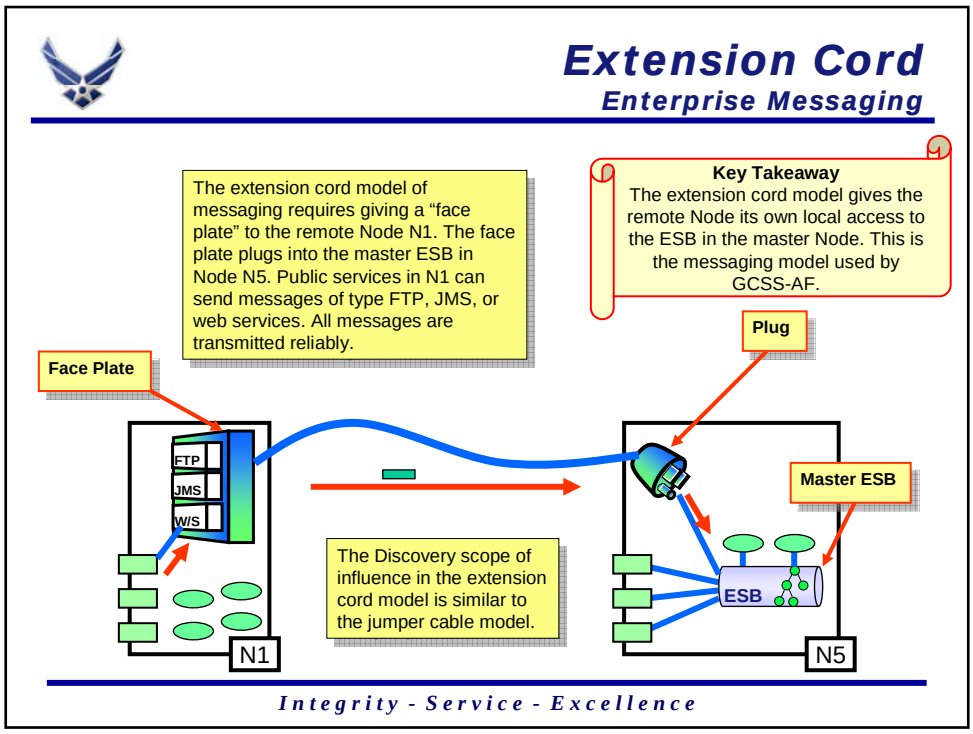
The extension cord model gives the remote node local access to the master ESB via a "face plate" with a variety of supported protocols, such as FTP (file drop), JMS (publish/subscribe) and web services (SOAP/WSDL request/reply). All of the supported protocols take the input message, and map it to the reliable messaging protocol used by the extension cord. At the plug end at the master ESB, the message is extracted from the reliable messaging protocol, and put back into the original protocol.

[37] See Appendix – Foundations for a summary of Enterprise Service Bus (ESB) capabilities and architectural features.

This makes SOAP/WSDL web service calls reliable over a WAN, and requires the use of asynchronous messaging. The reason is that messages may take a while to transmit over the WAN, exceeding the timeouts of the HTTP/S protocol.

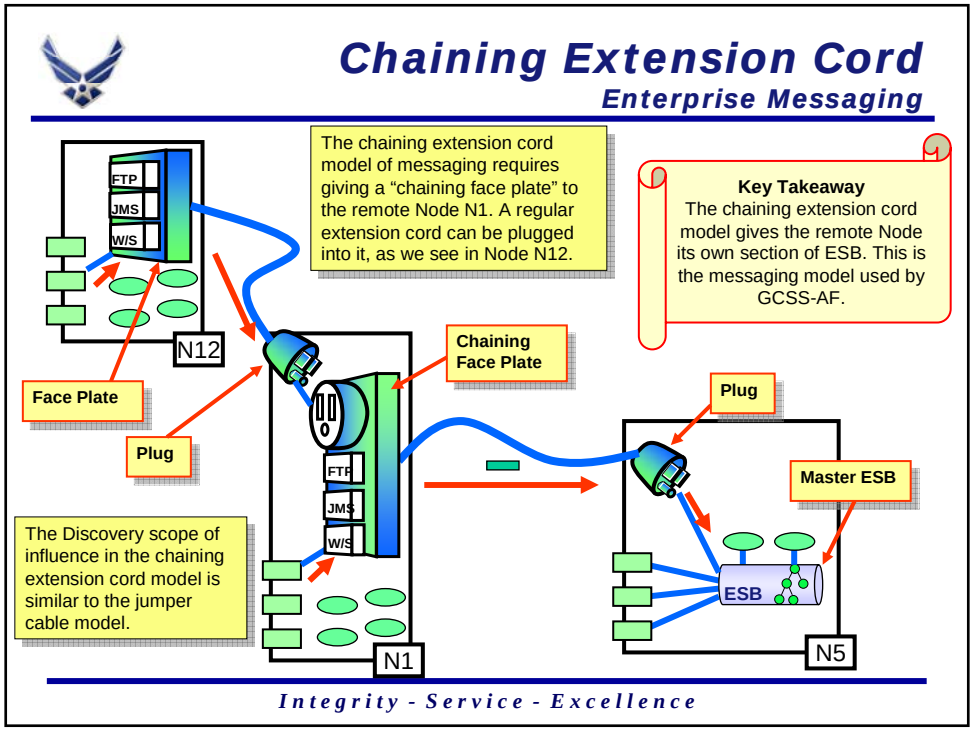
The Discovery scope of influence over this model is similar to the jumper cable model. Both caller and called services, and the message payload metadata must be federated between the respective Discovery scopes.

The limitation to this model is that the master ESB is in a similar relation to the remote Nodes as the Main Queue is to participating Queues in the centralized model.



4-5 Enterprise Messaging- Extension Cord

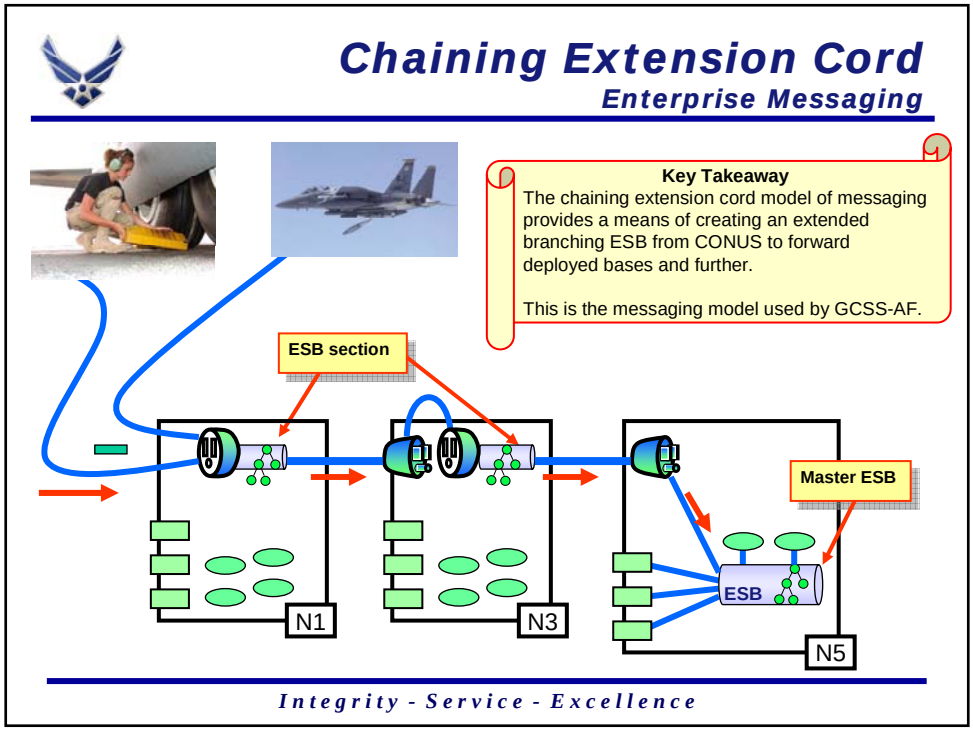
The next level of extension cord model is the chaining extension cord. This messaging model delegates a piece of the ESB, via a chaining face plate to the remote Node, including the ability to run an extension cord off of the chaining face plate.



4-6 Enterprise Messaging- Chaining Extension Cord

This in effect strings an extended ESB through a set of Nodes. This is the messaging model being developed by GCSS-AF. This gives us the ability to string an ESB through existing Nodes from CONUS to forward deployed and beyond. The services in each of the respective Nodes can plug in at their own pace.

The example below shows a straight chaining extension cords through a set of Nodes. The general topology is a root ESB with branches. The leaf nodes are the outermost reaches of the ESB and can be called "ESB edge". We can connect forward deployed Operations and Support activities in to the ESB edge as shown in the diagram below. This gives people at the ESB edge guaranteed messaging back to CONUS.



4-7 Enterprise Messaging- Chaining Extension Cord

The table below is a complete summary of various integration methods, inside and between Nodes.

Method of Integration	Intra-Nodal	Inter-Nodal
Publish/Subscribe	JMS over ESB inside Node	JMS over ESB extension cord
Request/Reply	SOAP/HTTP inside Node	SOAP over reliable message service (central, jumper cable, extension cord)
Orchestration	BPEL orchestrating SOAP/HTTP inside Node	BPEL orchestrating SOAP over reliable message service (central, jumper cable, extension cord)

4.3 Enterprise IA- Security Services

IA and security as an enterprise service is not covered in this paper. Currently GCSS-AF uses AF Active Directory to coordinate personnel identities and X.509 certificates to coordinate server and application identities. The security perimeter authenticates, and applications apply role-based authorization. There is no standardized set of roles.

We recognize the need for uniform security policy management and are monitoring the developments in the NCES program.

4.4 Ecosystem of Data

Data is central to operational capability, so now we consider how data is circulated through Nodes. Although we use data interoperability for messages exchanged in net-centric conversations, we do not fully examine the topic of data interoperability. For more on this topic please see a recent MITRE Technical Report by Blevins[38].

ECOSYSTEM OF DATA: Data is created and maintained by transactional systems that expose services, and these services live in Nodes.

At the foundation of all data circulated throughout the enterprise is the transactional system. These are the systems that maintains data via create, replace, and update functions.

Data is exposed via net-centric conversations, via message exchange. This message exchange can be a SOAP/WSDL query as depicted below, or JMS publish/subscribe. Both methods decouple the system being queried from the system doing the query. This allows both systems to evolve independently and maintain the net-centric conversation.

CARDINAL RULE: Do not access another system's database directly. Use net-centric conversations instead.

Accessing another database directly, via ODBC, JDBC, Discoverer or similar tools breaks net-centricity. The reason is simple. By using these direct database access tools, you need to know the exact schema of the target system. This makes your system dependent on the target structure at best, and prevents either of your systems from evolving independently. Moreover, as the first few Joint COIs are showing, the desired end state of net-centric conversation is different and evolved from any of the systems that must participate in them[39]. Thus, matching an existing system does not get you closer to being able to have net-centric conversations that the Joint Staff is demanding. In fact matching other systems databases can yield undesirable co-dependencies[40].

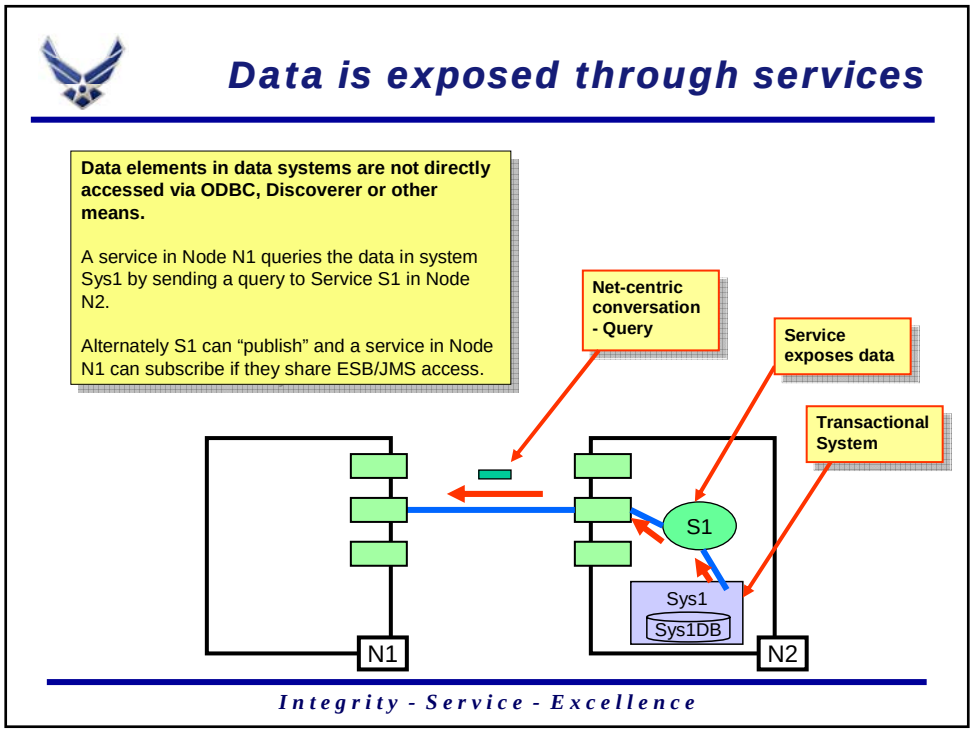
CARDINAL RULE: Access data via a net-centric conversation. Use either JMS publish/subscribe, SOAP/WSDL request/reply, or BPEL business process orchestration for complex interactions.

---

[38] See also "Integration-Interoperation_MP.doc" by Terrence Blevins, MITRE Technical Report, June 2005

[39] GFM (Global Force Management) is a good Joint COI example. Here the J8 is creating an XML schema to represent an "authorized force structure", with the Air Force, Army, Navy and Marines adding service specific features. This XML schema does not match any of the databases that currently maintain data that this structure represents.

[40] See JOPES and Air Force DCAPES



4-8 Data Exposed through Services

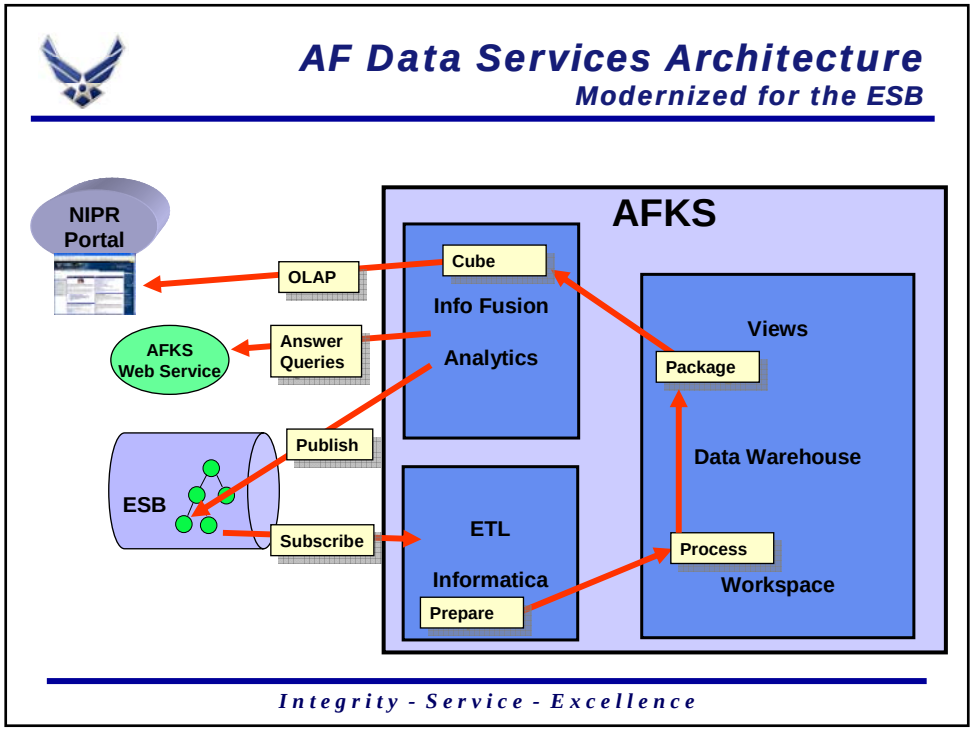
Data can be "rolled-up" by a data warehouse to get a cross functional view of the data. The roll-ups are an aggregation of data subscribed (or queried by) the data warehouse. Below is a view of how the Air Force Data Services of GCSS-AF acquires data.

Its primary means of acquiring data is via JMS publish/subscribe. This fits well with existing legacy systems that currently issue bulk file reports and publish via FTP. The GCSS-AF ESB can capture the FTP, and convert it to a JMS publish/subscribe where any system can subscribe to the report[41]. This also fits well with systems that are modernizing to publish XML change records. Both types of data are published to the JMS Topic, and the data warehouse (or other system) can subscribe to it. Of course this can be supplemented with request/reply SOAP queries.

Below we see the internal high level architecture of the AF Data Service data warehouse. The ETL (extract/transform/load) part of the warehouse subscribes to a JMS Topic. The data is moved into the warehouse and combined with other data as appropriate to create a cross-functional view.

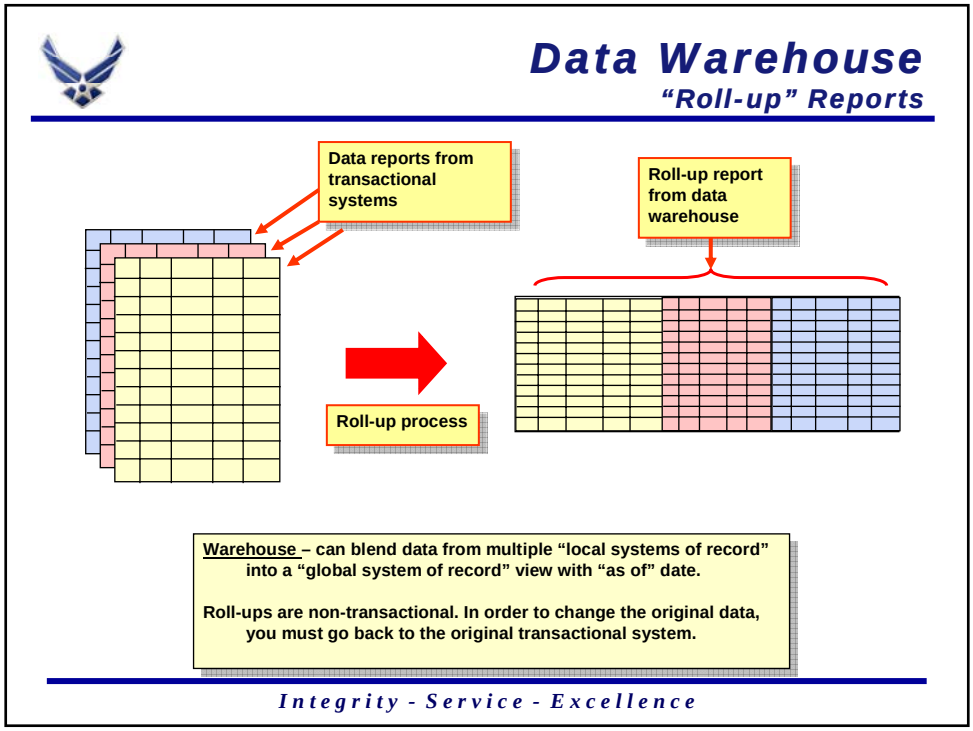
The data is then "cubed" in an analytic module. Once a view is constructed, it can be accessed via an OLAP application from the Portal, re-published to another JMS Topic for other systems to subscribe to, or made available via a web service that other services in other Nodes can query.

[41] Any system that is authorized by the publishing Portfolio Manager



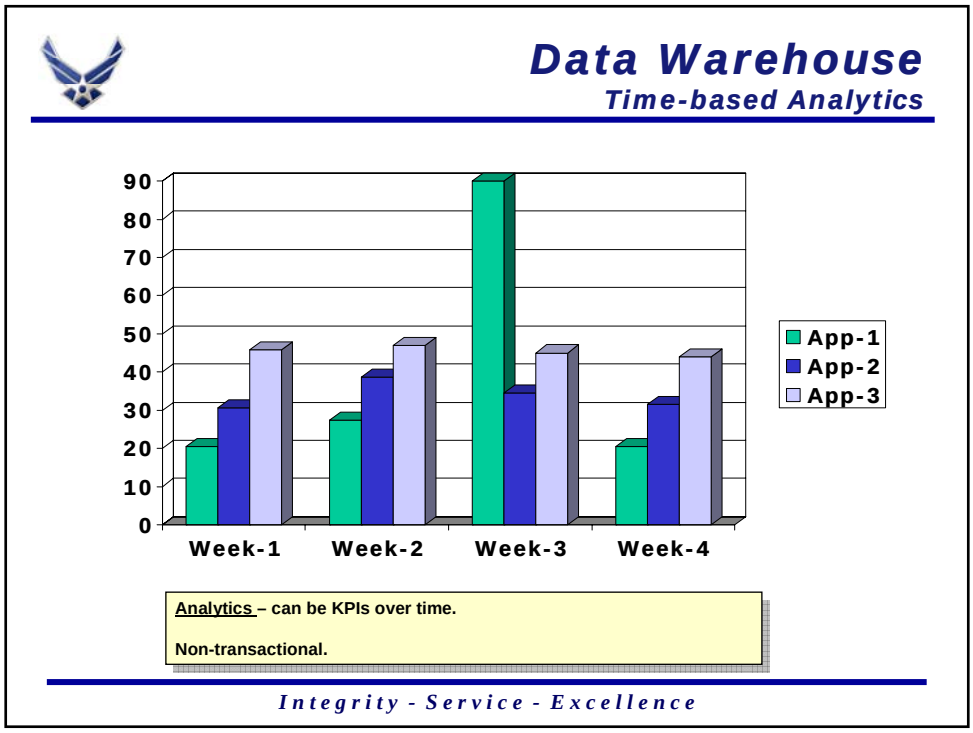
4-9 AF Data Services Architecture- Modernization for the ESB

The higher up in the organization you are, the greater the need for a cross functional view. Cross functional views are created by data warehouses that collect reports and transaction activity from the foundation transactional systems. Typically the cross functional reports have a wider view, and less detail than the original data.



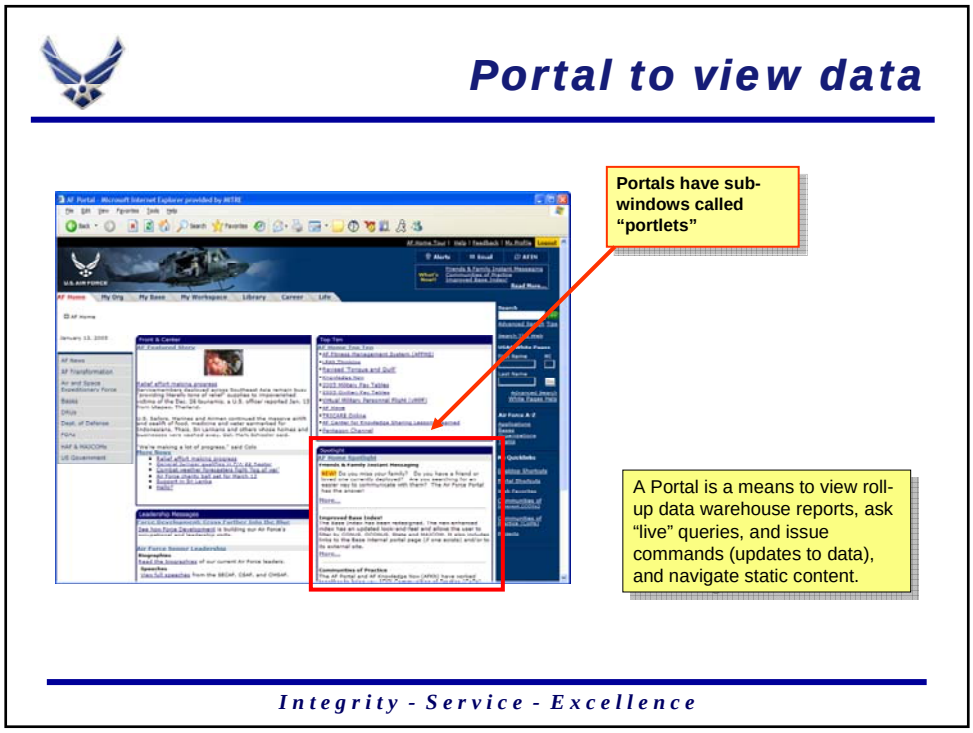
4-10 Data Warehouse

Data warehouses can also produce time-based and other dimensional analysis. This is useful to learn about and understand trends. This kind of cross-functional time-based analytics is not possible with a report from a single system.



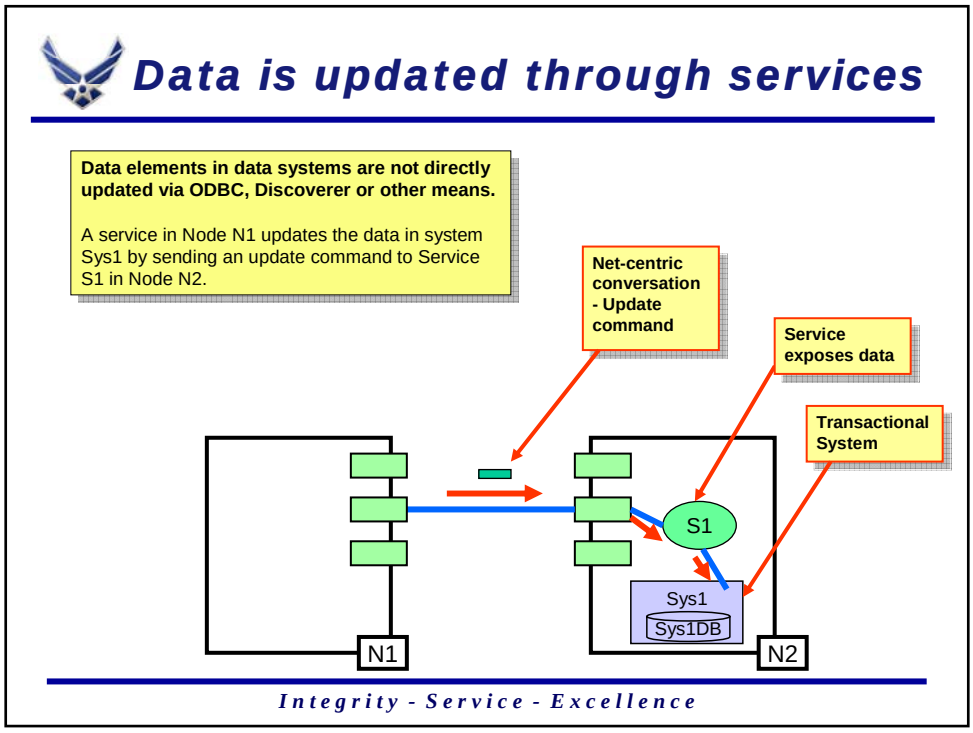
4-11 Data Warehouse- Time- Based Analytics

Data can be presented as the "official as-of-date-time truth" by combining officially sanctioned data warehouse roll-ups with "live queries" to the transactional systems. These pieces of data can be displayed via portlets in a Portal, accessed with a standard browser. This gives a Commander a broad and useful view over his/her area of command, with the ability to take action.



4-12 Portal for Viewing Data

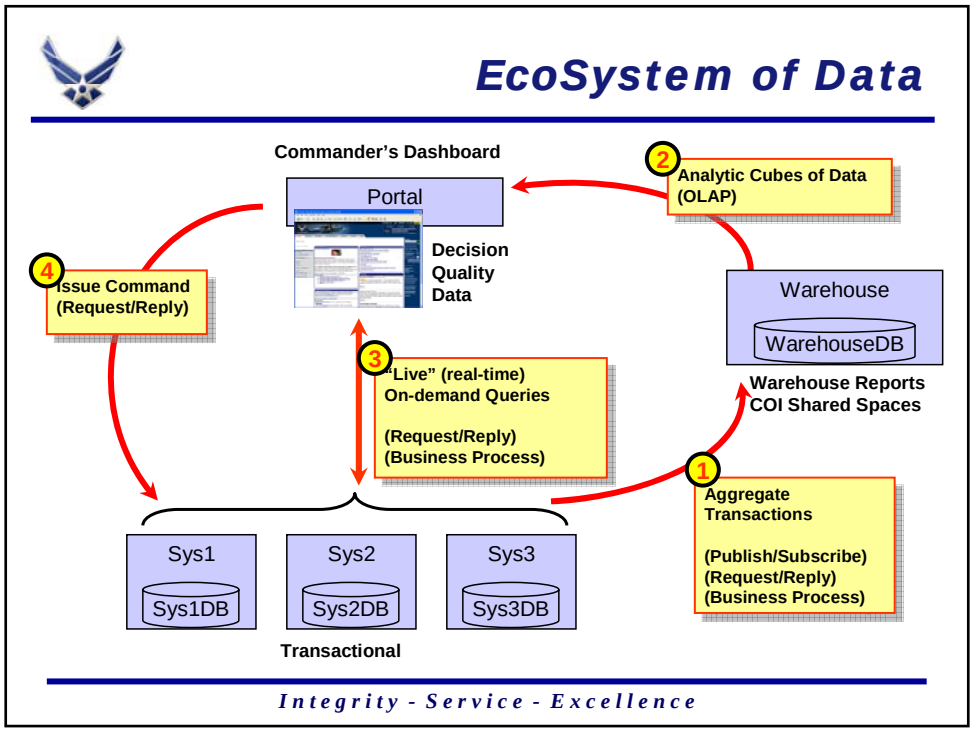
When a decision is made to create an "effect" a command can be issued to services to take action. Taking action will always mean updating data in some transactional system. As with queries, updates are always done in a net-centric conversation by sending a message to the service that wraps the target system.



4-13 Updating Data through Services

Putting all the elements together, we get a high-level picture of the ecosystem. In the diagram below, we see that data from transactional systems is aggregated into the warehouse. This creates a non-transactional "as-of" report that can be presented as a "cube" of data which can be navigated via an OLAP application in the Portal.

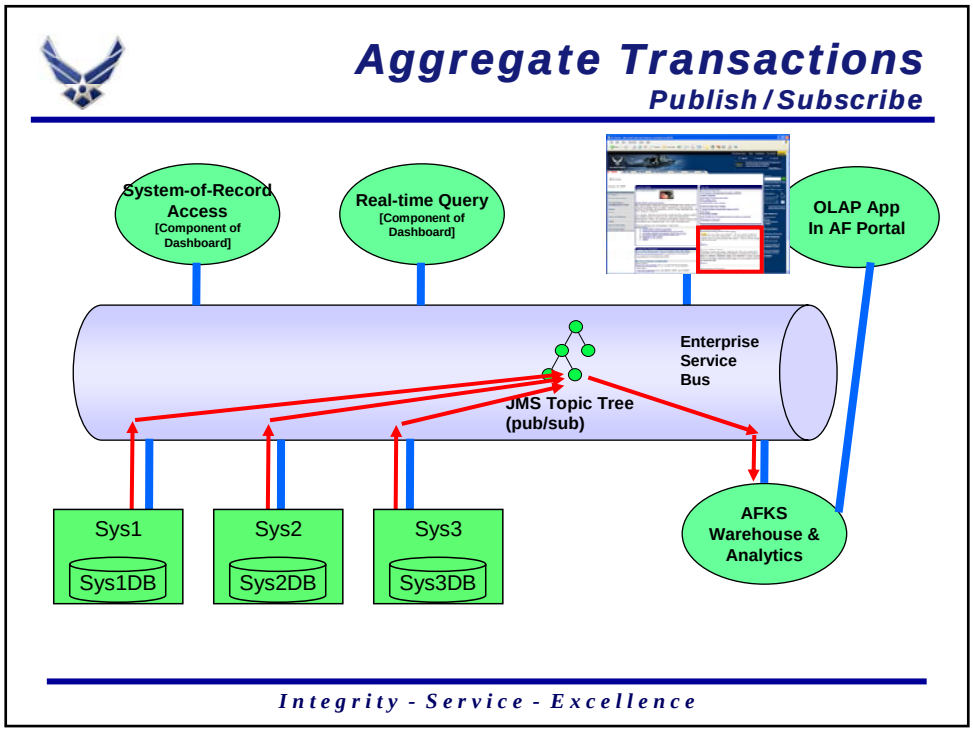
The warehouse data, along with the ability to do particular "live" queries give the Commander a Dashboard of decision quality data. When the Commander wants to issue a command, s/he interacts with the Portal which issues a command to the appropriate service(s).



4-14 Ecosystem of Data

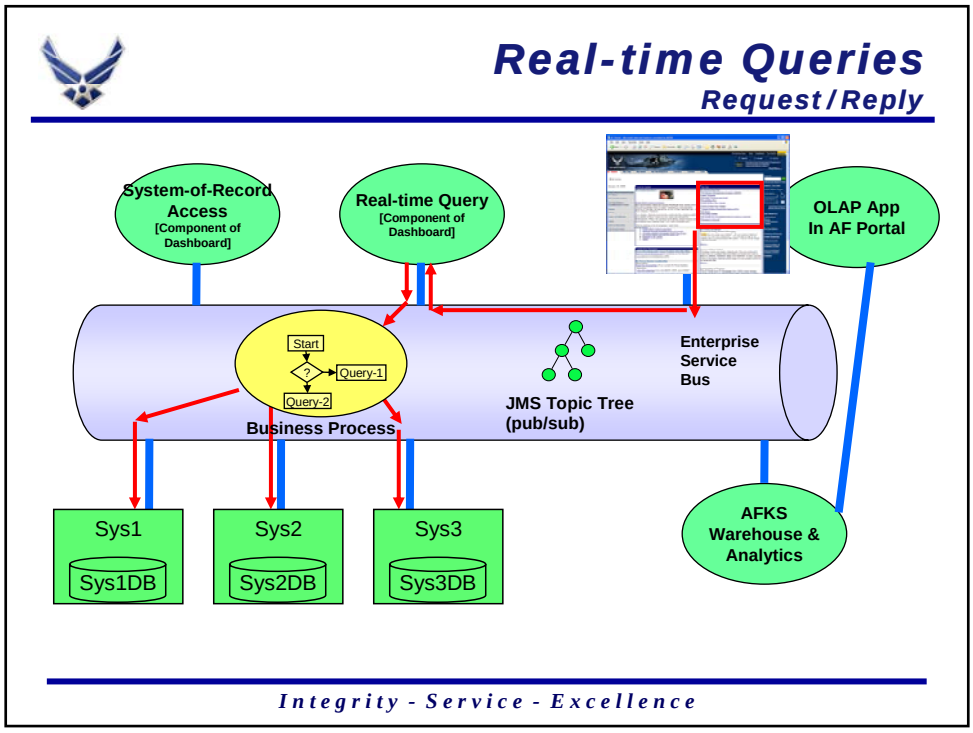
These are the basic elements that constitute our ecosystem of data. Now let's look at steps #1, #3, and #4. Step #2 is the use of an OLAP Business Intelligence tool which is beyond the scope of this paper.

Step #1 – Aggregate transactions – is where the data warehouse subscribes to transactional system data reports (large, extract reports) and change record activity (one change at a time). Typically each system, via its service will publish to its own unique JMS Topic.



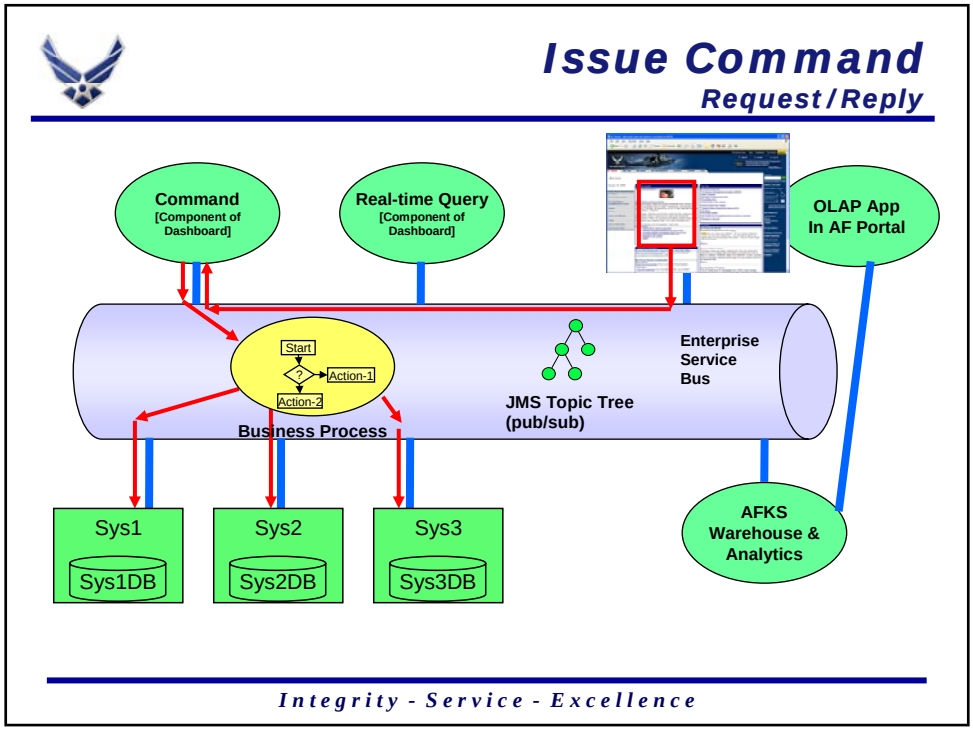
4-15 Aggregate Transactions- Publish/ Subscribe

Step #3 – Real-time (live) queries – can be initiated from inside a portlet that is the GUI for a real-time query service. This service, in the general case, will create a business process to orchestrate a net-centric conversation to query across the target services. The business process computes the answer, and passes it back to the service, which then displays the answer in the portlet.



4-16 Real Time Queries- Request/Reply

Step #4 – Issue Command – can be initiated from inside a portlet that is the GUI for a command service. This service, in the general case, will create a business process to orchestrate a net-centric conversation to update state (perform the command) across the target services. The business process determines if the command was successful, and passes the answer back to the service, which then displays it in the portlet.



4-17 Issue Command- Request/ Reply

4.4.1 Incumbents and Satellites

The set of transactional systems consist of incumbents and satellites. An incumbent is a large system that "owns" a large piece of the "market share" for that domain area. For example when the DIMHRS[42] system comes online, it will be an incumbent. The service unique (Air Force unique) components will be satellite to the incumbent. Both the incumbent and satellite will interact via net-centric conversations, which will enable further development of services.

The process for preparing for large ERP modules is as straightforward as it is daunting. The ERP process will be large, long-term and disruptive. A full treatment is beyond the scope of this paper. However, for current legacy systems that are thought to be future satellites, there are important steps that can be taken to prepare and make the transition as smooth as possible:

1. Connect the legacy system to an ESB via an adapter. In GCSS-AF the most common case is the FTP adapter.
2. Consolidate the number of interfaces on the legacy system. Many systems have redundant interfaces. The fewer unique interfaces you support, the easier it will be to adjust to the ERP system
3. Change your output from bulk file to XML change records – and strive to make the XML schema as close as possible to relevant Joint COIs as possible. The Joint COIs will be the heavy-weight drivers for net-centric conversations in the future.

[42] DoD Human Resource ERP system, with service unique components. When it comes fully online it is projected to fulfill 90% of the functionality of the AF Personnel systems.

4. Create services for all interactions with your legacy system. The ERP incumbents will only interact via services and net-centric conversations. These net-centric conversations should align with Joint COIs and other major functional COIs.
5. Plug your services into the business processes of the incumbent ERPs. All major ERP vendors (SAP, Oracle, PeopleSoft, SeeBeyond, etc.) provide templated business processes as their major value-add. These business processes are driven by BPEL engines, similar to the BPEL engine in the GCSS-AF ESB. There will be some interaction via publish/subscribe, but mostly via BPEL.

The major challenges for converting a legacy system to services that plug into BPEL processes are two-fold:

1. People – organizations need to start thinking about how they execute their function as business processes. This will take some time, and cannot be rushed. The best advice is to start simple by mapping all of your systems, and diagram the flow of bulk files from one system to another, and indicate all of your error processing, both automatic and manual. This will give you an idea of process flow from one system to another.
2. Technical – This can be just as challenging. There are two dimensions:
    a. What functionality to keep? – There won't be a clean cut line of systems and functionality to keep. Most likely bits and pieces from various systems will be crammed together into services. How do we track the movement of these requirements?
    b. How do I multi-task? – Many systems were designed for "one run" or one user at a time. A service by design handles as many as request service. This requires sessions, understanding critical sections of code, etc. In other words the work force needs to be retrained. Start early.

5 In Closing

We have reviewed the principles of interoperability and integration. The core principle is to support the rapid creation and update of net-centric conversations.

The agility of the enterprise can be defined as the highest sustainable rate of creation and update of net-centric conversations.

By keeping our focus on net-centric conversations we can create a simplified picture of enterprise architecture where services are grouped in Nodes, and Discovery scopes of influence organize how knowledge of (proxies of) public services and message payload metadata are shared.

The enterprise is the set of Nodes we are interested in. This is a forest of Nodes with no one "root" Node. This includes DoD, Coalition, Commercial Nodes, etc.

We reviewed how enterprise messaging services can be used to provide reliable messaging and ESB/JMS messaging pathways weaved through Nodes from CONUS to forward deployed positions.

The ecosystem of enterprise data includes transactional systems, data warehouses, decision data portals, real-time queries, and command oriented services.

Access and update data via services and net-centric conversations, <u>not</u> direct access (ODBC, etc.)

As ERP modules are introduced, they will become functional incumbents. This means that the surviving legacy systems must be satellite to the ERP modules, and provide "service unique" (AF unique) functionality. The way to prepare legacy systems is to upgrade them with services and plug them into the BPEL business processes that the ERP systems provide.

The balance of this paper is a set of appendices:

1. Architecture Proposal
    a. Crawl Discovery services to build a master capabilities list
2. Generic Data Center Proposal
    a. Create a few generic blueprints of data centers (Nodes)
3. Cross-Domain Integration
    a. Publish/subscribe cross-domain solution (in GCSS-AF Fall 2005)
    b. Web service cross-domain solution (FY06)
4. Foundations
    a. SOA
    b. ESB

Appendix A Architecture Proposal

The current methodology of creating DoDAF artifacts is suitable for highly customized, GOTS-centric, stove-piped, locally optimized systems.

This methodology applied to emerging net-centric systems leaves us in an awkward position, with several serious enterprise architecture concerns throughout the Air Force and DoD:

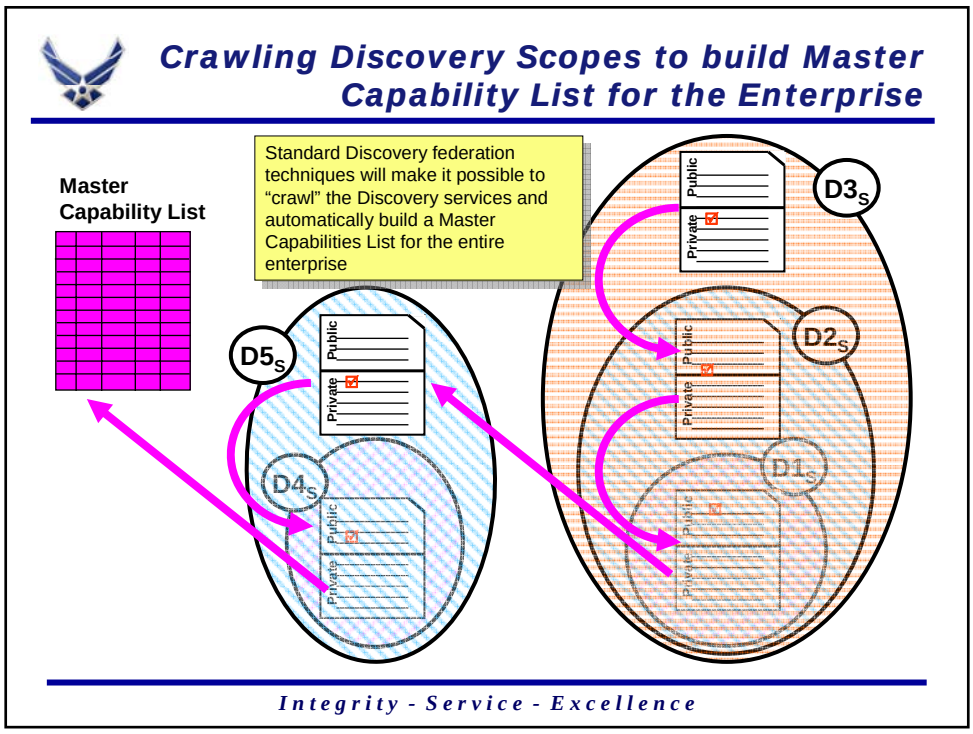
1. Enterprise architecture artifacts (DoDAF) are inadequate for net-centric architectures. As evidenced, there are numerous efforts trying to determine how to express SOA in DoDAF.
2. Enterprise architecture artifacts have an immense amount of detail and are rarely up-to-date. This is a reflection of the highly customized nature of our data centers (Nodes). If we migrate to generic data centers (see the Generic Data Center proposal below) we can eliminate a lot of architectural detail, and focus primarily on deployed services.
3. The Mission Capability List is useful, but how do we make sure this information is up-to-date?

We need to evolve to an enterprise architecture methodology that is suitable for COTS-centric, globally optimized, net-centric architectures.

Proposal:

1. Focus on creating a useful mission capabilities list. This should primarily be a list of public services of Nodes.
2. Ignore private services.
3. Ignore the details of the Node (move to Generic Data Center) infrastructure
4. Create the mission capabilities list by automatically *crawling* the nested and peer-to-peer Discovery services whose scope of influence covers the Nodes we are interested in.

In the diagram below we see a depiction of a crawling process. Starting with a top level Discovery, we can crawl recursively and depth-first to visit all the Discoveries in the path.



A-1 **Building of Master Capability List for the Enterprise**

VALUE: The master capabilities list expressed as public services is suitable for the entire AF or DoD to consider composition of mission threads, and rapid gap analysis.

A sample master capability list could look like the table below

Public Service	Method	Payload	Owning Node	Nested inside Node

ROADAHEAD: The *crawling* process required to create the mission capability list requires that the Discovery federation process be standardized.

This requires that we solve the following problems:

1. Consistent information (service and metadata profile) in all crawled repositories
2. Consistent federation process between Discoveries
3. Develop a crawler (not presently a commercial product)
4. Update and socialize the new net-centric Master Capability List (MCL)
5. Apply the new MCL to mission thread use case analysis and determine if further improvements need to be made.

Appendix B Generic Data Center Proposal

Today's Air Force information based systems are moving to COTS-centric, globally optimized net-centric architectures, and commercial best practices.

We have an opportunity to create a set of generic data center (Node) architecture patterns with the capability to run contemporary services (JMS, SOAP, BPEL) and interoperate with other Nodes.

These generic Node architectures can be available for a program in a two different ways:

1. The program can decide to build a new data center, determine the right-size model to build, and just copy the proven architecture
2. The program can decide what type of data center to use, and find the one that is currently operating that most closely meets its needs.

In both cases, we avoid hand-crafting yet another unique expensive data center.

The benefits of generic data centers (Nodes) are significant:

1. Acquisition can focus exclusively on adding capability (services) not more redundant, highly hand-crafted, locally optimized infrastructure
2. Integrators will respond to RFPs and RFQs more uniformly
3. If a program needs a Node, the cost of acquisition, maintenance and sunset is known in advance with low risk
4. The design, stand-up and operation of Nodes can be opened up to multiple integrators, increasing competition.

The design of the Generic Node(s) can be globally optimized, addressing one of the great paradoxes of the net-centric effort.

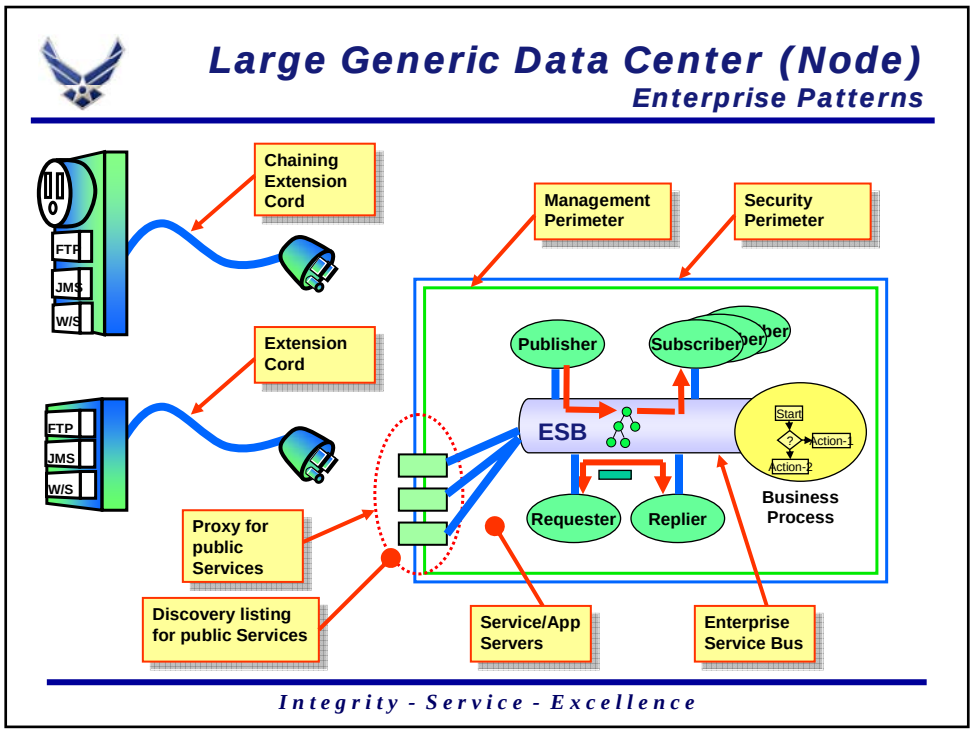
The great net-centric paradox is how do we optimize globally, when funding is stove-piped? The answer is that only the generic data centers (Nodes) have to be globally optimized. The rest are services which can be deployed on any Node. If there are only a few models of Node to select from, then the majority of program capability acquisition can be focused on functional services.

This is not a proposal to list products to purchase and install

This is a proposal to develop a few generic Node architectures, not to lock us into a fixed set of products to install. One of the advantages of this proposal is to have product choice where it makes sense, but to constrain the overall architecture to one that fits with a net-centric enterprise architecture as outlined in the "Principles" section earlier in this paper.

The large data center (Node) example is below. This will have the capacity to run:

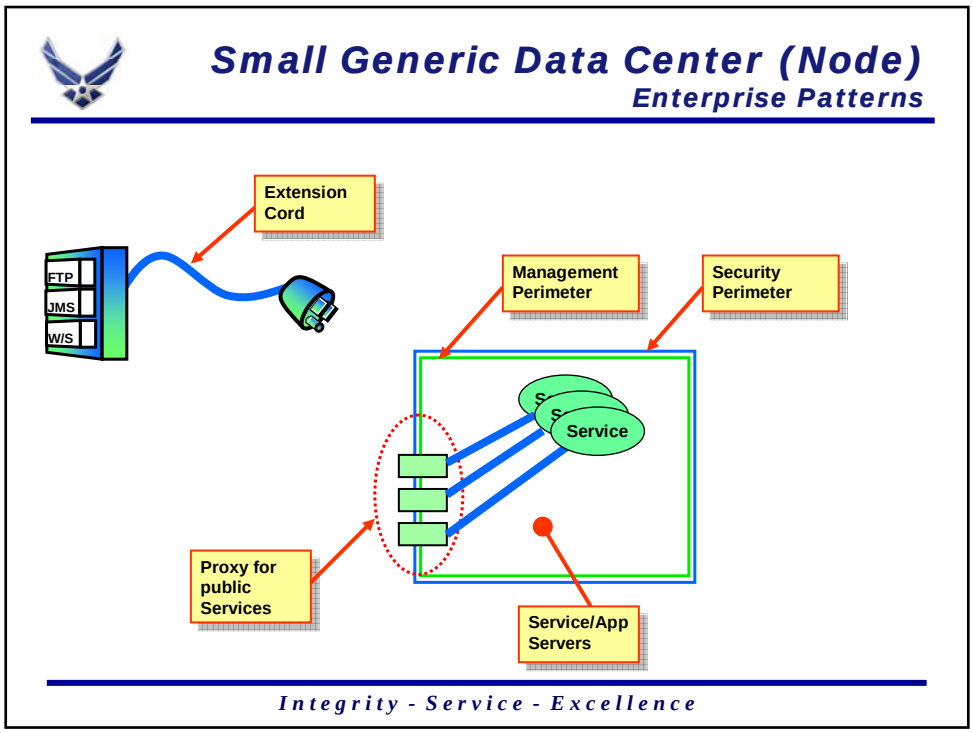
1. An ESB (including proxy for services)
2. Popular app servers (BEA, IBM, Oracle, .NET, etc)
3. Discovery
4. Security
5. Management
6. Extension cord
7. Chaining extension cord



B-1 Large Generic Data Center (Node)

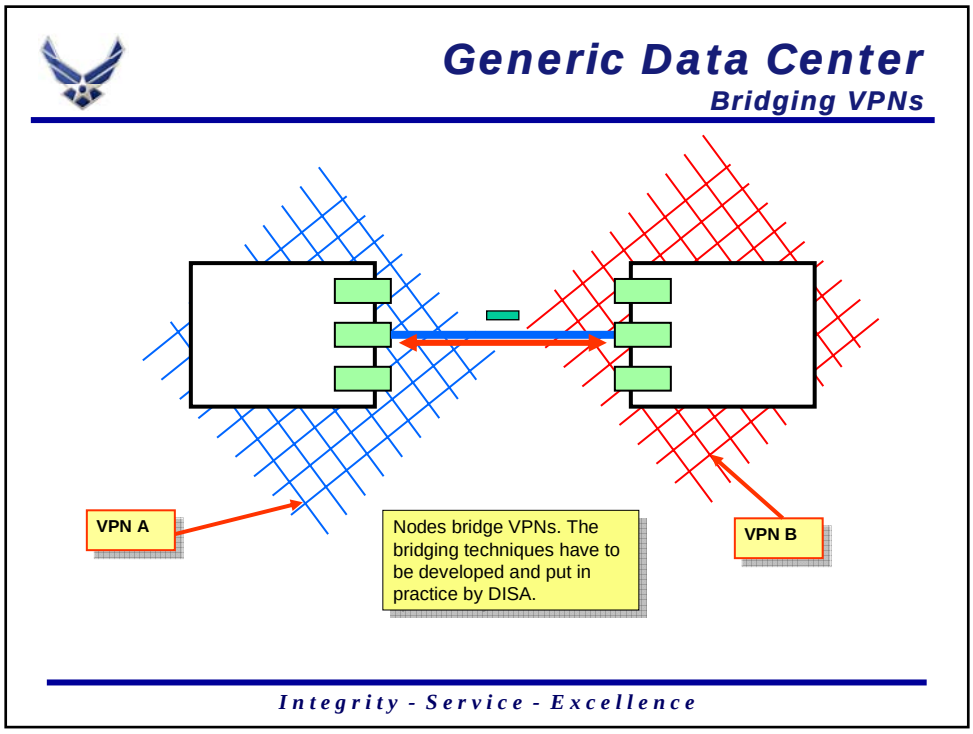
A small (minimal) Node has the capacity to run:

1. Popular app servers (BEA, IBM, Oracle, .NET, etc)
2. Security
3. Management
4. Proxy for public services
5. Connect to the face plate of an extension cord



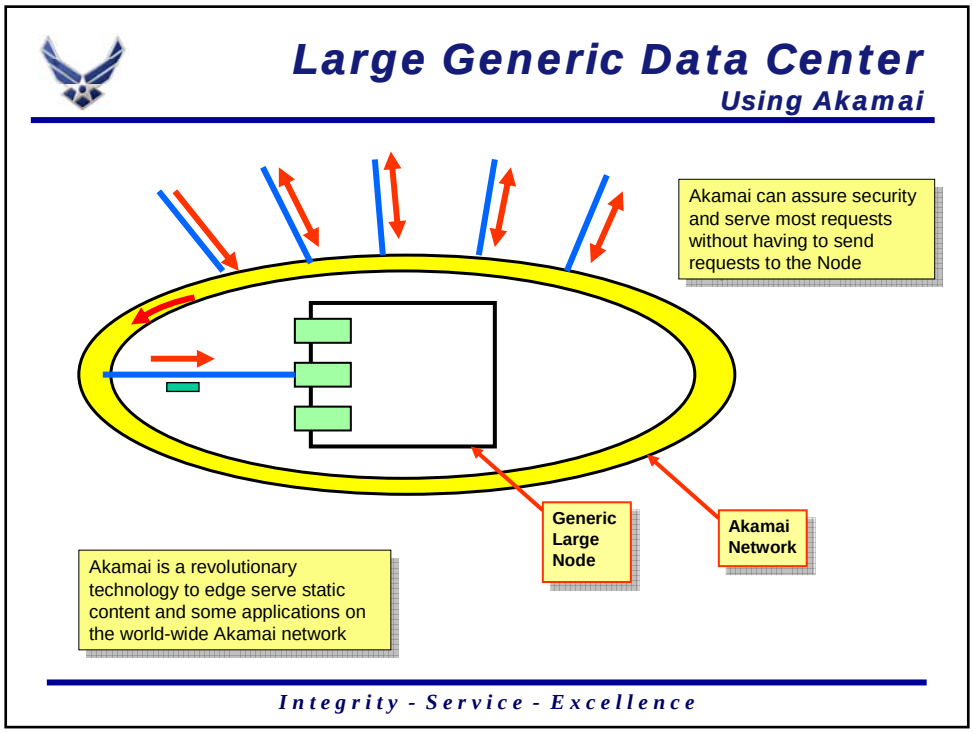
B-2 Small Generic Data Center (Node)

Another reason to focus efforts on a few Node architectures is to develop repeatable practices with respect to bridging VPNs and cross-domain integration. By creating a small set of Node architectures, and developing repeatable VPN bridging techniques, we can then make creating new net-centric conversations a reality. Currently it can take months to establish connections in DISA megacenters using B2B connections. The rate of connection creation is held back by policies that are not well understood.



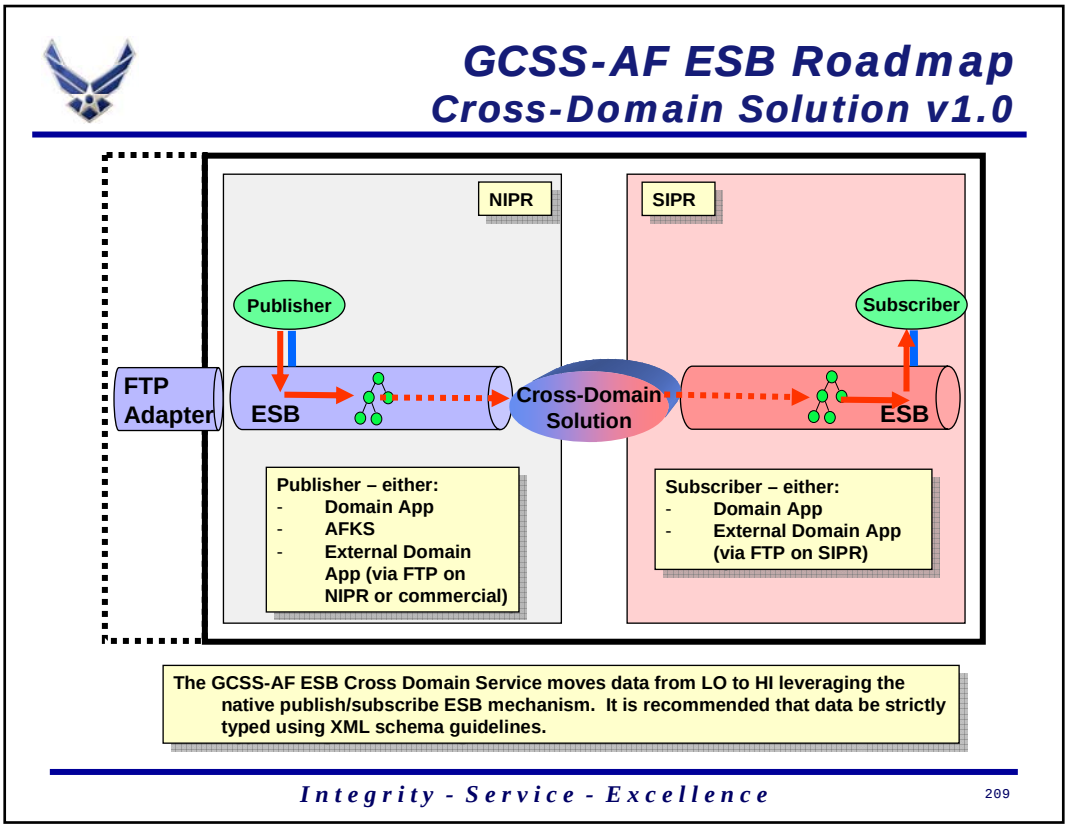
B-3 Generic Data Center- Bridging VPN's

Another capability to generalize is how large Nodes can use the Akamai network for edge serving. Akamai has the capability of serving static content and some applications. Many requests can be handled by Akamai without ever sending a request to the Node.



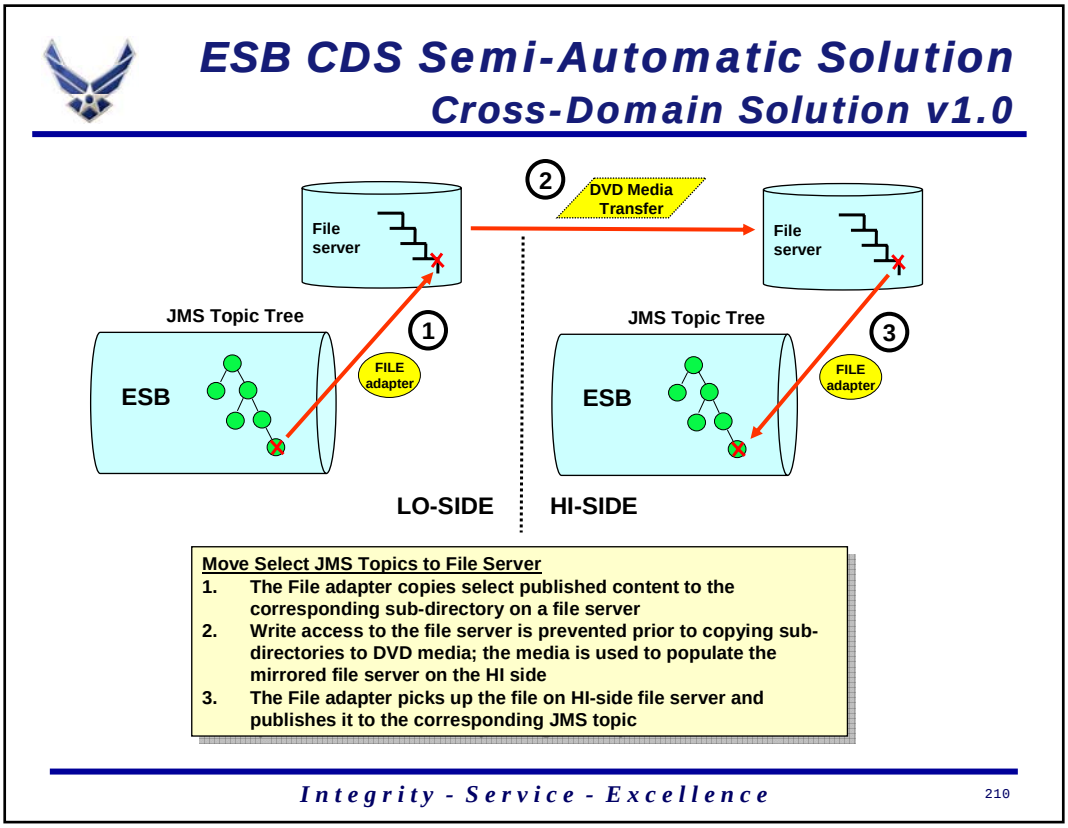
B-4 Large Generic Data Center- Using Akamai

Finally there is cross-domain capability. This is also a generic large Node capability. Below we review how the GCSS-AF data center is building this capability. The GCSS-AF Node is organized around the ESB. The v1.0 of the cross domain service (Fall 2005) will allow for subscribers on the SIPR side to get information published on the NIPR side.



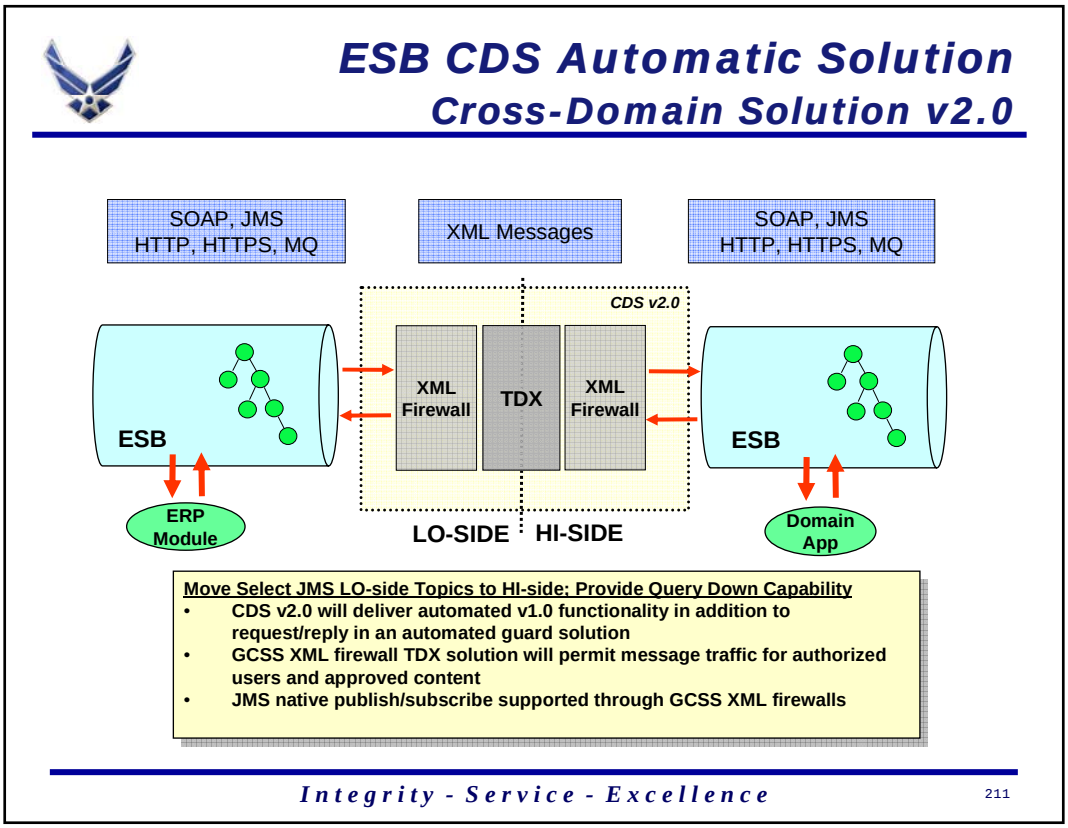
B-5 GCSS-AF ESB Roadmap- Cross Domain Solution v1.0

Below we can see the implementation leverages JMS publish/subscribe adapters already in place for the GCSS-AF ESB. The information published on the NIPR side will be imaged periodically onto a data DVD and republished on the SIPR side. There is virus scanning on both sides.



B-6 ESB CDS Semi-Automatic Solution – Cross Domain Solution v1.0

The next version of the cross domain solution v2.0 will be available in Fall 2006. It also supports JMS publish/subscribe, as well as web service calls. The difference is that instead of batching the data, each message will be sent as soon as its created. The core technology of the cross domain solution v2.0 is the TDX guard, with XML firewalls on either side.



B-7 ESB CDC Automatic Solution- Cross-Domain Solution v2.0

In summary, creating a small number of generic Node architectures will free up tremendous resources to focus on mission capability delivered as services.

Lastly, we recommend using GCSS-AF as a good example of a large Node generic architecture with over 500K users and over 100 applications. This data center has innovated generic Node architecture, delivered the first Enterprise Service Bus in the Air Force, the first large scale use of Akamai, and soon to be first generic cross-domain solution for JMS publish/subscribe from NIPR to SIPR.

Appendix C Foundations

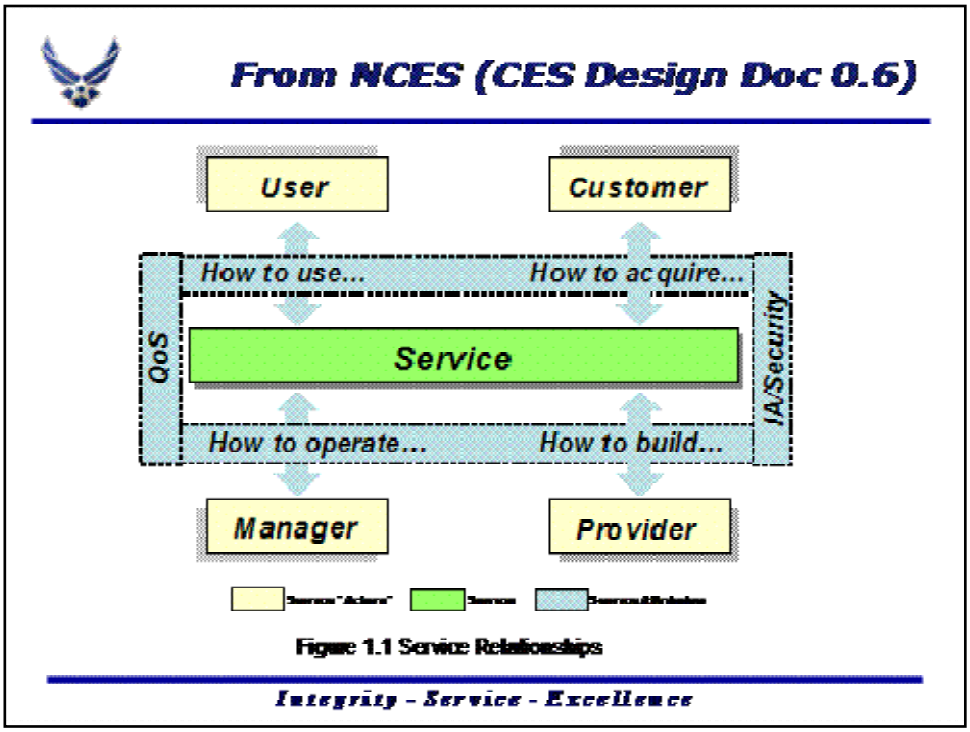
This appendix reviews the foundation of interoperability – services, and a key organizing principle of services, the Enterprise Service Bus. For more detailed reading, please refer to the Bibliography.

C.1 Services

C.1.1 The Unit of Change

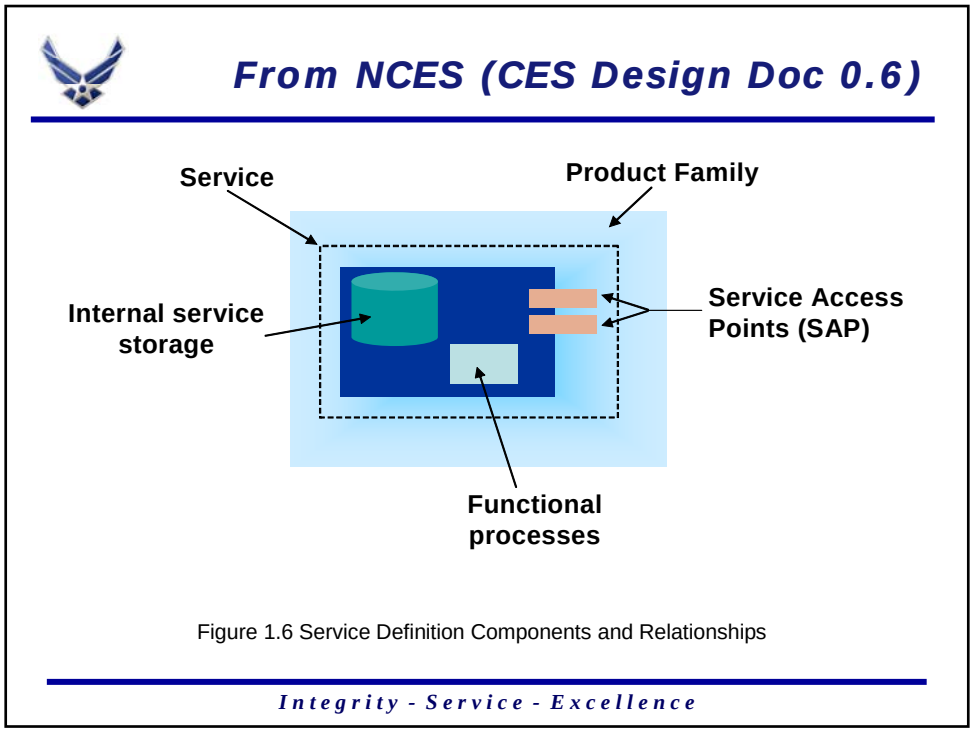
Fundamentally, a service is a unit of change. It is the smallest unit of change in the enterprise. This unit of change is managed via its interface, whether it's expressed as WSDL, and FTP address, or a phone number.

The NCES CES Design Document highlights the fact that every service has relationships with a number of interested parties. Each party will interact with the service in its own manner.



C-1 From NCES (CES Design Doc 0.6)

The NCES CES Design Document illustrates that a production service has service access points. These SAPs are the interface, and can be anything from a WSDL file, in the case of web services, to a JMS interface for publish/subscribe, to an FTP address in the case of a legacy system. In addition, business services such as a help desk are included; in this case the SAP is the phone number.

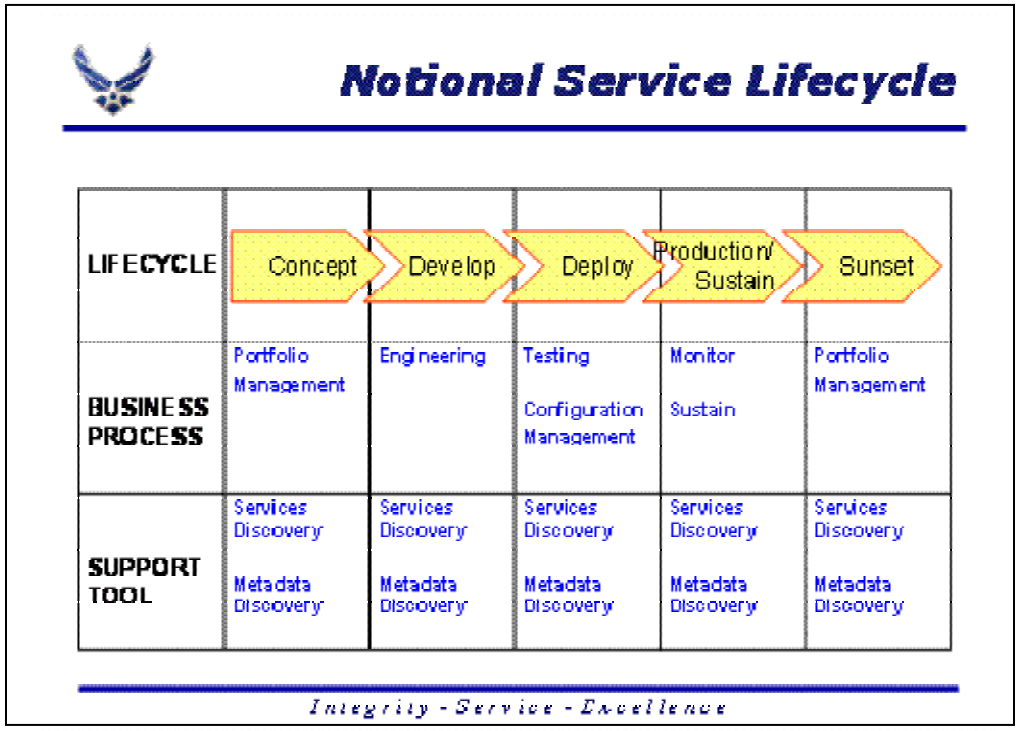


C-2 From NCES (CES Design Doc 0.6)

C.2 Service Lifecycle

C.2.1 Supporting Business Processes

Every service has a lifecycle with milestones between phases. The interested parties will interact with the services via an institutionalized business process. There will be one or more tools used to support the services through these business processes.



C-3 Notional Service Lifecycle

C.2.2 Supporting Tools

This document will refer to tools such as "Discovery" services. There is still much work to be done to completely understand the full set of service lifecycle business processes. Nonetheless, there is utility in considering how the registries of the four main Discovery services will stand in relation to each other. The four main Discovery services, taken from NCES are: Services, Metadata, People, and Content. These relations are considered in the "Principles" section below.

C.2.3 Requires Social Adaptation

One consideration that is not discussed much in the DISA/NCES documentation is the extent of social adaptation required to fully use SOA.

C-4 Social Organizations adoption of SOA

This topic is beyond the scope of this paper; however note that some organizations have radically reorganized around SOA. Please see SOA instruction by R. Wilson (MITRE)[43].

The key point is that all the constituents must adapt. The users, the creators of the services, and the people that maintain and sustain the services need a roadmap for change that shows how to organize around SOA.

C.3 Service Orientated Architecture

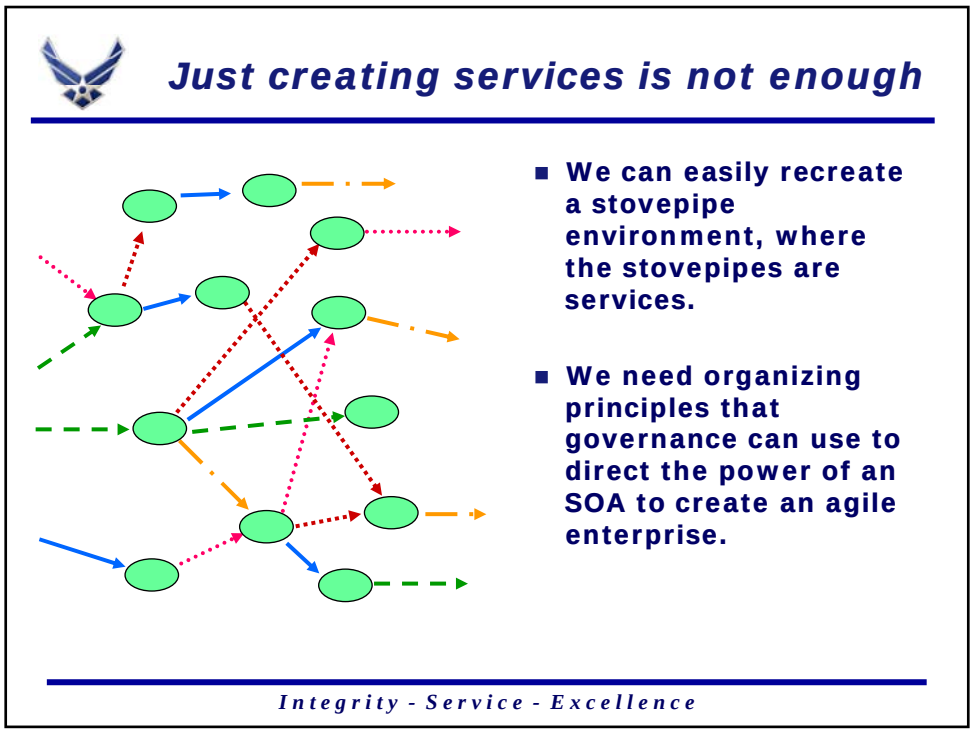
C.3.1 Promise of Compose-ability and Adaptability

SOA architectures promise great advances by easily building composite applications and adapting to changes in the environment. The biggest promise is that of an agile enterprise. An agile SOA-based enterprise does not just magically appear. It's actually a lot of hard work because it is so easy to create services. The trick is to create the "right" services. If there is no governance or organizing principles then we run the risk of recreating a stovepipe environment, with services being the stovepipes.

C.3.2 Requires Organizing Principles

SOA requires governance, and SOA governance requires organizing principles to reduce the complexity of potentially thousands of services.

[43] "D500 BootCamp-Service Eng v0-15.ppt", Robert Wilson, MITRE, April 2005



C-5 Creating Services, is not enough

We will see that that we have a few organizing principles we can apply:

1. Services can be grouped in Nodes
2. Services within Nodes can be organized around an Enterprise Service Bus
3. The Nodes and Services follow certain principles of interoperability and integration that we will see later in this paper.

C.4 Enterprise Service Bus

For a complete treatment of ESB, consult IBM Redbooks, O'Reilly[44], BEA, Sonic, Fiorano, etc. Below is a summary of the architectural principles and usage guidelines for the GCSS-AF ESB, which closely follows best commercial practice.

C.4.1 Motivation

An interesting consideration is the question "why go to all the bother?" If we wanted to just keep doing what we have been doing, then the answer is "let's not bother".

However, a key driving factor in SOA and ESB is the tremendous pressure to change. If it were not for the need to change at a faster and faster pace, we shouldn't bother with SOA and ESB.

What is driving the pressure to change? From the Air Force Operations Support point of view, there are two big factors:

1. Operations Support is straining to support the warfighters' increased tempo requirements

[44] "Enterprise Service Bus", David Chappell, O'Reilly 2004

2. ERP Modules are being introduced that will have a tremendous impact on Operations Support programs (elimination, deprecation, shift to service unique complement to the ERP module)

Programs that survive in the future will be programs that integrate into the ESB, and add value to an ERP module, via a BPEL business process. We'll see how to do this below, in our five phase plan.

C.5 SOA Organizing Principle

The Enterprise Service Bus (ESB) is an organizing principle for services within a node. For a full discussion of the Enterprise Service Bus, see other papers on this subject by this author. The following is a summary.

The primary organizing principle of the ESB is that it is an SOA equalizer. The ESB can take services with a variety of Service Access Points (SAP – a DISA concept) or interfaces, and via adapters, "plug" them in. With either native programming, or adapters, all services will interact on the ESB via JMS for publish/subscribe, or SOAP/WSDL for request/reply. As a complement to SOAP/WSDL, services can be orchestrated in a business process via a BPEL engine.

By constraining the types of interactions to JMS and SOAP, and moving the business process logic to a BPEL engine, rather than hard coding it in the (web) services, we decouple the services from one another and lay the groundwork for an agile enterprise.

C.5.1 5 Aspects

The five aspects of an ESB are:

1. Intrinsic Architecture – an ESB has core architectural features
2. Methods of Integration – an ESB provides three methods of integration
3. Integration Architecture – services and processes use the methods of integration within a Node
4. Extensibility – an ESB is extensible, and separate ESBs can interoperate.
5. Points of Presence – an ESB lives in a data center, even when that data center is distributed

C.6 Intrinsic Architecture

ESB as an architectural construct is not at this time completely defined by standards. However, there is common agreement[45] that an ESB supports:

- **Messaging** – Guaranteed, highly available messaging between participants.
- **Routing** – Ability to route Publish/Subscribe and Request/Reply messages to their endpoint destination without sender knowledge of the ESB topology.

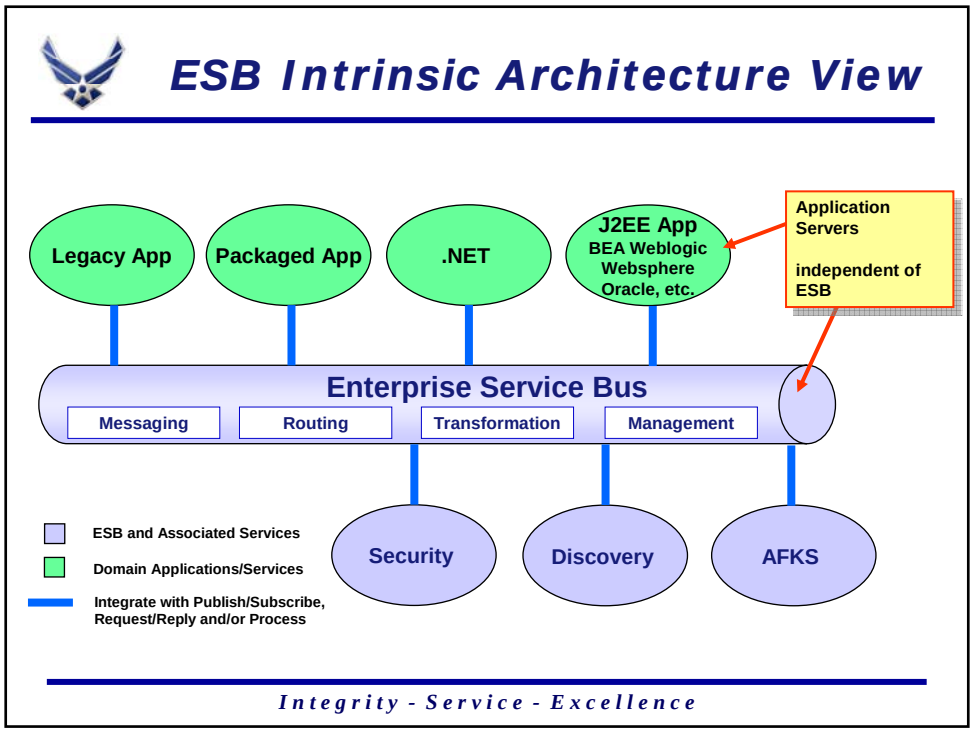
[45] The interested reader can consult the websites of ESB Vendors, such as Sonic Software, Fiorano and IBM to learn more about the architecture of typical commercial ESBs.

- **Transformation** –Transformation between formats and protocols so participants can consume each others payload.
- **Management** – Central management of the federated distributed topology of the ESB is necessary for scalability.

Thus an ESB is a distributed and federated, yet centrally managed message bus that supports transformation of payload. We can think of an ESB as segments of a guaranteed messaging plumbing system.

Each service connects once to the ESB, whether the service is a legacy system, a packaged COTS application (ERP module), or a newly developed application of either J2EE or .NET variety. The connection is via either native technology, or through adapters. An adapter is a bridging capability that is used to connect legacy applications to the ESB. New applications written for the ESB do not need adapters.

Domain applications and ERP modules connect to the ESB with a JMS or SOAP interface. Data sources in one application are not directly accessed by another application. Prior methods of SQL extraction and FTP will be replaced by adapters and publishing data to JMS topics in the ESB. JMS topics are covered below.



C-6 ESB Intrinsic Architecture View

Tenet: GCSS-AF ESB enables integration. Any Community of Interest (COI) can integrate

- Within the enclave
- Extend outside the enclave

This means that GCSS-AF will facilitate machine to machine integration, even if the machines do not reside in the GCSS-AF enclave. This is a crucial point. Even legacy systems residing outside

the enclave can participate in net-centric Publish / Subscribe using the FTP adapters. This is a very low cost way to increase net-centric participation.

Tenet: COI data sources are not directly accessed. Instead we integrate using:

- Publish /Subscribe (JMS)
- Request/Reply (SOAP/WSDL)

This is another crucial point. We will move away from point-to-point brittle, and proprietary database connections, and move toward asynchronous net-centric message exchange. The two types of messages supported are JMS and SOAP. These can be read by any machine and will increase the rate of integration, and improve the flow of information.

Tenet: GCSS-AF ESB will be as "self-serve" as possible by COI applications

- Easy to use integration patterns
- Connecting does not require modifications to GOTS code

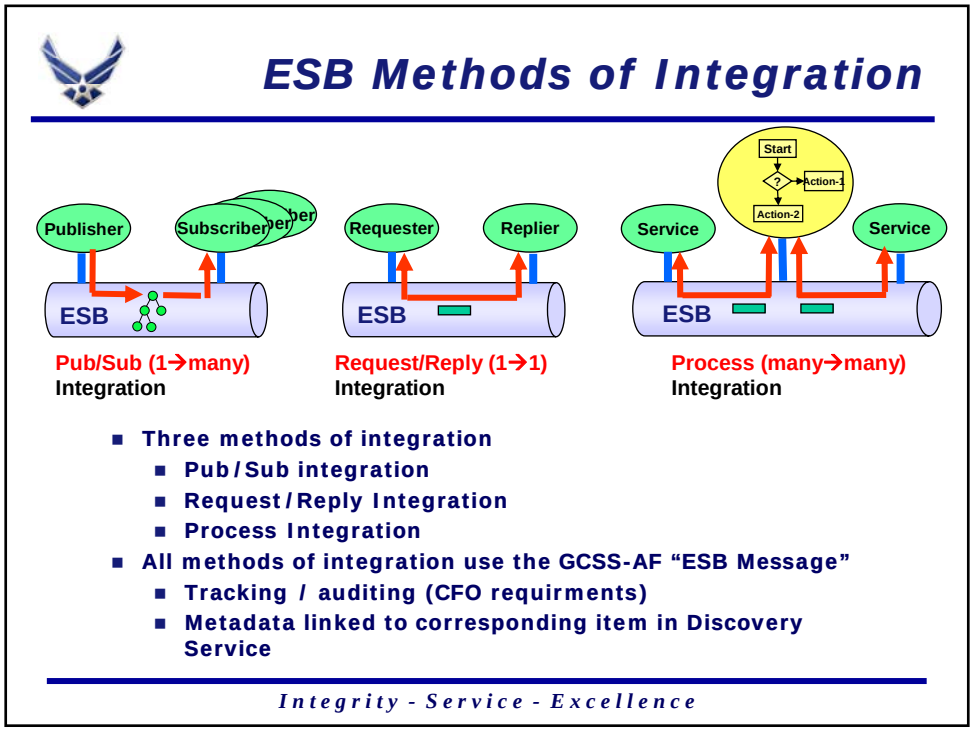
It is imperative that we remove any barriers to entry in this net-centric marketplace of data. Therefore, COI domain applications must be able to find and use simple integration patterns, and connect to the ESB without needing to wait for GOTS code to be rewritten. Therefore, the ESB is 99% COTS, and where there are GOTS adapters (such as the FTP adapter), they are 100% configuration driven.

Tenet: The COIs and programs make data decisions, not the GCSS-AF SPO or other agency. It is important for the COIs to make their own data decisions, and use the ESB as a means to implement those decisions.

C.7 Methods of Integration

The ESB has three integration capabilities which are:

- Publish / Subscribe – the subject of this paper
  - 1-many messaging
- Request / Reply – classic SOAP/WSDL web services
  - 1-1 messaging
  - Query for data
  - Issue a command
  - We recommend non-blocking asynchronous messaging
- Process Orchestration – orchestration of web services
  - If a number of web services need to be coordinated, consider creating an orchestrated process.
  - The number one value of an orchestrated process is that error handling can be done consistently, and the state of the process is centralized.



C-7 ESB Methods of Integration

The **Publish/Subscribe** capability is the JAVA Messaging Service (JMS), and is a standard part of J2EE technology. JMS allows a Publisher to publish a message (report or event) once to a JMS topic, which is a queue inbox with the name of the inbox organized hierarchically [such as Community of Interest (COI) → Application → Report type, …]. Any number of Subscribers will automatically get a copy of the message from the inbox it is subscribing to. This relieves the Publisher from the burden of maintaining who wants the information. The message is deleted once all the Subscribers receive the message. The ESB has the option for a durable Subscriber which leaves a copy of the message in the inbox until a disconnected Subscriber reconnects. A Subscriber also has the option for wildcarding(*) subscriptions which means, for example, that all messages of new ReportTypes will automatically be received, if subscribing at the "Application*" level.

The **Request/Reply** capability is SOAP/WSDL Request/Reply, which is one of the W3C standards. The SOAP message can be transmitted either HTTP/S (not recommended over WAN) or via the guaranteed messaging backbone (recommended). SOAP Requests allow a Requester to Query for data, or Request an action to be performed. Requesting a Query of data is the preferred method (over JMS) when the provider of the data has too much data to publish simple reports. For example a Weather Service could be Queried for "what is the weather – at this lat/long – for this period of time"? In addition, SOAP Requests can be used to perform an action, process a business object, etc.

The **Orchestration** capability is the Business Process Execution Language (BPEL) which provides orchestration and is one of the OASIS standards. The BPEL engine (pictured as a yellow process flow) uses SOAP/WSDL messages to coordinate actions among Services, and maintains centralized process state and logging. One of the Services could be an inbox of a Commander to

include human-in-the-loop processing of exceptions. BPEL provides long running transactions with compensating actions to handle faults. This is the industry standard method to perform transactions longer and more complex than simple ACID or two-phase commit transactions.

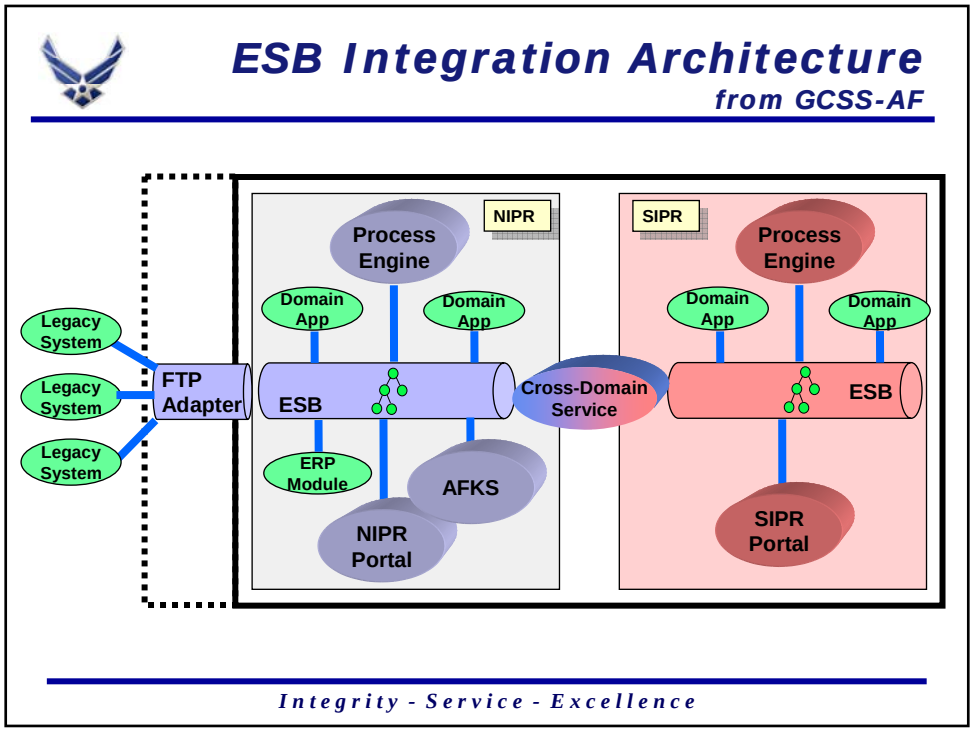
C.8 Integration Architecture

GCSS-AF ESB is a machine to machine integration architecture that is net-centric for both machines and people.

Machine to machine connectivity is achieved with guaranteed asynchronous messaging, in NIPR, SIPR, as well as NIPR → SIPR. Human connectivity remains via the portal and now has a wider range of machines and data to access. AFKS connectivity and value is also increased via the ESB.

In the diagram below we can see domain apps in green either directly attached to the ESB, or in the case of legacy systems, coming into the ESB via FTP adapters. In addition to domain apps, we show the ERP modules in green. All of the domain apps, legacy systems, and ERP modules represent inherent mission capability.

The NIPR portal, the SIPR portal, AFKS (Air Force Knowledge Services, Air Force Data Services) and process engine are all major tools of the integration architecture. The FTP adapter and other adapters provide a low-cost method for legacy systems and other domain applications to plug into the ESB.

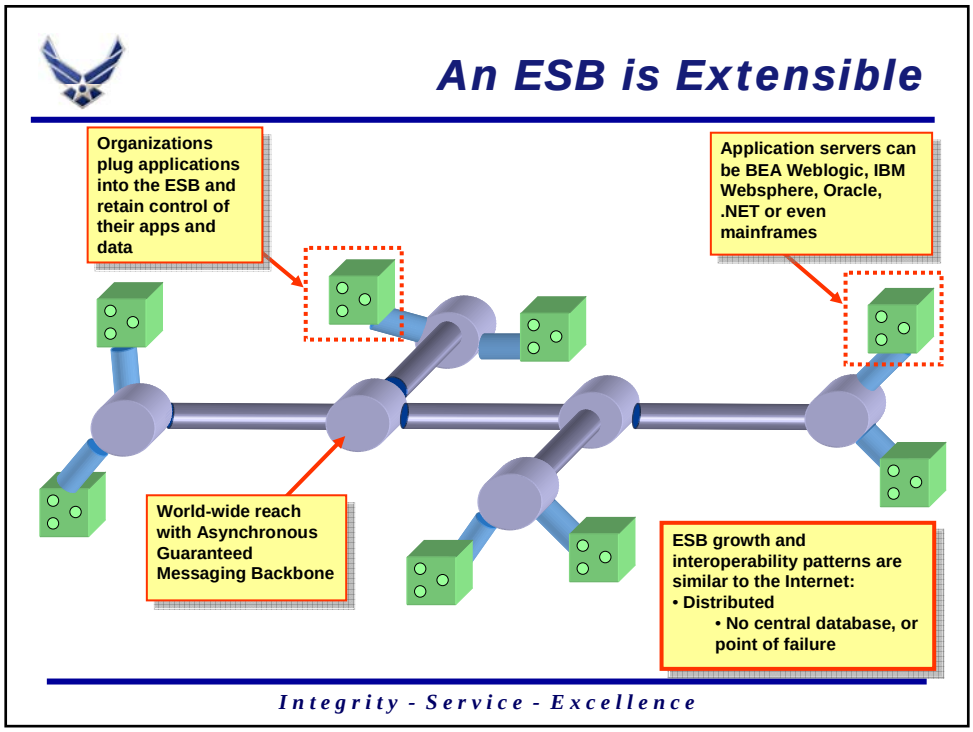


C-8 ESB Integration Architecture

C.9 Extensibility

In the diagram below we see the domain apps in the green square appservers, and the message bus is in the steel gray and blue pipes. The importance of separating apps from messaging cannot be overstated. This separation allows us to connect any app server into the ESB via standard JMS and/or Web service connections. This separation also allows us to grow the ESB messaging backbone independent of any app server.

The ESB makes platform and technology comparison charts obsolete. Now we can connect any system to the GCSS-AF ESB using standard connectors, since interoperability is no longer an issue.

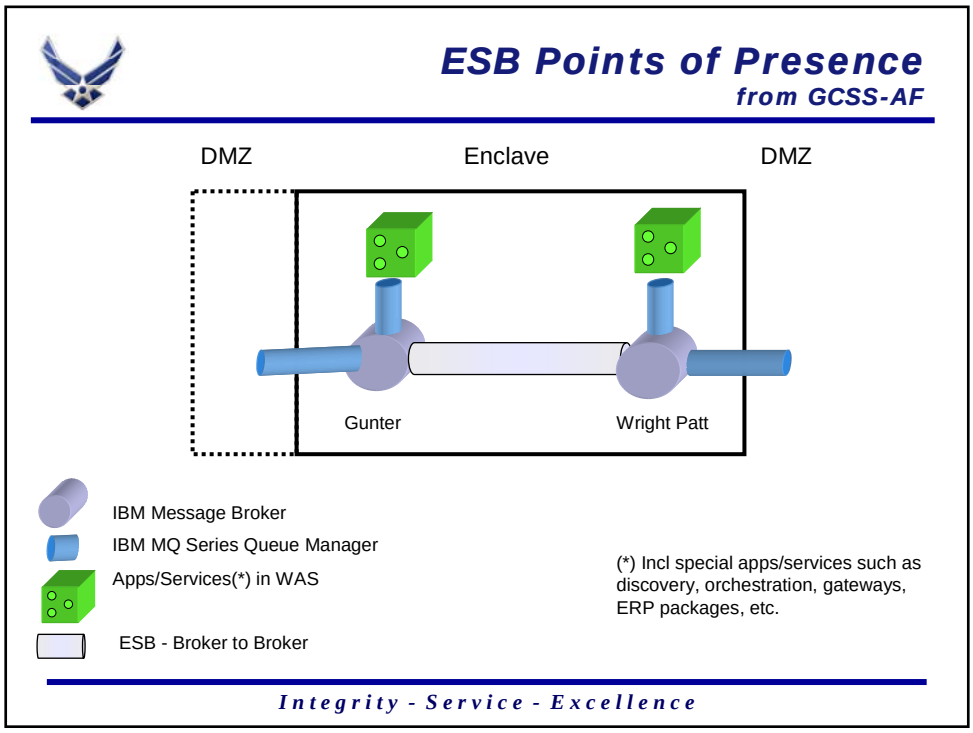


C-9 Extensible ESB

C.10 Points of Presence

An ESB that spans datacenters provides a messaging backbone that spans a distributed enclave. Below we see that the domain applications run inside appservers that are co-resident with persistent queues. This brings guaranteed messaging right next to the applications. Messages are routed by the Message Brokers, and the JMS topic trees reside in the Message Brokers also. Applications from Datacenter-1 or Datacenter-2 do not need to know the location of either other applications, or the JMS topic trees.

We consider the GCSS-AF ESB that spans the enclave to be an “ESB Segment”. This is in recognition that the GCSS-AF ESB will not be the only ESB in the Air Force (or DoD, Federal Gov’t, Coalition, or Commercial sector for that matter) and serves as a useful term to define interoperability below.



C-10 ESB Points of Presence

C.11 5 Phases of Use

A key challenge in adopting any new integration strategy is how to let each individual adopting system adopt at their own pace. This was a key finding in recent federal IRS SOA project failures. Any integration strategy will fail if all participating systems have to change simultaneously.

Therefore, we have developed adoption patterns that allow an individual system to evolve at their own pace. We call these asymmetric evolution patterns. The key points are:

- Being able to break co-dependencies with other applications, especially FTP dependencies, and
- Minimizing the work necessary for adoption

We have developed five basic patterns which are summarized below:

1. Interface consolidation - this is a widely used pattern in commercial practices, where the recommendation is to eliminate redundant functionality, or interfaces. In AF Operations Support terms, if a system outputs five feeds, then that system should consider publishing one feed, that is a superset. Then the subscribers can read the superset and take the data they need. This is a much more cost effective, and net-centric way to publish data.
2. ESB visible/private - this is another widely used pattern in commercial practices, where the recommendation is to hide your implementation, behind a public interface. In AF Operations Support terms, if a connecting system has several

3. hidden backend systems, the backend systems do not have to modernize. Only the connecting system, which has a visible interaction to the ESB, has to modernize or adapt into the ESB.
4. Break FTP co-dependency - this pattern is unique to the GCSS-AF ESB. This pattern allows a legacy system that has FTP co-dependencies to modernize into the ESB without causing rework in dependent systems.
5. Bridge FTP to pub/sub - this pattern is also unique to the GCSS-AF ESB. This pattern allows a legacy system that has FTP co-dependencies to publish into the ESB without modernizing.
6. Output changes instead of complete database - this is an old pattern of commercial practices, where the recommendation is to output transaction activity one activity at a time. In AF Operations Support terms, systems should stop dumping complete database outputs, and start publishing each individual change activity.

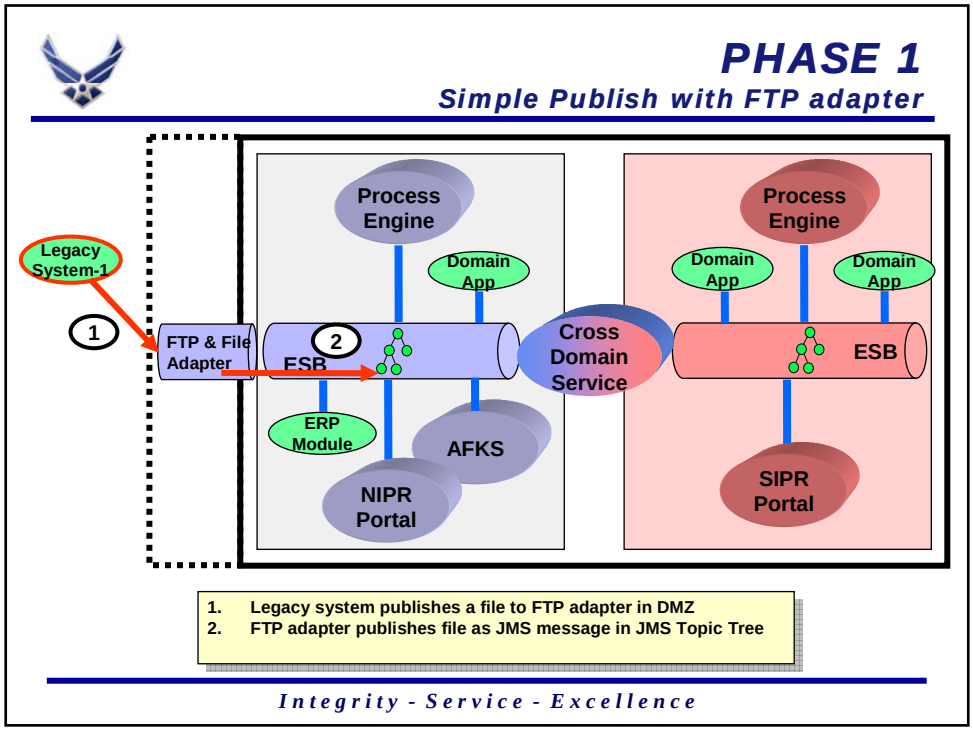
In order to simplify adoption we have recast these patterns into a five step plan of action – the five phases of ESB use. The five phases of use are based on the above principle of "asymmetric evolution". Every native service and legacy system that adapts into the ESB needs to evolve separately and independently from any other service or system. This principle extends to legacy systems that have existing relationships with each other. If a legacy system adapts into the ESB, it must be able to maintain its existing relationships, without incurring programmatic or funding dependencies. We will see a summary of how this asymmetric evolution principle is put into practice below.

1. The five phases of use are:
2. Simple Publish with FTP adapter
3. Consolidate output interfaces/feeds
4. Migrate to XML change records over JMS
5. Add web service interfaces
6. Plug into business processes

The following examples are taken from GCSS-AF.

C.11.1 Phase 1 Simple Publish with FTP Adapter

This is the starting point for most systems in Air Force Operations Support. The majority of systems operate by sending bulk files to each other via FTP. With the ability to break FTP co-dependencies via the FTP adapter, we can migrate one system at a time. The first step is to get all trading partners of the target system to point their FTPs to the ESB instead of the target system.



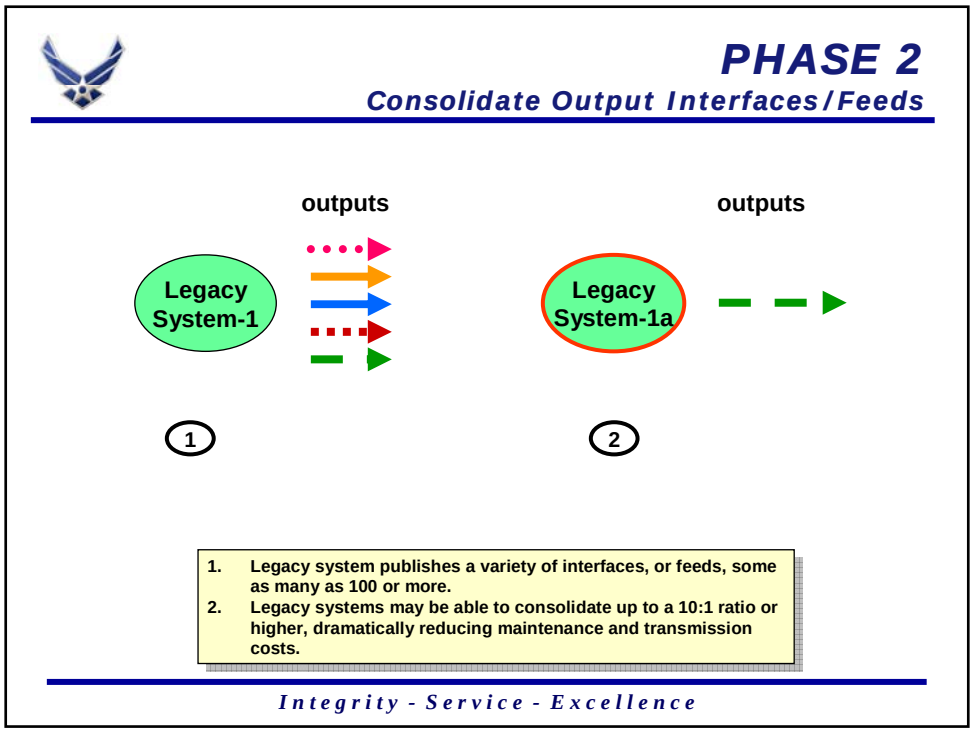
C-21 Phase 1

C.11.2 Phase 2 – Consolidate output interfaces/feeds

Once a legacy system is publishing and its trading partners are subscribing to the ESB, the legacy system can then evolve, in conjunction with it's trading partners, to reduce the number of unique interfaces (usually bulk files).

The usual pattern for reducing the number of interfaces is to replace or supplement many closely related interfaces with one superset. This introduces the challenge of change management. We recommend:

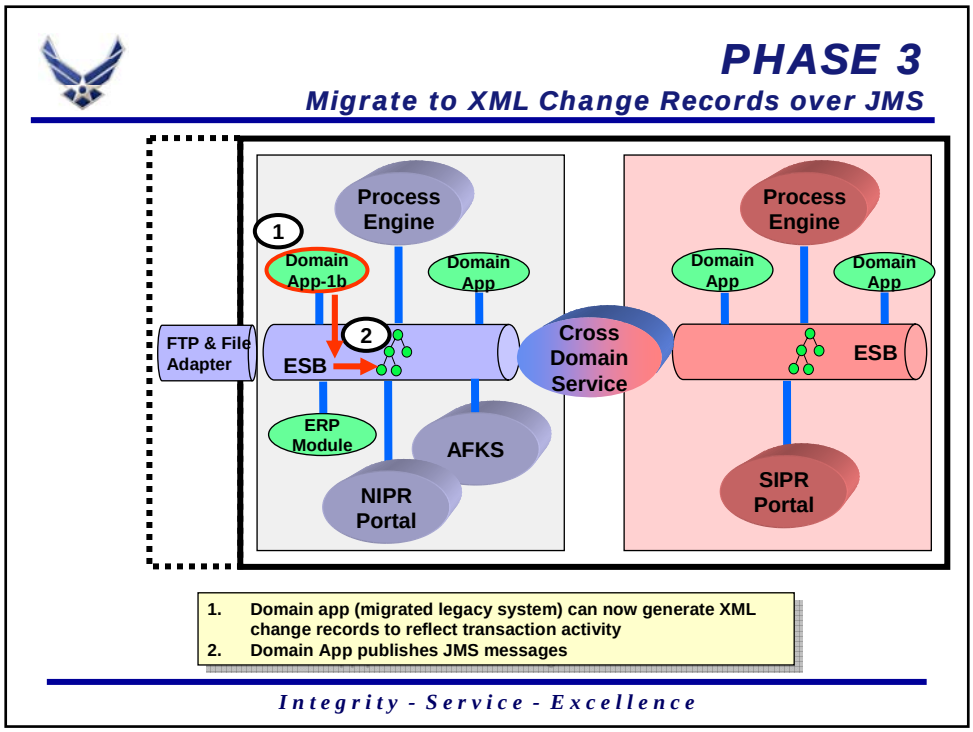
1. Initially keep sending the old interfaces as well as the new superset. This keeps old trading partners happy.
2. Set a date when the old interfaces are "deprecated". This gives programs a timeline for migration to the superset bulk feed.
3. As programs migrate off of the old interface, stop using it, and sunset it immediately – including retiring the JMS Topic



C-12 Phase 2

C.11.3 Phase 3 – Migrate to XML change records over JMS

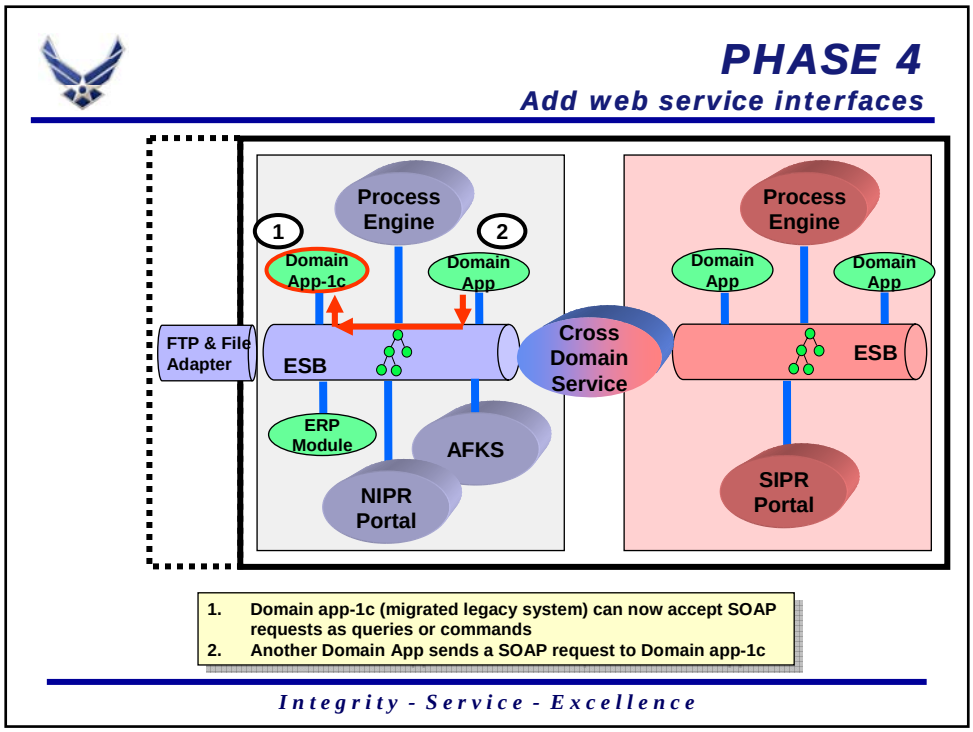
The next step is to evolve from a bulk file output to a single change record at a time. In addition, make the output an XML file instead of a single row of the older format. Lastly, publish via JMS either internally to the enclave, or via an extension cord.



C-13 Phase 3

C.11.4 Phase 4 – Add web service interfaces

If you think the legacy system will need to plug into business processes, you need to add web services. Web services support SOAP/HTTP which is required by a BPEL engine, which is the only supported means to execute a business process.



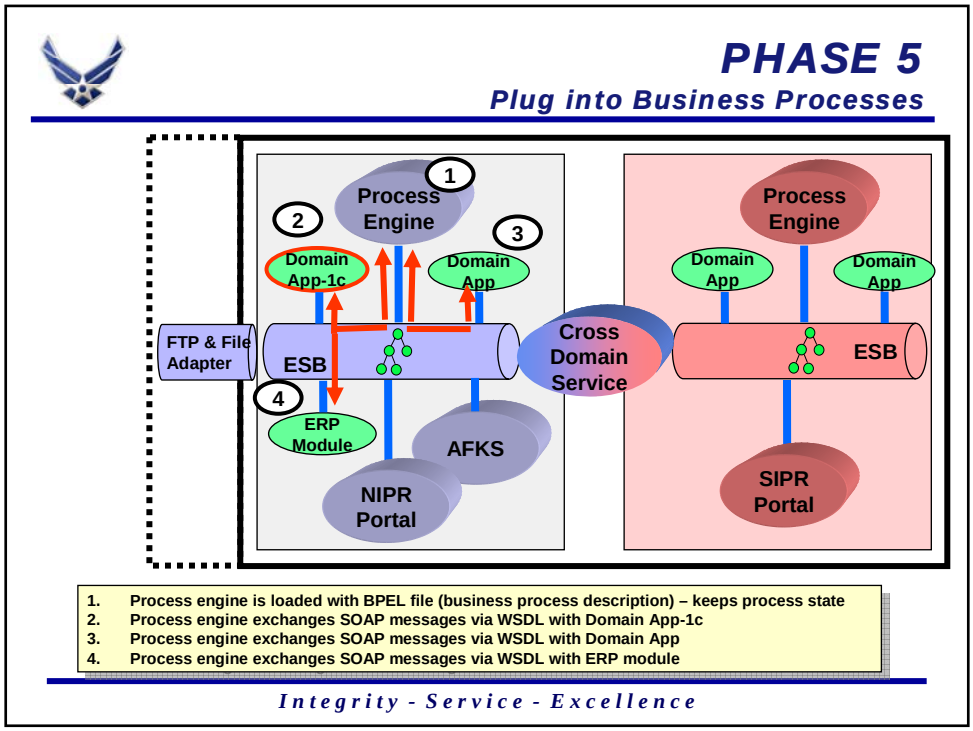
C-14 Phase 4

C.11.5 Phase 5 – Plug into business processes

An important case where you need to plug a legacy system into a business process is if the legacy system will provide "Air Force unique" functionality to ERP functionality. The most important form of integrating with ERP modules is to plug into a BPEL process they own.

The BPEL engine starts a process when it receives a message instructing it to start one. Then, the instructions (similar to flow chart) are read, and all of the actions are translated into SOAP/WSDL calls of the identified systems.

Humans plug into the business process behind the web services. For example one of the actions in the flowchart could be to ask a web service for the decision of a human being. That web service would not respond to the BPEL engine until the human interacted, and provided a decision.

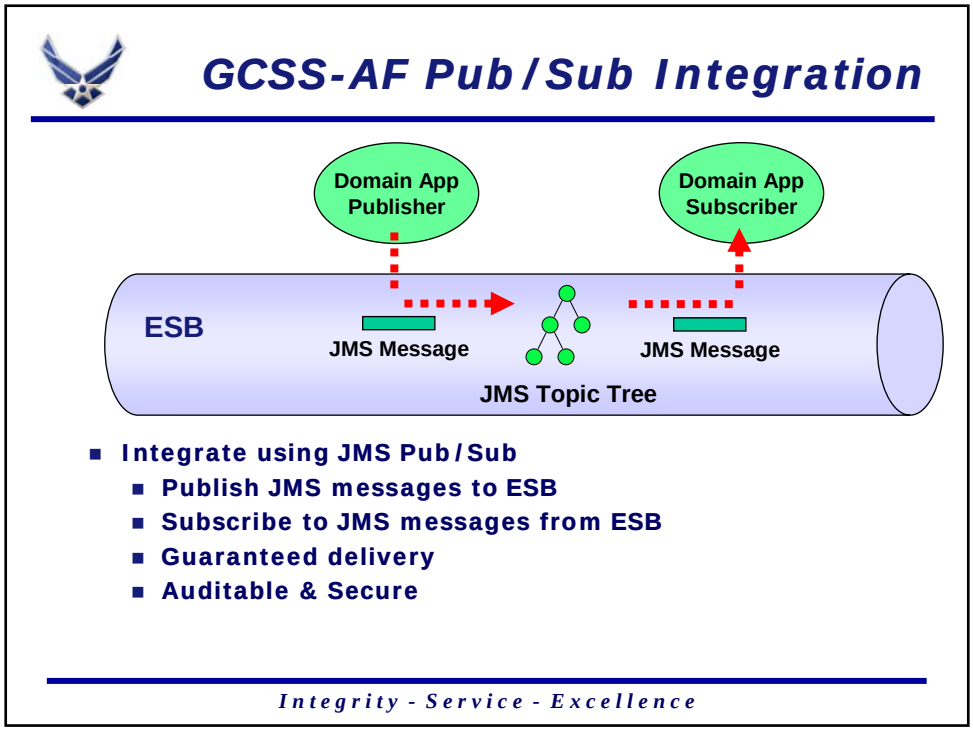


C-15 Phase 5

C.12 Publish/Subscribe

Pub/Sub (JMS) is part of the J2EE standard, and a commercially accepted defacto component of any Enterprise Service Bus. Pub/Sub works by having a Publisher (app or service) create a JMS message and send it to a JMS Topic. One or more Subscribers will then receive the JMS message automatically.

This is more effective than older polling methods for distributing data. For example in Operations Support we have requirements for disconnected users, so the JMS Topic can be configured to wait for a Subscriber to come back on line and get all pending messages.

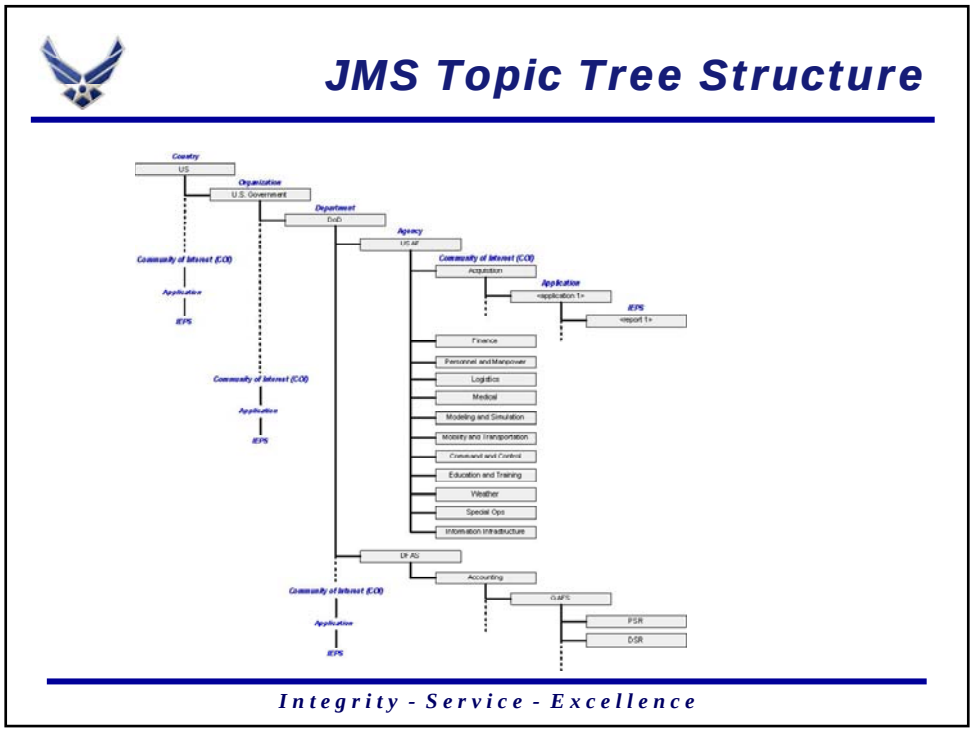


C-16 GCSS-AF Pub/Sub Integration

The published data is organized into a hierarchical tree to allow for wildcarded subscriptions. The general structure consists of a tree that resembles government structure:
US→Gov→DoD→AF→[TRIPLET]

The triplet is:
COI→App Name/Business Process Stage→Report

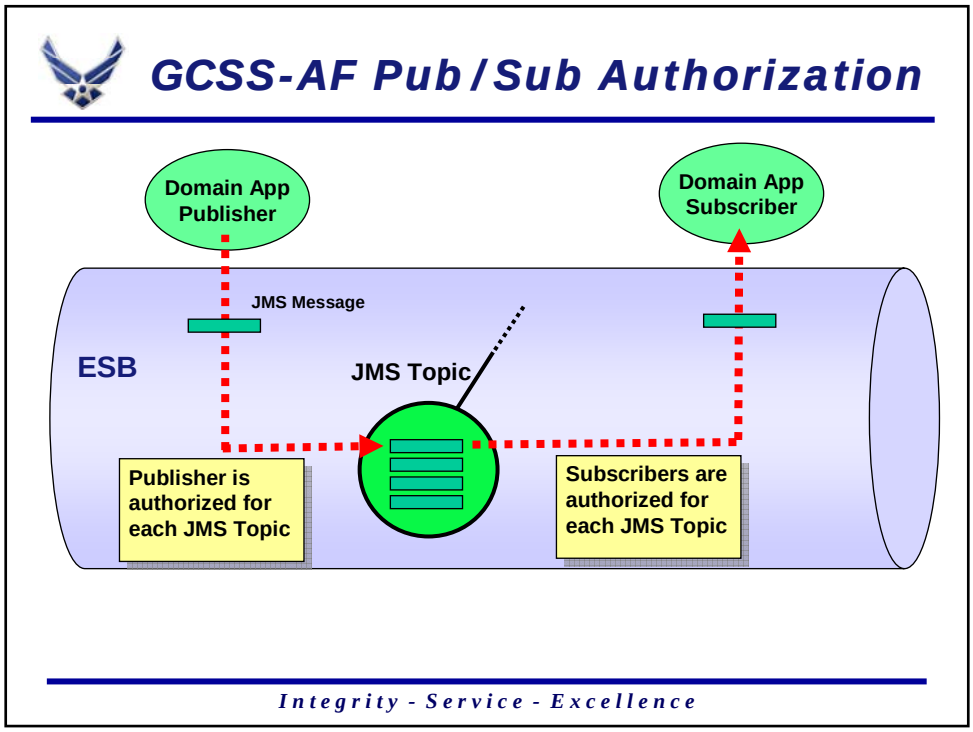
The triplet is "owned" by the COI and can be applied at any point in the structure. The example below shows the triplet extending off of AF, but it can be put anywhere in the government structure.



C-17 JMS Topic Tree

Each JMS Topic can be "locked down" as tight as we need, or left open. We anticipate that the initial uses of Pub/Sub in many cases will emulate the tight specific interface agreements in place today. We also anticipate that for some carefully defined data, there may be more open subscriptions. Currently we are advocating the use of Interface Agreements to facilitate emulating current patterns and accelerating adoption.

As seen in the diagram, the broker that maintains the JMS Topic can be configured to exactly specify what app can publish and what app can subscribe. Authentication is via X.509 or restricted FTP accounts, so identity is assured.

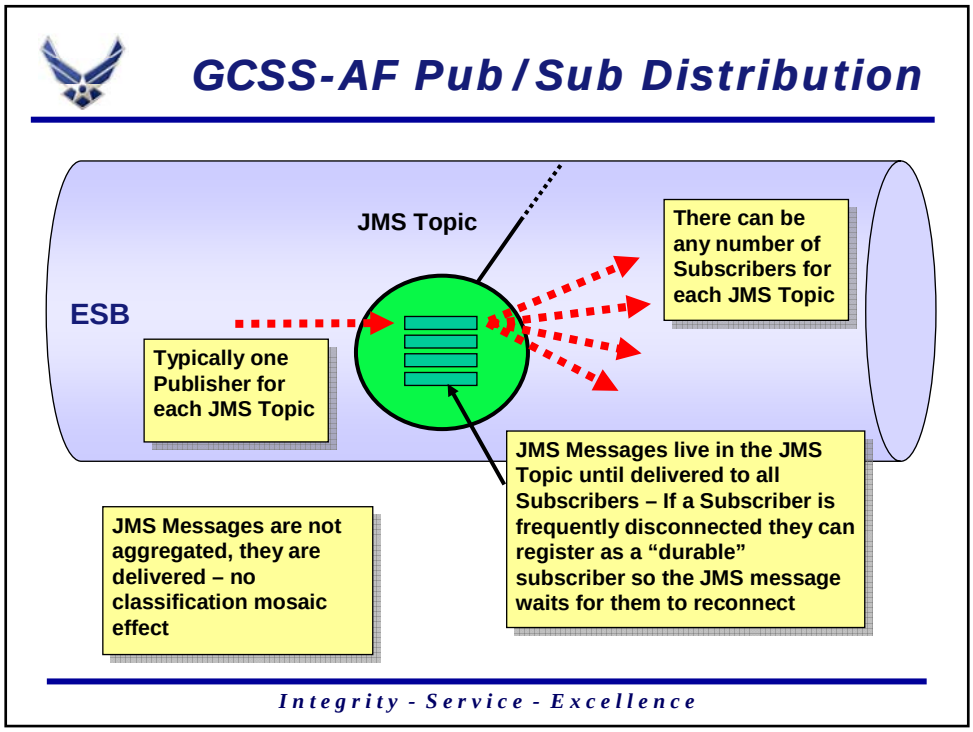


C-18 GCSS-AF Pub/Sub Authorization

JMS Messages are Published into a JMS Topic, then delivered to as many subscribers as are authorized. Subscribers that are marked "durable" will have their messages held until they return if they become disconnected. This creates an on-demand response to data and events flowing in the system, and is more scalable than older distribution methods such as polling a database for updates.

The ESB does not aggregate. Thus, there is no "mosaic effect" that can inadvertently create classified data out of select non-classified data. Messages are Published to JMS Topics, then delivered ASAP by the broker with no waiting for Subscribers.

In contrast, a data warehouse may aggregate by subscribing and holding a variety of data. That is its purpose. In this case, the benefit is to create a wide cross-functional view, and the result may in fact be classified data. If this is the case, then this data needs to be moved to a data warehouse on SIPR.



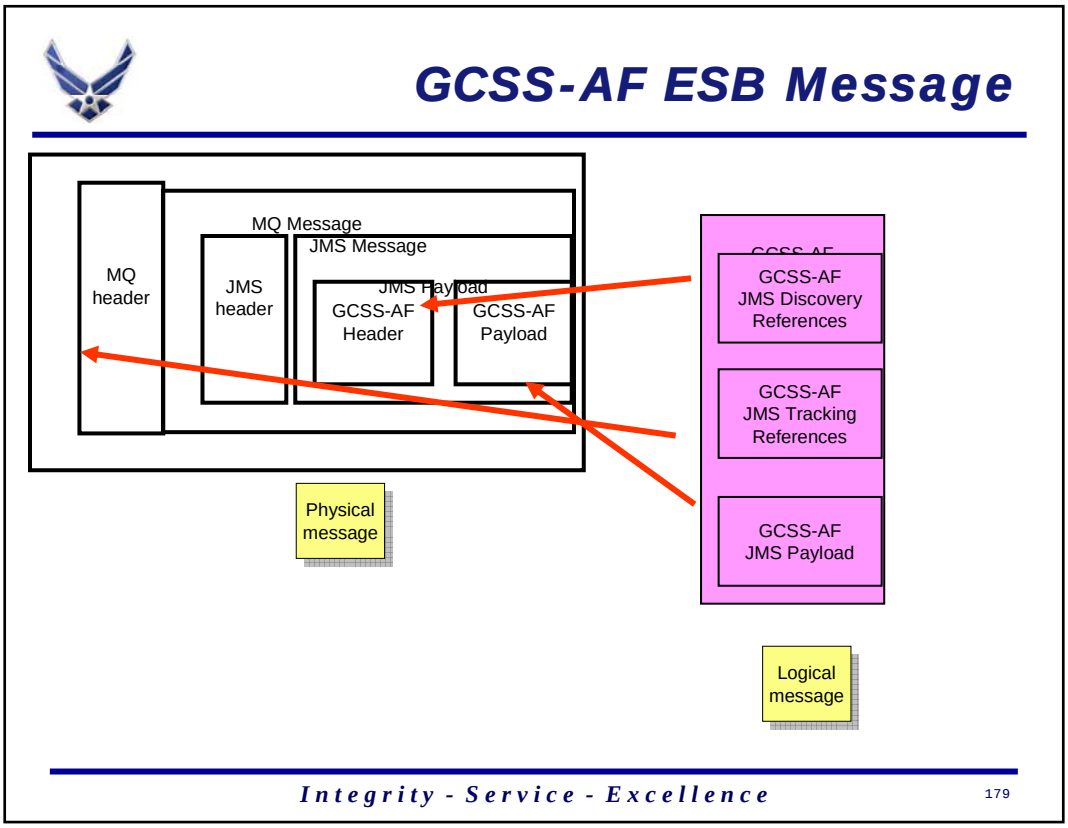
C-19 GCSS-AF Pub/Sub Distribution

C.13 Message Envelope

The Enhanced ESB message envelope supplements a standard commercial message structure. It is a logical grouping of header and payload within a message for consistency for publish/subscribe, request/reply and orchestrated processes. The categories of header information are as follows:

1. Discovery references
    a. Service/application ID
    b. Metadata ID
2. Tracking and auditing references such as
    a. Unique msg#
    b. Timestamp
3. Distributed process state references (future)

An example Message Envelope for JMS and how it corresponds to the wire format is shown below:



C-20 GCSS-AF ESB Message

Distribution List

Internal

D010

George Providakes,

D200

Backman, Tom

Farish,Stephen

Gannon, Thomas F.

D370

Quigley,William

D400

Norman,Douglas

D520

Partridge, Chris

Reed, Harvey

Wilson, Robert

D560

Harris, Deborah L.

D580

Adams, Chris

Chamberlin, Eric

Landsman, Seth

D810

Walsh, Jean

W010

Petroski,Frank

W300

Stein,Fred

Project

ESC/EISS/EID

3 Eglin Street

Hanscom AFB, MA 01731-2100

Besselman, Lt. Colonel Joseph

Farinello, Joseph

Gindhart, Major David

Whitmore, 1st Lieutenant Joseph