

**Department of Veterans Affairs
Office of Information and Technology (OIT) Product Development (PD)**

VA Service Oriented Architecture (SOA) Technical Framework



**DRAFT
V 0.3.1
April 16, 2012**

Draft

Abstract: VA has recognized that a Service-Oriented Architecture (SOA) is the most effective mechanism to allow VA agencies to share information and to implement business processes that span organizations. It also sees the use of an SOA as the most appropriate mechanism for allowing multiple VA organizations to share interfaces with external entities. This document describes the framework within which VA expects to implement its SOA. This document discusses the technical aspects of Service-Oriented Architectures and is intended to provide technical guidance for VA technology infrastructure needed to instantiate the SOA and to allow services to be developed. This document does not deal with the business process reengineering or other changes to VA policies or processes that may accompany a move to an architecture such as SOA. As used in the VA SOA, the term service is distinct from the term “Web service.” While the SOA supports and recommends the use of Web services where appropriate, it recognizes the need for services other than Web services and allows such non-Web services where necessary to meet VA or agency mission requirements. This document assumes that the reader is familiar with large-scale application systems and the methodologies used to support their design and development.

Document Approval

Lorraine Landfried
Executive Director
OIT, Office of Product Development

Date

Additional Signatures (please print name and title below signature):

Date

Date

Date

Date

Document Control

Amendment Record

Issue	Date	Author	Comments
0.1 (Draft)	12/1/2011	Jeffrey Mohr	Initial Draft
0.2 (Draft)	12/15/2011	Jeffrey Mohr	Technical Edit and Content Update
0.3.(Draft)	1/9/2012	Jeffrey Mohr	Updates to reconcile with Enterprise Application Architecture
0.3.1 (Draft)	4/16/2012	Jeffrey Mohr	Minor updates for consistency

The latest version of this document supersedes all other previous issues.

Distribution Record

Issue	Date	Recipients
0.1	12/1/2011	Selected Reviewers
0.2	12/15/2011	ASD and PD Management and Technical Review
0.3	1/9/2012	ASD and PD Management and Technical Review
0.3.1	4/16/2012	ASD and PD Management and Technical Review

Index of Acronyms and Abbreviations

ACL	Access Control List
ASCII	American Standard Code for Information Interchange
API	Application Program Interface
ARM	Application-Capability Reference Model
ACID	Atomic, Consistent, Isolated, and Durable
ATO	Authority to Operate
BAM	Business Activity Monitoring
BPEL	Business Process Execution Language
BPMN	Business Processing Markup Notation
BRM	Business Reference Model
B2B	Business-to Business
CPIC	Capital Planning and Investment Control
COE	Center of Excellence
CA	Certificate Authority
CRL	Certificate Revocation List
C&A	Certification & Accreditation
CCB	Change Control Board
CIO	Chief Information Officer
COTS	Commercial Off the Shelf
COI	Community of Interest
CCBS	Core Common Business Services
CCIS	Core Common Infrastructure Services
DAS	Data Access Services
DIS	Data Interchange Services
DCOM	Distributed Component Object Model
DRM	Data Reference Model
DBMS	Database Management System
DoD	Department of Defense
DAA	Designated Approving Authority
DR	Disaster Recovery

DNS	Domain Name Service
DLL	Dynamic Link Library
EDI	Electronic Data Interchange
EDIFACT	Electronic Data Interchange For Administration, Commerce and Transport
EAI	Enterprise Application Integration
EA	Enterprise Architecture
ELDM	Enterprise Logical Data Model
ERAM	Enterprise Release Allocation Matrix
ESB	Enterprise Service Bus
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations
ETL	Extract, Transform, and Load
SRM	FEA -- Service Component Reference Model
FBCA	Federal Bridge Certificate Authority
FEA	Federal Enterprise Architecture
FEA	Federal Enterprise Architecture
FIPS	Federal Information Processing Standard
Fisma	Federal Information Security Management Act of 2002
FPKIPA	Federal Public Key Infrastructure Policy Authority
FTP	File Transfer Protocol
FOC	Full Operational Capability
GUID	Global User Identification
GAO	Government Accountability Office
GUI	Graphical User Interface
HL7	Health level Seven
HSPD	Homeland Security Presidential Directive
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secured Socket
MQ	IBM WebSphere MQ Series
IV&V	Independent Verification and Validation
IOC	Initial Operational Capability

ISA	Interconnection Security Agreement
ICD	Interface Control Document
IATO	Interim Authority to Operate
IA	Internal Affairs
J2EE	Java 2 Platform, Enterprise Edition
JDBC	Java Database Connectivity
JMS	Java Message Service
LDAP	Lightweight Directory Access Protocol
LAN	Local Area Network
MHS	Military Health System
MI	Major Initiative
MTBT	Mean Time Between Failure
MTTR	Mean Time To Recover
MOA	Memorandum of Agreement
MOU	Memorandum of Understanding
MOM	Message Oriented Middleware
MOM	Message Oriented Middleware
NARA	National Archive and Records Administration
NIEM	National Information Exchange Model
NIST	National Institute of Standards and Technology
OLAP	On-line Analytical Processing
OLTP	On-line Transaction Processing
ODBC	Open Data Base Connectivity
ODMS	Operational Data Management System
PRM	Performance Reference Model
PRM	Performance Reference Model
PED	Personal Electronic Device
PII	Personal Identifiable Information
PIV	Personal Identity Verification
POC	Point of Contact
PIA	Privacy Impact Analysis
PTA	Privacy Threshold Assessment

PHI	Private Health Information
PMAS	Project Management Accountability System
PKI	Public Key Infrastructure
PUB	Publication
RFID	Radio Frequency ID
RA	Registration Authority
RMI	Remote Method Invocation
RPC	Remote Procedure Call
Root CA	Root Certificate Authority
REST	Representation State Transfer
SSL	Secure Socket Layer
SAML	Security Assertion Markup Language
SRA	Security Risk Assessment
ST&E	Security Test and Evaluation
SBU	Sensitive But Unclassified
SQL	Sequel Query Lane
SCBA	Service Based Component Architecture
SCRM	Service Component Reference Model
SIP	Service Insertion Package
SLA	Service Level Agreement
SRM	Service Reference Model
SO	Service-Oriented
SOA	Service-Oriented Architecture
SOx	Service-Oriented Architecture or Service-Oriented Integrations
SOI	Service-Oriented Integration
SOAP	Simple Object Access Protocol
SLS	Single Level Security
SDLC	Software Development Reference Model
SP	Special Publication
SORN	System of Record Notice
SDLC	Systems Development Life Cycle
TRM	Technical Reference Model

TRM	Technical Reference Model
TLS	Transport Layer Security
UDDI	Universal Description, Discovery, and Integration
UID	User Identity
UTC	Coordinated Universal Time
VA	Veterans Administration
VistA	Veterans Health Information Systems and Technology Architecture
VPN	Virtual Private Network
WSDL	Web Service Definition Language
WSIL	WS-Inspection
XSD	XML Schema Definition

Table of Contents

1.	Service-Oriented Architecture	15
1.1.	<i>Introduction</i>	15
1.2.	<i>SOA Services vs. Web Services</i>	15
1.3.	<i>Role of Services</i>	17
1.4.	<i>Goals of the VA SOA Effort</i>	17
1.5.	<i>Purpose of this Document</i>	18
1.6.	<i>Scope</i>	18
2.	Basis for VA Service-Oriented Architecture	19
2.1.	<i>Relationship of this SOA to the OneVA EA and other Architecture Documents and Guidelines</i> <i>20</i>	
2.2.	<i>Relationship of Services to the VA Systems Development Life Cycle</i>	21
2.3.	<i>Service Support for Business Process Management</i>	21
2.4.	<i>Service Centricity</i>	22
2.5.	<i>Types of Services</i>	29
3.	SOA Implementation Considerations	29
3.1.	<i>Use of Standard, Syntactically, and Semantically Harmonized Data Definitions</i>	29
3.2.	<i>Extensible Markup Language (XML) as a Wire Format</i>	31
3.3.	<i>Need for Services to be Independent of Legacy Systems</i>	33
3.4.	<i>Need to Tie SOA to Technology-Based Architectures</i>	33
3.5.	<i>Nature of Services to be Provided</i>	34
3.6.	<i>Architecture Tiers / Service Layers / Service Types</i>	35
3.7.	<i>Enterprise Service Bus</i>	42
4.	ESB Conceptual Model	53
4.1.	<i>Interface Layers</i>	55
4.2.	<i>Functional Services Layer</i>	55
4.3.	<i>Utility Services Layer</i>	56
4.4.	<i>COTS Messaging Layer</i>	56
4.5.	<i>VA SOA Management and Governance Tower</i>	57
4.6.	<i>VA SOA Security and Privacy Tower</i>	57
5.	ESB Logical Model	58

5.1.	<i>Interface Level Services</i>	58
5.2.	<i>Web Services / Services</i>	60
5.3.	<i>Services Management</i>	62
5.4.	<i>Logical Model Component Descriptions</i>	63
5.5.	<i>Operational Overview</i>	70
6.	Governance	72
6.1.	<i>Use of the OneVA EA as the Basis for SOA Governance</i>	74
6.2.	<i>Portfolio Management</i>	75
6.3.	<i>Development, Specification, and Modification of Services</i>	75
6.4.	<i>Systems vs. Services</i>	78
6.5.	<i>The Effect of the SOA on Cross-Project Dependencies</i>	79
6.6.	<i>Project Responsibilities</i>	80
6.7.	<i>Sources for Services</i>	81
6.8.	<i>Service Contracts</i>	83
6.9.	<i>Service Level Agreements</i>	84
6.10.	<i>Managing SLAs</i>	86
6.11.	<i>Repositories, Directories, and Registries</i>	86
6.12.	<i>Service Discovery and Service Registries</i>	88
6.13.	<i>Publishing Services</i>	89
6.14.	<i>Quality Control for Services Published at the Enterprise Level</i>	89
7.	Data Architecture	90
7.1.	<i>Data Ownership and Stewardship</i>	90
7.2.	<i>Data Services</i>	91
7.3.	<i>Data Sharing</i>	94
8.	SOA Security	103
8.1.	<i>Domain Model</i>	106
8.2.	<i>Token or Credential Based Security</i>	108
8.3.	<i>Security Implementation for Services</i>	111
8.4.	<i>Identity Management Services</i>	114
8.5.	<i>Services Logging</i>	115
9.	SOA Services Programming	118

9.1. <i>Services Programming Model</i>	118
9.2. <i>Publishing Services</i>	119
9.3. <i>Exposing Web Services</i>	119
9.4. <i>Service Contracts</i>	122
9.5. <i>Programming Synchronous vs. Asynchronous Services</i>	122
9.6. <i>Atomic Services</i>	122
9.7. <i>Orchestration</i>	123
9.8. <i>Compensating Services</i>	123
9.9. <i>Correlation</i>	123

List of Figures

Figure 1 - Data Adapter to Translate from a Native Application Format	30
Figure 2 - Placement of XML Adapters.....	32
Figure 3 - Architectural Layers.....	36
Figure 4 - SOA Stack in the Enterprise Application Architecture	43
Figure 5 Point to Point vs. Standard Messages with Hub	44
Figure 6 - ESB Capabilities	46
Figure 7 -- Exception Processing for Orchestrated Services	50
Figure 8 - ESB Conceptual Model.....	54
Figure 9 - ESB Logical Model.....	65
Figure 10 - Data Access and Data Interchange Services	92
Figure 11 - Possible Locations for Data Transformation Adapters	93
Figure 12 - Data Reference Characteristics.....	100
Figure 13 - Recommended and Allowed Data Transfer Mechanisms vs. Organizations to which the Data is Transferred.....	102
Figure 14 -- Digital Signature Process Flow	110
Figure 15 - Service Providing Multiple Functions and Multiple Interfaces.....	118
Figure 16 - Orchestration of a Service	120

List of Tables

Table 1. Data Transfer Mechanism.....	97
Table 2. Data Sharing Relationships.....	99
Table 3. Data Reference Recommendations.....	102
Table 4. Key Security Capabilities	114

1. Service-Oriented Architecture

1.1. Introduction

A Service-Oriented Architecture (SOA) is an architecture in which business functionality is provided by a set of discrete, communicating “services” rather than by monolithic applications¹. VA has determined that using an SOA to provide internal and external applications access to data owned by and processing performed by VA is the best approach to sharing data within VA, acquiring data from external sources, providing data to external users, and reducing redundancies across the VA business areas. The SOA is also viewed as the best approach for both sharing data and assuring interoperability across VA applications as well as supporting cross functional integration and data sharing across functional silos within a single business area and across business areas. Each of these uses of an SOA is viewed as equally important aspects of this SOA.

The use of an SOA is intended to provide many operational benefits, including improving reuse of code, providing more current views of data to all who need it, improving business functionality; all while lowering costs and improving overall operational effectiveness. High levels of code reuse come from having a single service to perform any given function, rather than the more traditional use of code from a code library.

Some consider an SOA to be an abstraction layer that provides agile business services to the user while hiding the underlying complexity of the technology. However, an SOA is much more than just an abstraction layer. It provides a very different approach to architecting solutions than has been used previously and requires a mind shift from designing monolithic (and ever growing) stovepipes to designing reusable components that can be combined in multiple ways to perform different functions.

One issue with SOA is that there are many definitions of SOA and many organizations have implemented something that they call an SOA, but each of which is markedly different. Therefore, lacking a standard definition of SOA, the definition of SOA as used in this Technical Framework is as defined in this Technical Framework, and should not be confused with alternative definitions that may be used elsewhere.

1.2. SOA Services vs. Web Services

There is general agreement in the literature that the term service as used in SOA is distinct from the term service as used in “Web services,” which are services specified by and provided based on a specific set of standards – usually with a name in the form WS-XXXX. As used in this paper, the term “Web services” specifically refers to services provided in compliance with that set of standards.

¹ Architecture normally includes much more than just technology. An architecture normally includes business functionality, people, and process as well as technology. However, the SOA described in this document is a technology architecture. This document does not describe changes to the VA business or process architecture that result from implementation of the SOA. This SOA is a technology / application architecture and not a business architecture. Aspects of the VA architecture including descriptions of business functionality, people, and processes are included in other portions of the OneVA EA.

As used in this paper, services is a more general term that includes Web services, but also includes services that are not compliant with those standards.

“Web services” as that term is most often used are very coarse grained services and will be used primarily for interactions with users and systems external to VA. The term Service-Oriented Integration (SOI) has been used to refer to the use of services to expose functionality of legacy systems and thus foster the integration of functionality of those systems into a modern enterprise. Some writers have used to acronym SOx to refer to both SOA and SOI. Initially at least, a major use of services implemented as part of this SOA will be to expose functionality of legacy systems and to serve as a basis for the modernization of VA systems and thus, more properly fall under the rubric of SOI. In this paper, the term SOA is used to cover the full range of service related concepts including both SOA and SOI, but decidedly not limited to Web services². This SOA covers providing services both to provide interoperability between VA systems as well as to support integration of those systems whether they are in the same business area or MI or whether they are in different business areas or MIs.

The use of services is an architectural style and paradigm for how applications are accessed and used substantially predates the “Web” or “Web services.” A Service-Oriented Architecture could be developed entirely for internal use and without regard to any of the Web services standards and protocols etc.³ Web services includes a number of standards that allow services developed by different organizations to interact in a standard manner and to develop levels of trust between the service provider and the consumer. An organization could implement an SOA and support no Web services or Web services standards. An organization could also implement a set of Web services while still supporting its business through a set of monolithic mainframe services. In fact, this approach is a very common approach to SOA implementation. The approach taken in this SOA is to publish a set of fine grained services that can be orchestrated into large complex systems.

One goal of this SOA is to change the orientation of VA systems from being large monolithic systems that grow ever more complex as new requirements are added over time. The issue with these complex monolithic systems is that they grow ever more costly to support and maintain in part because any change to the system may require regression testing of the full system. By building systems through the orchestration of fine grained services new requirements can be met by building a small number of new services and orchestrating those new services with existing services – allowing each orchestration to be tested independently, thus decoupling the changes from each other and limiting the growth in the complexity of the systems.

² This SOA involves layering what may be termed a conventional SOA over a traditional n-tier application architecture thereby obviating the possibility of implementing traditional Web services that provide external service consumers with access to internal functionality. Services in the presentation layer may provide “Web services” to external users or systems and may make service requests (typically by means of MQ Series messages) to business logic tier services. It is the services at the business logic tier that are the primary focus of this effort.

³ Although the use of “services” as the basis for an architecture substantially predates the development of “Web services”, SOAs did become highly popular until the advent of “Web services” based on open protocols developed.

Externally exposed services should conform to applicable Web services or other required “non-service” standards. These services will be relatively coarse grained and will provide relatively major functional capabilities. Services that are only exposed within an application will likely be much finer grained and are less likely to conform to Web services standards. The notion is that services exposed external to VA will be relatively coarse-grained services – services that perform a reasonable amount of work. These services may be composed of a number of fine-grained services. Since services are also viewed as a mechanism to simplify applications and to speed application development, services may need to be very lightweight and to be used where there is a very high degree of trust between the service provider and service consumer – a much higher degree of trust than typically exists between Web service consumers and Web service providers who are typically unknown to each other.

Each VA business area or MI may also internally publish a relatively much larger number of services designed exclusively for use within that business area or MI that will not be published externally⁴ and which will not be directly accessible external to the business area or MI, but which may be called or accessed by business area or MI services that are externally published. These internal services may, or may not, conform to some or all Web services standards. Further, individual projects may develop and publish services exclusively for use within their internal application that may not be compliant with Web services standards. These internal services will be relatively fine-grained and will perform discrete functions.

1.3. Role of Services

This SOA draws a distinction between SOA services and Web services or even large coarse grained services. This SOA requires the use of services for the full development of applications specifically to include all data access logic, business logic, presentation logic, and interface logic. This means that this SOA must define the full range of services required to support all aspects of system operation. Therefore, this SOA will define a far wider range of services and the infrastructure required to support them then might be typical in many SOA descriptions.

1.4. Goals of the VA SOA Effort

Major goals of the implementation of the SOA described in this document include:

- To provide information sharing across VA and between VA and DoD
- Reduce the implementation of duplicative interfaces with external organizations and the implementation of the same capabilities in multiple VA systems
- Foster greater reuse of existing services that reduce cost and maximize application efficiencies

⁴ While the statement that many services may not be exposed externally to a business area, MI, or system may seem to contradict the goal of this SOA which is intended to promote interoperability and integration across systems, business areas, and MIs, the use of very fine grained services makes exposing all services across the enterprise impossible as many fine grained services will necessarily be built on the assumption that certain work has been performed (e.g., inputs have been validated, data has been converted to an internal format, users have been authenticated etc.)

- Implement all modernized applications completely as collections of services and that services are not just used to share information between applications or between agencies, but as the core of the system
- Allow projects to rationalize and modernize those systems without impacting users of the information
- Allow systems and databases to be updated, merged, and / or rationalized
- Provide for the development of syntactically and semantically harmonized messages across VA

1.5. Purpose of this Document

This document is intended to serve as a framework to guide the instantiation of an SOA and associated technologies throughout VA to promote the sharing of information between VA and DoD, across portfolios of Major Initiatives (MIs) and business areas within VA, and with organizations outside of VA. These uses of an SOA would be considered to be the typical use of an SOA and application of services. However, this framework is also intended to provide the basis for the implementation of applications using finer grained services than would be used to share information across business areas or MIs. The use of an SOA for these purposes embodies a significantly different application development philosophy than is commonly used and which has resulted in the complex, monolithic applications that are common today. Because a change to application architecture philosophy is included, it is necessary to discuss the philosophical basis for SOA and the development of services and applications rather than just providing a set of directions on how to implement the SOA.

This document is not intended as an implementation document for specific services, specific applications, or the infrastructure needed to support those services; but only as an architectural framework. Later documents will contain the technical detail describing the implementations of the capabilities identified in this document. Those documents will provide the technical details of the implementation of the VA SOA.

1.6. Scope

This document is intended to provide a framework for the implementation of the VA SOA. This document describes many details of the VA SOA and its use as an underlying architecture for the development of applications and for sharing information between VA organizations and between VA organizations and organizations external to VA. This document does not address the services that will be implemented by VA organizations or the message sets that will be used to communicate this information. This document is oriented towards the architectural issues related to the development of services, the definition of Core Common Services to be used across all of VA and the services to be used within a single VA Business Area. This document pertains primarily to the infrastructure and the application structures needed to provide and access those services.

The implementation of the SOAs will be accomplished within the larger context of the approved VA Enterprise Application Architecture (EAA) and VA IT Architecture Principles. Any efforts to implement services consistent with this SOA will need to comply with the EAA and the IT Architecture Principles.

It is not the intent of this framework document to cover all of the subjects contained of an SOA or to discuss the full range and extent of architectural issues inherent in the design and development of

systems. Architectural issues are only presented to the extent that they either impact the SOA or are impacted by the SOA. It is recognized that the discussions included herein are abbreviated and are necessarily incomplete as they address architectural issues specifically relevant to SOA versus traditional development. This SOA framework document provides the first definition of the VA SOA. Further documents, to be provided over time, will provide additional specificity on SOA governance, SOA infrastructure design, and SOA services.

Application designers and developers are responsible for understanding all other design guidance, Software Development Lifecycle (SDLC), enterprise architecture, and security requirements that already exist within VA. This includes responsibility to adhere to the Project Management Accountability System (PMAS) and ProPath direction.

This document provides an application and application transport level description of services and the SOA. It does not provide descriptions below that level and in particular does not delve into the details of the communications layer that must underlay this architecture. The communication layer is described in the EAA. Therefore, this document does not deal with topics such as the encryption of data for transfer into or out of the VA or between the VA and DoD; nor does it deal with the details of the location and configuration of the firewalls and routers needed to provide the connections between services. This document does not describe the physical implementation of the services or the underlying infrastructure necessary (e.g., configuring multiple, redundant servers to provide high levels of reliability and availability).

2. Basis for VA Service-Oriented Architecture

This SOA is a federation of SOAs and SOA infrastructure. The VA SOA will include shared infrastructure and an architecture designed to allow the sharing of information. The SOA is designed to allow services to be shared across VA or to be restricted to a single business area, MI, or application. Further, since the VA SOA is intended to serve as the basis for the development of applications, it is necessary to maintain the integrity of the applications to ensure that all data validation and controls are maintained.

Fundamentally, in an effective SOA implementation, integration is a byproduct, rather than an enabler, of the architecture. This difference distinguishes SOA from other distributed computing architectures that came before it, such as client/server or n-tier. Both of those earlier architectures specified logical tiers and the connections between them, using middleware as the glue that created connections between systems in a tier and across tiers – and the more tiers, the more “glue.” In the previous n-tier architectures any combination of data across systems either occurred “at the glass” (i.e., on the workstation screen) or through complex middleware that would need to do conversions of data definitions between disparate data from multiple systems on the fly. In these architectures the glue is required to perform the complex transformations of data required to allow integration across independently developed processing silos. This is a core problem with traditional Enterprise Application Integration (EAI) implementations where potentially there would be a need for $n*(n-1)$ data transformations. The middleware used in these architectures must also orchestrate the processing which is hard-coded into the applications and which must be modified every time that business processes change. It is the continual addition of layer upon layer of this control logic – each layer being added to handle some new condition which has arisen or to allow reuse of code for a new function – that makes many legacy applications so very complicated.

2.1. Relationship of this SOA to the OneVA EA and other Architecture Documents and Guidelines

This SOA framework contains information related to material in the the FEA Service Component Reference Model (SRM⁵), Data Reference Model (DRM), Technology Reference Model (TRM) and Application-Capability Reference Model (ARM). The FEA SRM describes the services components provided by / performed by systems in VA. The FEA SRM represent services that might be shared across agencies and are not services as would be defined or developed as part of an SOA. A single FEA SRM service might be implemented as a series of SOA services and a single SOA service might be part of the implementation of the multiple SRM services.

The TRM identifies standards which technical solutions or products must adhere to as well as lists of products that are approved for use within VA. It is expected that any implementation efforts based on this SOA framework will be built using products that are in the TRM.

The DRM identifies the data that is used within VA and includes a significant amount of metadata related to the information that VA collects and stores. While the SOA will impact the FEA SRM and TRM, the DRM serves as a basis for and input to the SOA. The services that are established as a result of this SOA will be implemented to exchange information as specified by the DRM and Data Architecture. In addition to the DRM, there will be a VA Data Architecture that describes the data that VA collects and how it is managed. One component of the Data Architecture is the Enterprise Logical Data Model that identifies and defines the information that VA collects and serves as the basis for the syntactic semantic harmonization of data within VA, for standard messages between services within VA and for information sharing throughout VA.

In addition to the Data Architecture, there are a number of other components of the OneVA EA that are not included in the FEA reference models. First, there is the EAA which is the overall VA Application Architecture of which the SOA is a part. While the EAA describes the use of services, their role in application development, and their relationship to other elements of the architecture. The VA Security Reference Model describes the implementation of security mechanisms across the VA and the manner in which they are implemented. Both the manner in which the services are developed and deployed and the infrastructure on which they run will need to comply with Security Reference Model.

In addition to the EA, VA has a number of other documents and guidelines that control the design and development of systems. These include ProPath – that describes the process for developing systems, Project Management Accountability System (PMAS) – that describes the management of system development projects and a wide range of policies and directives, etc. Development and implementation of services under this SOA is expected to follow and be consistent with all such guidance. Specifically all implementations as part of this SOA will need to be based on existing security architectures, requirements, and mechanisms that exist within VA.

⁵ While the FEA calls the Service Component Reference Model the SRM, the VA OneVA uses the term SRM to refer to the OneVA Security Reference Model and includes the service component information in the Business Reference Model (BRM)

2.2. Relationship of Services to the VA Systems Development Life Cycle

The VA Systems Development Life Cycle (SDLC) is governed by PMAS and ProPath and describes the processes and governance for the development of applications within VA. The development of applications based on the use of services and the development of services themselves will need to conform to the all VA SDLC guidance.

The OIT Enterprise IT Principles and other VA SDLC guidance mandate the use of agile development methodologies which result in the deployment of application at least once every six months and preferably more frequently than that. The use of services is ideally suited to agile development methodologies because the services are themselves small discrete units of functionality that exists as black boxes – where the inputs and outputs are visible and the definition of the transformation from the inputs to the outputs is known, but the details of how that transformation is performed is hidden. Because services are small units of functionality and are self-contained, they are well suited to agile methods which emphasize the very frequent delivery of independent releases.

2.3. Service Support for Business Process Management

Because in this SOA business process control has been externalized from the services, external Business Process Management tools (e.g., workflow engines or Business Process Orchestration tools) may be used to control process and workflow. When such tools are used, the business process can be specified in a standard manner using the Business Process Execution Language (BPEL) – a Web services based standard for specifying business process flows. Many tools exist which can perform process orchestrations written in BPEL – i.e., control the flow of control among a group of SOA services to accomplish a business function. Because only the inputs and outputs to a service are visible to the consumer, VA could change BPEL engines without any noticeable impact and multiple Enterprise Service Buses (ESBs) could be implemented using different BPEL engines.

While the formal use of BPEL would require the use of the full Web services stack, this is not recommended for high volume and relatively static workflows. Many of the BPEL enabled products are able to allow input of process workflows / orchestrations in the BPEL language while allowing the underlying services to be based on protocols other than those specified by the Web services stack. For example, a series of Java programs communicating via Java Message Service (JMS) messages.

While the intention would be to use Commercial off the Shelf (COTS) tools to support process orchestration, it is possible to orchestrate services without the use of specialized tools or specifying execution sequences in BPEL. In some instances the process flow will be exceedingly simple and be executed a very large number of times. In these instances, where the performance requirements would preclude the use of sophisticated tools, such as Web services-based process control, and where the sophisticated capabilities of the tools provide limited added value, the control logic used to determine service execution sequences and / or the conditions under which specific services or groups of services are executed may be controlled by a custom developed programs, scripts in any of a number of scripting languages, or through the use of automated tools that do not require that all services be expressed fully as Web services. Because these programs only contain control logic and not the services that perform the actual functional work, they can be quite small. These programs should be kept simple and should perform a single function lest they become as large and as cumbersome as the monolithic programs that they are replacing.

2.4. Service Centricity

An SOA replaces the traditional integration-centric mentality with a service-centric mentality. In this view, the services form the crux of the infrastructure. Because the services are discrete units of functionality and contain minimal flow or control logic – because the process flow (workflow) has been externalized and the only flow or control logic required is the flow and control logic internal to the service – the business analyst can compose the discrete services to support business processes and thereby manage business logic in a declarative fashion using a process choreography or workflow tool. These tools allow the business analyst to combine existing services, manual processes, and newly defined processes into a processing stream to support a business process. The definitions for the newly defined processes can then be provided to programmers for implementation. Using this paradigm, all of the control logic governing the order in which services are performed and the circumstances under which are governed is in the workflow engine or business process choreography tool, and therefore, is not internal to the service. This allows reuse of services without the intertwining of business logic for different functions that is common in most legacy applications.

Business process choreography deals with the control and sequencing of processes in a workflow, a workflow designed to accomplish some business function. An SOA can be used to support business workflows and to tie services to business functions, which ensures that there will be a business owner for each service and that the services support the business needs. The SOA does this by providing a basis for the definition of services that are tied to the business needs by providing the services tied to the FEA SRM services based on the OneVA Business Architecture. Each atomic service will have an owner. Each time a set of services is choreographed or orchestrated a new service is created which will have an owner. That service can then be included in further orchestrations. Further, since OMB approves budget requests and measures effective use of that funding through the Performance Reference Model (PRM), Business Reference Model (BRM) and these reference models are tied to the business architecture, tying the SOA service to the BRM and business architecture makes it easier to provide performance measures and document business improvements obtained through the SOA and therefore, to ensure OMB continues to fund the efforts.

The technical attributes of the technology underlying these functional services (making sure they are available, scalable, and otherwise meet the service-level agreements and other policies that apply to them) can be managed separately from their functional and business process attributes. The act of integration in this world-view moves to the process level, where business analysts and business process architects connect services to one another as they compose processes. The technical infrastructure that enables this composition to take place therefore no longer centers on integration in the sense of connecting systems or applications together, but rather on building and supporting the services that the business needs.

It is the goal of this SOA that all modernized applications will be implemented as a collection of services and that services are not just used to share information between applications or between agencies. Service interfaces, also known as a service facade, will need to be developed for legacy systems to allow the functionality inherent in the legacy applications to be accessed as services. In the context of this SOA, an application is a collection of software services that automates a business function. The approach for collecting and grouping these software services depends on the specifics of the business functions being automated and the relationships and interfaces between those functions.

2.4.1. Services and COTS Software

It is recognized that many modernized applications will be implemented either in whole or in part using COTS products. VA encourages the use of COTS for business application development where feasible. Virtually all of the major vendors of business oriented, functional COTS are migrating their products to SOAs, or at least exposing their products' functionality as services either instead of or in addition to more traditional APIs. While the goal is to move to a full SOA where all modernized applications are deployed as services, the extent to which COTS functionality is or can be exposed, as services will be dependent upon the degree to which the COTS vendors move their products to SOAs. There is no intention that COTS software be modified to conform to the SOA. If a COTS package does not expose functionality as services, then a service facade may be placed in front of the COTS application to expose its functionality as a set of services for external consumption, just as is done for legacy mainframe systems.

COTS products are also used to provide infrastructure capabilities. In some instances these infrastructure capabilities will be exposed as services (e.g., authentication services) and in others the COTS products will not be exposed as services but provide the basis for the services.

2.4.2. Building Applications through Services

Applications are to be based on a loosely coupled, autonomous, services oriented processing model. Atomic services are defined as "small", discrete programs that do a "small" piece of discrete work, which then may be composited into larger services to provide significant functional capabilities. The work may be a small segment of business process functionality, a data interchange service, a component of the infrastructure, or other such discrete capabilities that may be used by multiple applications or users. Services are either atomic services or composite services. Composite services are orchestrations that include at least two services, each of which may be atomic or composite services. Services can be combined with control logic to build applications. The control logic needed to combine atomic services into composite services and atomic or composite services into applications is contained in the orchestration logic external to the functional services. Services are designed to be used "on demand" by many applications.

Applications will consist of a set of loosely coupled services that will be used both internally and exposed to users external to the application through a number of mechanisms including a secure messaging service which will be mediated by an Enterprise Service Bus (see section 3.7 Enterprise Service Bus). Services internal to an application can be invoked through an Application Program Interface (API) or through a message-based interface, or can be loosely coupled and be accessed as a full Web service.

Services allow developers to focus on application specific business and presentation logic without having to worry about the mechanisms by which the services are provided. The service hides the complexity of the infrastructure and allows incompatible technologies to co-exist and cooperate. The limited access points provided by a service allow for tightly controlled security. Each service is documented, tested, and certified by security once. However, privacy compliance reviews are conducted each time a service is reused to ensure privacy compliance requirements are met across the

full data and data usage life cycle. Services can be upgraded⁶ and scaled independently because of this loose coupling.

2.4.3. Communications between Service-based Applications

The intent is for communication between applications to be message-based using standard syntactically and semantically harmonized messages based on the entities in the Enterprise Logical data Model (ELDM). Whether a messaged based mechanism is used between services within an application (e.g., between subsystems or between a front-end (presentation layer) service and a back-end functional service) would be a design issue to be decided by the application designers. An HTTP feed through the ESB from a presentation layer service to a back-end service would be allowed if the application did not require the additional services provided by a messaging service – such as guaranteed delivery. Point-to-point communication with services external to an application other than through an ESB should be discouraged in all but the most extreme circumstances.

The requirement for communication through the ESB does not extend to communication between internal components of a COTS application or between a COTS application and its proprietary data store. COTS applications are not required to replace direct API or SQL calls to databases maintained only for those COTS applications. COTS applications will communicate with other portions of the enterprise through the messaging service and will access services that exist in the enterprise in the same manner as custom developed applications. COTS applications are required to be able to be called as a service by other applications in the enterprise. This requirement can be met by placing an adapter between the COTS application and the messaging service.

Services external to an application communicate and cooperate via the network by calling other services through a message-based interface. While messages between internal applications are expected to be based on standard VA formatted, semantically harmonized messages; messages to external applications may be implemented using existing or industry standard message formats (e.g., Health Level Seven [HL7] or National Information Exchange Model [NIEM] etc.)⁷. In addition to the syntax and semantics of the messages – for both internal and external messages, there will be a transport mechanism that will serve as the basis for transmitting the message. The message transports include, but are not limited to:

- COTS messaging product messages.
- Simple Object Access Protocol (SOAP).
- HTTP / HTTPS.

⁶ Although services can be upgraded independently, it may still be necessary to maintain multiple versions of services since it is likely that an application would want to be tested with the new services prior to their use. It will usually not be possible to upgrade all applications or parts of the infrastructure at the same time. Further, as services are upgraded to provide new capabilities, some systems may still require the older version.

⁷ While VA may specify the syntax and semantics of messages within VA, VA has no authority or ability to dictate message formats used by organizations external to VA. Therefore, this SOA is based on the assumption that the current format, semantics, and transport for external messages must continue to be supported.

The only access to services external to an application is through the message interface. Messages are routed to the requested service through an Enterprise Services Bus that is part of the SOA infrastructure. The internal details of the service must be hidden from users of the service so that the internal structure of the service, the platform on which the service is hosted, or the sources of data for the service can change without impacting any applications calling the service.

2.4.4. Syntactic and Semantic Harmonization

This SOA is based on applications being composed of fine grained reusable services that communicate with one another through the use standard messages used. For this approach to work each service must interpret the data in the same manner; it must have the same meaning for all services that use it. This SOA is based on the use of semantically and syntactically harmonized messages. While data elements in standard messages will be semantically and syntactically harmonized, their formats will be allowed to vary as format conversion will be performed by the ESB.

2.4.4.1. Semantic Harmonization

Semantic harmonization of data ensures that data elements with the same name have the same meaning and that data elements that have different names have different meanings. As systems choose data element names there is a high probability that the same name will be used for different quantities. Systems that use the same name for different quantities cannot communicate unless and until they can agree on the meaning of the data that they exchange.

For example, one may ask how many patients are on a floor of a particular VAMC. Admitting might respond with the number of person admitted and assigned to the floor, whether they had reached the floor or not; the nurses might respond with the number of people physically present and not count those that are off the floor for tests or surgery; housekeeping and food service might give yet a different answer. It is not that any of these answers is wrong, they are just answers to different questions and if data is to be shared across the enterprise, each must be represented by data elements with different names.

2.4.4.2. Syntactic Harmonization

Syntactic harmonization refers to the form of the data. Data must not just have the same meaning, but the meta data related to it must also match. Are names sent as <FIRST, MI, LAST> or <LAST, FIRST, MI> etc. To be able to share data, the data validation rules must be the same across the enterprise so that data that is entered into one system will not fail validation checks in another. In addition, other aspects of the data elements must match, such as allowable values and validation rules must also match

Since the OIT VA Enterprise Management Principles and Application Architecture require that there be a single authoritative instance for all data and no updates of copies of the authoritative instance are valid until the authoritative instance is updated. But, what this also required is that once one system accepts a data update it will pass all data edits and validations for the authoritative instance of the data. But the data once validated is validated, it must be able to pass the data validations of all systems that use that data.

2.4.4.3. Format Harmonization

Format refers to the physical form of the data – i.e., whether a number is represented in binary format, as ASCII character representing numbers, or an XML string with the value (i.e., <VARIABLE

NAME = “data string”>. Depending upon the nature of a service, the format of the messages may vary. Web services will use SOAP over HTTP for message transport which will require XML based messages, communications between VA systems may be through a commercial messaging product and be for example JMS over IBM WebSphere MQ, or between closely related services messages may just be binary data sent over a TCP/IP link. The ESB will be able to perform message transport and message format transformations, but is the ESB is not used for communication between services than both sender and receiver will need to use the same transport and message format.

2.4.5. Synchronous vs. Asynchronous Services

The preferred design approach for services is that they be designed to operate asynchronously and to use asynchronous communication between services. However, use of asynchronous communication is not always desirable, or in some cases even possible, so use of synchronous will be common and in some instances required. Therefore, use of synchronous communication will not be an exception, but a common practice.

The asynchronicity of services serves to sever direct connections between the service consumer and the service provider. With asynchronous communication, the service consumer makes a service request to the service provider and does not wait for the response to the service request before proceeding onto other work. At some later point in time, once the service provider responds to the service request, the service consumer accepts the response. The service consumer does not necessarily interrupt any other work to process responses, nor does the service provider lock any database records on the expectation that the service consumer will make any further service requests.

This is very different from a transaction-based processing paradigm in which a consumer would read and lock a record and then come back to update the record – e.g., read a bank balance, subtract funds from the balance, and then update the balance. In a transaction-based system, it would be considered a disaster if the record (the customer balance) was updated between the time it was read and updated. In an SOA, this is allowed to happen whenever the services are provided asynchronously. Because the process is asynchronous, changes to the underlying data can be made between the time the service response is generated and the time that it reaches the service consumer. This may or may not be a problem depending upon the nature of the service being requested and nature of the business process being performed.

With transaction-oriented systems, locking of a database can be tolerated as the transaction duration will usually be less than a second and even then, locks can be a severe performance problem. With services, where a process may be performed over a long period of time – hours to days – and in which there may be several human interactions, such locks would be disastrous.

The difference between the asynchronous and synchronous service calls is similar to the difference between asking for a price for a product – in which the buyer expects that the quoted price will be honored for a period of time – and asking for a price on a stock – in which the customer is quoted an instantaneous price that can be expected to change continually. Services defined in an SOA are normally expected to operate asynchronously, but this is not always possible.

Places where asynchronous services are not appropriate include some types of online access to a system or when an application needs to update information held by another and must do so using the most current value of the information. Security services are another area in which asynchronous service calls are not appropriate. If a service call is made to an authentication service, a response will be required before a process will be allowed to proceed.

Although systems should be built to be autonomous using asynchronous messaging, in many cases, this does not make business sense. In some cases a service request must be answered immediately or the response is of little use. If that is the case, the requesting process will not be able to wait for a response before proceeding, and may determine that no response will be provided in a timely manner and proceed without the response. Therefore, the ability to communicate asynchronously will be of little importance since the business process will discard the results if they are not returned immediately.

Other exceptions occur when an application seeks to update data held by another application. In such cases, the application must be able to respond to a failure of the remote service to perform the update; thus, it may be necessary for the application requesting the update to wait to ensure that the update completes successfully before continuing.

Any service request that requires a timely response will use synchronous communications. This includes all service requests that involve a human waiting on a response or a service that is requested by an application that must get a response before it can proceed.

There are two basic approaches for Interprocess communication – a Remote Procedure Call⁸ (RPC) based approach or a message-based approach. While RPC-like systems can be implemented using a message paradigm, the preference is to use a pure message-based approach to preserve the autonomy and loose coupling of the services. Synchronous messaging almost forces the system into an RPC-like paradigm rather than the preferred message-based paradigm. Therefore, the choice of synchronous vs. asynchronous communication is also a choice between an RPC-like and a message-like systems paradigm.

2.4.6. Autonomy vs. Interoperability

An SOA is a set of architectural principles that define how relatively autonomous systems or components interoperate. Autonomous and interoperate are opposing ideas. An SOA follows a set of principles that specifically deal with the tension between two contrasting ideas: autonomy and interoperability.

Software systems that are autonomous have these characteristics:

- They are **CREATED** independently of each other.
- They are **BUILT** by well-defined groups.
- They **WORK** independently of outside systems.

⁸ A Remote Procedure Call (RPC) is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. That is, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remote. When the software in question uses object-oriented principles, RPC is called remote invocation or Remote Method Invocation (RMI).

- They provide **SELF CONTAINED** functionality. The functionality that they provide would be useful even if it were not associated with any higher-level systems.
- They are **LOOSELY COUPLED**. The services and systems operate independently and communicate through a standard set of mechanisms that are independent of either service or system.

The second key word is “interoperate,” which means to cooperate, to work together, and to interact in a meaningful way. An SOA provides a blueprint for how autonomous systems, under certain well-defined circumstances, interoperate. Key principles of interoperating autonomous systems are:

- **Asynchronous Communications** – When communicating asynchronously neither system interrupts its own work to wait for some other system to complete its task. Asynchronous communications are the preferred method of communications, but asynchronous communications is not always possible.
- **Availability** – Synchronous Services must be available to consumers at the time that they are requested (i.e. the service must be up and running).
- **Heterogeneous Communications Channels** – If two autonomous systems interoperate, then there must be a communications channel between the two that is independent of the technology used by either system.
- **Proof of Identity** – If two systems are truly autonomous of each other, then there is a natural healthy mutual suspicion that must be maintained. In order to get beyond this, there must be a way for each system to convince the other that it is really the system that it claims to be.
- **Error Management across the Enterprise** – If services interoperate and cooperate to perform work across the enterprise there must be error-handling mechanisms that can handle errors across multiple services.
- **Workflow Coordination** – When multiple autonomous systems are communicating with each other asynchronously, keeping track of the progress of a transaction in the overall workflow effort is much more complicated than when systems are working together synchronously.
- **Audit and Logging across the Enterprise** – If services interoperate and cooperate to perform work across the enterprise there must be auditing mechanisms that can handle process flows across multiple services and can provide an end-to-end audit trail of all services accessed across the business process.
- **Statelessness** – Services should not be required to remember anything about a request after it executes that request.

2.4.7. Event Processing

Complex event processing can be performed by specialized COTS software packages, but it can also be performed directly through the capabilities described in this SOA. Since SOA services implement all or part of a business process an event is either the result of a service execution or the failure of a service execution. The completion of a service, the failure of a service, the partial completion or blockage of a service, or a service returning a specific result all might be used to generate an event notification. Listener services would use the ESB Publish / Subscribe (Pub / Sub) (see Section

3.7.3.1) mechanism to subscribe to events generated by other services and be able to take action based on the sequence of events that have occurred. This approach would perform event processing through the native capabilities of the SOA including the Pub / Sub messaging capabilities of the ESB and the BPM capabilities of the BPEL processing engine. Further event processing analysis would then be provided by the Business Activity Management (BAM) software (see Section 3.7.3.2)

2.5. Types of Services

Services are implemented to support the performance of business processes. The OneVA EA captures the business processes that VA needs to perform and, potentially, the services that are required to implement those business processes. But, services do more than just directly implement business functions.

The SOA will contain three types of services:

- **Functional** – Relate to the performing the functional work of VA. Functional Services are choreographed into workflows that perform business processes. These services are written, or provided by service developers and perform a specific business function. Certification & Accreditations (C&As) for functional services will be performed by the service developer or will be included in the C&A and Authority to Operate (ATO) of the application of which they are a part.
- **Data** – Access data stores and make data available to applications. They will normally be included in the C&A of the application that houses and maintains the authoritative instance of the data being provided.
- **Infrastructure** – Provide infrastructure support to be used by all applications. The C&A for these services is performed as part of the C&A of the infrastructure.

Because functional services perform business processes, they must be tied to the business processes that they implement and thus, must map to the BRM, the Business Architecture, and to the Application Portfolio. Just as business processes have business owners, so too must functional services have business owners who will be responsible for defining the business requirements that the service supports and the constraints (e.g., security, privacy, and access requirements) under which it must operate. The data services provide access to VA data stored in one of the applications. Thus, data services must be able to be mapped to VA data stores and thus, must be mapped to the DRM and Data Architecture. Infrastructure services provide basic (SOA) infrastructure capabilities and provide support to the functional and data services and the applications. The infrastructure services must map to the Technical Architecture and the Application Architecture.

3. SOA Implementation Considerations

The implementation of the SOA will require the development of both services and the infrastructure to support the operation of the services. This section deals with some aspects of the implementation of both the services and the infrastructure.

3.1. Use of Standard, Syntactically, and Semantically Harmonized Data Definitions

In this SOA, data transferred as part of a service is in a standard form using enterprise standard data definitions – semantically and syntactically harmonized data definitions. These enterprise standard data definitions can be used natively by the applications or through translation of native application data definitions performed by an adapter developed by the applications. This means that the ESB can

be developed independently of the data transformations and thus can be scalable without becoming ever more complex because of the need to accommodate ever more data transformations. This also means that the ESB developers need not know or understand the data formats used by each of the applications. Application data transformations are developed by those who know the application data formats the best, the application developers.

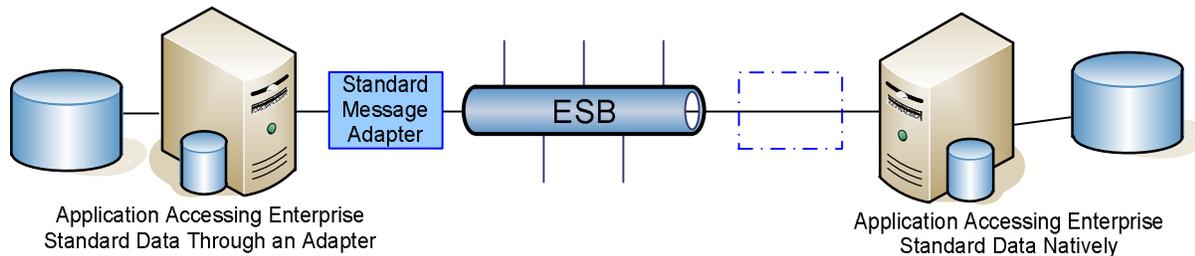


Figure 1 - Data Adapter to Translate from a Native Application Format

VA has decided on the use of an internal standard message format rather than a standard such as HL7 or NIEM because:

1. No single externally defined message set covers the full range of information that VA requires.
2. Multiple such standards exist and are required by various external organizations and various portions of the VA enterprise.
3. The multiple external standards are inherently inconsistent between themselves and cannot be harmonized.
4. Obtaining a semantically harmonized message set requires a single standard set of data definitions be used throughout VA that is different from any of the external standards.

The standard semantically harmonized provides a common set of data definitions across the broad functional areas within VA.

Moving to an internal information interchange standard is preferred rather than the use of other standards, even international standards, because pushing a VA specific standard provides a standard that in time all VA software can adhere to. This will provide VA the ability to have harmonized data across VA, and data will have the same definitions, semantics, and metadata regardless of where in VA that the data values are generated. International standards are important and do have a strong business case for use where they have been adopted by the major vendors or by VA's external customers. However, for a defined service usage, VA really does want to push harmonization. While the VA standard may adopt data definitions from external standards where they are not conflicting, syntactic and semantic harmonization are far more important than use of the external standard data definitions.

The SOA requires that the payloads for all service requests and service responses (i.e., message sets) conform to VA standard data definitions.

The SOA is based on a series of services that are a composited to support business processes. An SOA reduces the quantity of glue needed dramatically and basically replaces it with the architecture itself. One of the ways in which an SOA reduces the need for glue is by not allowing the transfer of

information using non-standard data definitions⁹. It is the lack of a standard set of agreed to data element definitions that has always been the Achilles heel of systems integration since the 1970s and has been the cause of the requirement for such heavy-duty middleware. Explicitly recognizing that the use of common data definitions is a major issue – implicit in an SOA – and solving this problem, rather than masking it with complex middleware, is what enables SOAs to succeed where other integration approaches have failed before it.

Architecture therefore plays a much more active role in an SOA than before, moving from being solely a design time activity into the runtime, providing the discipline needed to move integration, as well as business logic, into the hands of the business. In an SOA, workflow is kept separated from the services. Much of the work related to business process control is externalized from the services, thus simplifying the services and the need for complex coding required to “glue” components together and allowing one set of code to support multiple workflows. This allows the business processes to be rapidly composed and changed without one set of code overlaying and commingling with another. It is the transfer of data using standard definitions and the extraction of most of the process flow logic from the functional code that makes an SOA approach attractive.

3.2. Extensible Markup Language (XML) as a Wire Format

XML is the format de rigueur for communications with Web services and all Web services are based on the use of, or assume the use of, XML. As noted earlier, services in the sense of an SOA are not Web services, and thus there will be no requirement to use XML for transmission of information between services, particularly between services provided by closely related applications. Further, use of the VA standard does not require use of XML for the message transport although it does require that the message format be defined in XML in an XSD. XML is the preferred “wire format” for the messages sent between services where either the service provider or service consumer is outside of VA. XML formatted messages can be used within VA or even between related applications, but in those cases tradeoffs need to be made between the additional flexibility that XML provides and the performance impacts of its use. The overhead required to format, unformat and reformat character based XML messages make use of XML impractical for some high volume, high performance applications. Therefore, binary, open (e.g., JMS), or proprietary message formats may be used between applications when performance or resource limitations make use of XML infeasible or undesirable.

Where there are current standard message formats (e.g., HL7), these may continue to be used within an application. Adapters can be placed between each application and the message service to convert internal application formats to the enterprise standard message formats if the applications do not natively support the standard data definitions and message formats. The standard message adapters can be co-located with – or even on the same device as – the adapters needed to convert from enterprise data formats to native application data formats. The ESB will have the ability to provide

⁹ Clearly the need to use standard data definitions for data transfers only applies to transfers within VA. VA interfaces with a large number of external organizations where the interface is defined by federal law, inter-agency agreements, business agreements or standards or simply the fact that tens of thousands of organizations use the existing interface. External interfaces will need to be held inviolate and conversion to VA standard data definitions will need to occur within VA.

transformations between all supported message transport protocols. The decisions as to whether to use XML as the message format, or whether to use other formats cannot be made in a vacuum but must be based on business requirements and enterprise wide concerns.

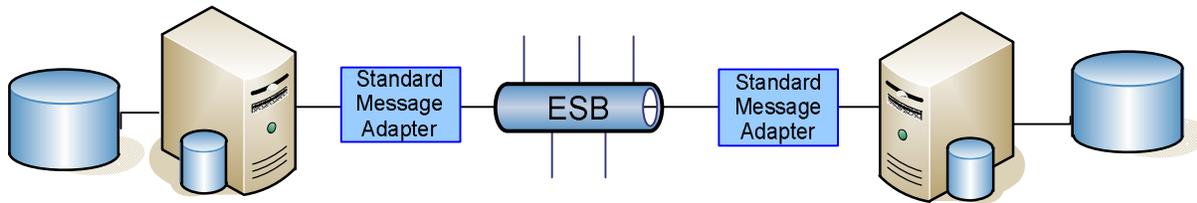


Figure 2 - Placement of XML Adapters

Web services are services that expose services over the web and use a specific set of protocols including SOAP, Web Service Definition Language (WSDL), Universal Description, Discovery, and Integration (UDDI), and XML etc.; they are designed to allow people and systems to consume services from providers unknown to them. What makes it possible to avoid the heavy burden of these “Web services” standards is that within VA, the users and applications are well known to one another and there is a level of trust between them. The use of XML and higher-level protocols promote interoperability between consumers and providers. Since XML is self-describing, it allows communications and interoperability between consumer and provider without prior negotiation of data formats or data syntax. However, there is still a need for agreement on the semantics of the data being exchanged. In the VA SOA the semantics – the data element definitions – are provided by the standard, semantically harmonized messages. In addition the syntax of the messages is agreed to because 1) the systems are closely related and 2) the data elements that are the basis for the messages have been syntactically harmonized.

However, since SOA denotes an architectural philosophy rather than a set of standards, many of the benefits of an SOA can be obtained through higher performance transports and mechanisms than through “Web services.” Further, many of the benefits, such as improved interoperability, that are only available through the full “Web services” protocol stack may be of minimal utility within an application or even between VA applications. There is also significant concern that the heavy protocol stack used for “Web services” is less efficient in support of very high volume transaction processing environments. The trade-off between the use of Web services and native messaging is a trade-off between interoperability and generality and performance. Production organizations with very large numbers of service accesses coming from internal users, and stringent performance requirements – bordering on real time requirements, readily sacrifice some interoperability (which can be provided through data harmonization) for performance. Approaches (e.g., J2EE RMI etc.) rather than the full Web services protocol stack would be more beneficial.

As will be discussed in Section 7.3 Data Sharing, there are specific recommendations as to the mechanisms that should be used for transferring data, and by extension, accessing services. The presumption is that access to services external to VA will be through Web services, access to services within an application would be message-based, and between applications would be message-based on service request volumes and performance requirements. For mission critical systems with severe performance constraints, the presumption will be that the services will be implemented using the highest performance mechanisms possible. These services should be exposed as Web services that

can be accessed through the ESB as well as native message services approach that can meet the performance constraints.

3.3. Need for Services to be Independent of Legacy Systems

The functional services must be tied to core business functions not to legacy systems (e.g., GET PATIENT DATA not GET VistA Data). Legacy systems will be modernized and evolve over time to support changing business processes. Services should be long lived and should transcend changes in the underlying systems. By aligning the functional services with the business processes rather than with the current systems, they become long lived and possess the ability to continue even in the event of major systems changes. Although the information required to support a service may be located in multiple databases within a system or databases contained in multiple systems today, all of that data might someday be in a single integrated database; but regardless of whether the data eventually migrates to a single database, all such data could be provided by a single service.

One goal in moving towards an SOA is to allow the application systems and databases to be updated, merged, and / or rationalized. This can come through the refactoring of current systems. Refactoring refers to modifying source code without changing its external behavior. It is expected that the changes to the legacy applications and their underlying databases will occur in a manner transparent to the front end systems that use the data and that interface with those systems. This transparency during the rationalization of backend databases and systems can only occur when the services are aligned with the business processes and the functional requirements that support those business processes, rather than current IT systems. In addition to providing access to legacy system databases, the user interfaces of the legacy systems must also be decoupled from the back-end components of the system and exposed as a set of services.

3.4. Need to Tie SOA to Technology-Based Architectures

The SOA will be implemented by and be used in the applications that are to be developed. They will be used to access data and will require a certain amount of common infrastructure to be developed to allow their use. For example, when one application requests data – such as the name of a Veteran or his / her address – and another application provides that data, they must agree on what constitutes a valid name or address as well as the form in which the address will be transferred. The development of enterprise-wide agreements on data element definitions and the metadata associated with data elements occurs during the development of the ELDM. The ELDM is an integral component of the Data Architecture. Data access rules, who owns the authoritative instance of a data element, how data will be distributed to systems, etc. are all addressed in the Data Architecture. Although a complete VA ELDM is not yet available, it will be developed over time, with particular emphasis on those areas in which systems are being modernized.

The ELDM will identify a set of data management functions that will be used to manage all data modifications (insert, update, delete). The Applications Architecture will include a layer of reusable services designed to implement all data access and modification functions and to provide applications data as specified in the ELDM. All modernized custom built services that access or modify data will call these common reusable services.

This SOA will need to form the basis for all further application development and will need to be incorporated into all appropriate application design patterns. The Application Architecture provides the guidelines for the development of the applications and the use of services. Design patterns for

standard classes of services will need to be developed and published for use throughout VA's development community.

3.5. Nature of Services to be Provided

Like many terms that have gained wide usage in the IT arena, services, as that term is used in SOA, could have a number of meanings depending upon the predilections of the person using it. A key decision to be made is what types of services will be provided. To many, the term SOA implies the use of "Web services" that expose services over the web and use a specific set of protocols including SOAP, WSDL, UDDI, and XML etc. Others have implemented SOAs solely using XML and a transport, such as a COTS messaging service, and which do not allow external web access. Some authors define services strictly in terms of functional or business oriented services that provide a core piece of business functionality and virtually require that the services be assembled using some type "business process orchestration tool," while others would include purely technical, or infrastructure, services such as a "file service," "print service," or "data interchange service" or "data access service."

Perhaps in part because of the confusion with "Web services," there is a perception that services can only be functionally oriented and not technically oriented. An SOA can include services of a technical nature. Just as there are a large number of business functions that are used repeatedly by applications and make good candidates for services, there are numerous technical services that are used repeatedly. Good applications programmers who fully understand the business processes are not usually good at developing the technical infrastructure code needed to efficiently implement those functions. Separating out that infrastructure code by using technical services will greatly improve application implementations, because application developers with business expertise can concentrate on designing and coding the business logic. Technical services can be defined to limit the amount of work that applications developers need to perform.

Good examples of technical services can be found in the data access area. Forcing applications to access data stores through a set of data access services provides numerous advantages. Severing the ties between the business logic and the data access logic allows the data access logic to be optimized separately from the business logic. Further, this separation allows database structures to be changed or databases to be merged without impacting the business logic. Other examples of good candidates for technically oriented services include services for building / parsing XML strings, sending messages, authenticating and authorizing users, populating object instances with data, storing results of an operation in multiple databases, etc. Infrastructure services are like other services in that while the inputs, the outputs and the rules for transforming the inputs to outputs are specified, the technical details of how the transformation is performed is necessarily hidden. Therefore, limiting application developers to implementing business logic allows the underlying data infrastructure to be changed without disturbing the application. The use of such technically oriented services allows the database structures to be changed without requiring application changes.

In theory, one could limit services to functional services and not include technical services in the architecture. However, including technical services in the architecture allows for a cleaner separation between business logic and data and systems logic and allows the former to be optimized independent of the later, allowing application programmers to concentrate on the business logic without needing to concern themselves with the need to understand how the data is organized or where it is stored or the details of the infrastructure supports those services. This allows the multiple underlying databases to be modernized, extended, and rationalized without impacting the applications. Changes to the data

base structure become transparent to the applications. When technical services are used to implement non-business logic functionality, these services are pushed to the general infrastructure, which can be optimized independent of the applications. Additionally, in practice, well-defined technical services often experience the highest reuse rate in an SOA. For example, a technical service that implements a security function may be called for every incoming and outgoing message.

The SOA should be specified in a manner that will allow Web services but which does not require the use of the full Web services protocol stack for high performance internal services. Internal services should be implemented as efficiently and as straight-forwardly as possible. Both functional and technical services should be allowed to allow for an efficient, common infrastructure to be developed for modernized applications.

3.6. **Architecture Tiers / Service Layers / Service Types**

Section 2.4.7 stated that there would be three (3) types of services:

- Functional.
- Data.
- Infrastructure.

The VA SOA requires use of a multi-tier architecture to deal with both architectural issues and security issues – such as the need to separate outside users from internal systems. The layers required by the SOA are the:

- Interface¹⁰ Layer.
- Business Logic Layer.
- Data Layer.
- ESB¹¹ and Supporting Infrastructure Layer.

The three types of services will be spread across the four layers in the architecture.

¹⁰ In much of the rest of the document the layers are described as Presentation Logic, Business Logic, and Data Logic. The Presentation Logic includes the logic required to display application information on any type of user device. In this section, the more general term “Interface Layer” is used to recognize that not all service (application) interfaces are with people, but may be with devices or external systems. Services to interface with devices or external systems are discussed in this section, but they tend to be developed by more infrastructure oriented personnel rather than application developers. Although device interfaces, external systems interfaces, and user interfaces are treated as a single layer from an SOA perspective, each is contained in a different layer of the Application Architecture.

¹¹ This is not a layer that developers use, as it provides the infrastructure on which the SOA is run. This is represented by a very much different layer in the application architecture than the SOA Services (business and presentation logic) or Data Services.

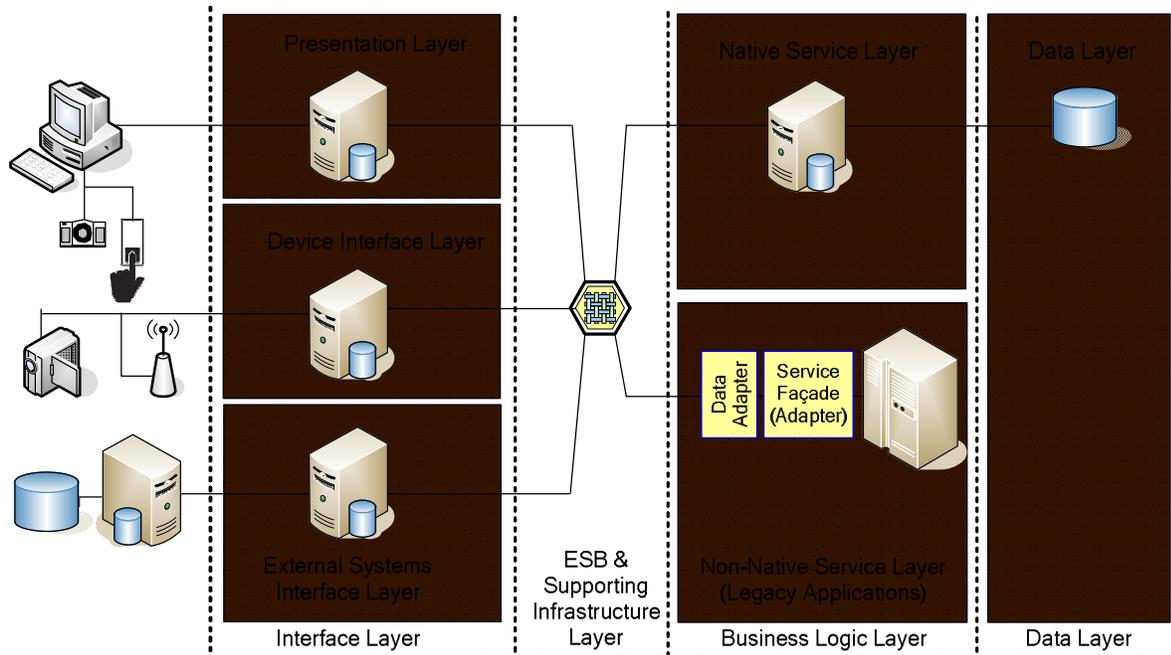


Figure 3 - Architectural Layers

Figure 3 shows the interconnection between the various levels in the multi-tier VA SOA architecture. Functional services will be found in the Interface and Business Logic Layers. The Data services will be found in the Data Layer, and the Infrastructure services will provide the connections between layers and provide support for each layer. Since system interfaces, device interfaces, and legacy wrappers are all developed for individual services / applications they will be part of the business logic layer. The Infrastructure services will provide capabilities such as authorization or logging.

The diagram provided as Figure 3 is a conceptual level diagram discussing conceptual layers and types of capabilities and is not intended to denote the actual implementation, which may be quite different and much more complicated due to requirements for firewalls, network segmentation, security requirements and guidelines, network engineering, and other components mandated by the EA and other VA directives, guidance, documents etc.. Further, the diagram is not intended to either require or prohibit the use of separate physical processors for individual layers.

The SOA tiered architecture does not include a tier for external systems, workstations, or devices, as these components would be clients for the services provided by the services built in conformance to this SOA, and thus, are outside the bounds of this SOA, and as such, they are beyond the purview of this SOA. While this SOA identifies specific protocols and transports for use between services covered by this SOA, it recognizes that interactions with external systems and devices will be required to use those protocols and transports that those external devices are able to use. There is one exception to this – the use of Web services as a mechanism for interactions with external entities. Web services can only be accessed using the specific Web services standards and protocols.

This SOA will need to account for the multiple tiers of the application environment and deal with the security concerns and issues as they relate to that environment. Services will need to be defined at each of the levels to perform the work unique to that environment. There will be a need to ensure that services are secure so that they can be certified and accredited and can handle the range of requests

from the diverse population that may make requests. This will require some limitations on the way services are accessed and who can access them.

3.6.1. Interface Layer

The Interface Layer will handle all communications between systems and users through a Graphical User Interface (GUI) (e.g., a browser) with all devices and with all system-to-system communication with systems outside of VA. When interfacing with end user devices, the External Interface Layer is normally called the presentation layer.

Services at the External Interface Layer are the only services that are allowed any interactions with external users (end-users and new system-to-system interfaces¹²). This means that all service requests from sources external to VA must pass through a VA developed service at the External Interface Layer. This provides:

- A single point through which all external requests must flow.
- A single means to ensure the authentication of all service requests without requiring full authentication by each service in the other levels.
- A single place where any virus checking or other preliminary checking of input may be performed.
- A point at which VA may assign an identity and a role to each request.

Further, the authentication of external users should be performed transparently to the user and be persistent for the duration of the session.

This layer will encompass constructs such as existing portals that provide a multi-firewall construct to separate external users from internal systems. This will provide a significant amount of security through a defense in depth construct that will allow external access to systems handling sensitive data.

3.6.1.1. Presentation Layer

Presentation layer services are to be used for all end-user interfaces with internal and external users. The distinction between internal users and external users is based on their manner of access to VA systems as to who they are. Internal users are users who access VA systems from inside the VA network or through the VA Virtual Private Network (VPN) while external users access VA systems through the internet.

- Presentation Layer Support for External Users

As used in this section, external user specifically refers to people and workstations external to trusted security zones (e.g., Veterans, persons at local hospitals etc.). Presentation services for external users will be browser or mobile device application based and will be limited to HTTP or HTTPS.

¹² Although current external interfaces are grandfathered from needing to comply with this requirement, as a practical matter most such system-to-system interfaces currently go through some layer where they are checked for validity (at least that they are well formed) and acknowledged.

A presentation layer service may only support users with one level of access or users in a single security group (e.g., Veterans or VAMC based physicians etc.). Existing application systems will be responsible for the support of their user populations until such time as those applications can be modernized with services based development. Once the external users communicate with the presentation layer service they can access the capabilities of the SOA through the application to which they are attached.

A key factor related to the design of this layer is to ensure that even were an external user to penetrate one or more services in the presentation layer, the processors in the business logic layer would still be protected. Where an application has implemented a portal to service external users that has already been certified and accredited by security, this portal should be used for interfaces that provide sensitive, Personally Identifiable Information (PII), or Protected Health Information (PHI) data to external users.

External users may access VBA information through a wide variety of personally owned devices and since they are personally owned, the VA will have no control over the software installed on those devices or the security of those devices. External user access may be provided through:

- **Web Browser** – Browser based access may be provided to external users through workstations, laptops, tablets, and smartphones etc. Web pages should have the same look and feel regardless of the device on which the information is displayed
- **Personal Electronic Devices (PEDs)** – Are user owned electronic devices that are able to run user acquired applications “apps” that may be used to display VA information to the extent that VA decides to author and distribute these apps. To be useful, these apps will need to access VA data on VA systems and therefore will need to be required to access VA data through the Presentation Layer services.
- **Kiosks** – Are VA owned devices that can be used to provide users, typically Veterans, can use to access VA data.

This approach is also necessary to allow there to be an underlying set of services at the business layer that perform processing using data more sensitive than can be released to the user. The service processing that data must be different from the service interacting with the user (e.g., a Veteran’s full medical record may need to be access to answer a query, but only some lessor amount of information may be presented to the user). Presentation layer services will only be allowed to access data at a single level of sensitivity.

Kiosks will provide a mechanism for Veterans to access VA data. While kiosks are VA owned and operated, and thus be under far more VA control than a user workstation, they will be used to provide access to information that would otherwise be accessible through user workstations, the kiosks would normally either be on the VA network or access the VA network through the VA VPN and be owned and controlled by the VA. Therefore, while technically they may seem more like internal devices, the information to be displayed on kiosks will be more like the information displayed on external devices, and thus, they are described both here and in the section on internal user access.

- **Presentation Layer Support for Internal Users**

As with the presentation layer for external users, the presentation layer for support for internal users will be provided almost exclusively by application portals and interfaces. Internal users are VA employees and contractors using VA supplied or VA approved devices accessed from the VA

network. Such end user devices will necessarily be limited in the applications that they run and the software that is used. These devices may include:

- **Workstations** – Interface with workstations will be browser based, but may include Java Applets, rich clients, and ActiveX controls¹³ for COTS software. Rich clients¹⁴ are an architectural direction that includes the download of code to the desktop and which supports HTTP communication with the servers where the basic processing is performed.
- **Attached Devices** – Internal users’ workstations may have attached devices that are used for data entry that are controlled by the workstation and which provide data within the context of a transaction (e.g., bar code readers or finger print readers etc.). The presentation layer services supporting internal users will also need to support these workstation-attached devices. Within the context of this discussion, devices (e.g., fingerprint readers or bar code readers) in kiosks would be considered to be internal components of the workstations.
- **Personal Electronic Devices (PEDs)** – PEDs are smart devices (e.g., smart phones and tablets) support local applications that access central servers in addition to web browsers that access traditional web sites. The web-based access will be similar to workstation access to the web sites, but web-sites may be customized to allow for the PED smaller screen size. PEDs may also include small applications “apps” specially developed for the PED which may access data on VA servers. The SOA Presentation Layer will need to support PED app access, but the PED apps are not part of the SOA.
- **Kiosks** – Are VA owned devices used to provide information to users, typically Veterans. While the information will be most similar to information provided to external users, the kiosks will typically be on the VA network or access the VA network through the VA VPN.

Internal users should be presented with a common look and feel across the range of applications that they use. This means that information should, to the extent feasible, have the same look and feel across the range of devices across which it may be accessed. This can be provided either by the use of a single portal or multiple portals with strong controls on format and look and feel. Support for these internal users likely will be a lighter weight portal than the highly secure portal that would be required for external users.

3.6.1.2. Device Interface Layer

There is one characteristic that makes the control of devices different than communicating with users or systems. These devices may operate asynchronously from workstation and the user. These

¹³ USGCB blocks end users from installing ActiveX controls. If the presentation components require ActiveX controls, they should be distributed using the central Desktop Management support.

¹⁴ Google Earth or the cloud based Office suites would be examples of rich client applications.

devices perform a “read” when they are used. The protocols to be used to communicate with these devices will depend upon the nature of the device.

- Internal Device Interface Layer

Some systems are required to control a wide range of devices from bar code scanners, cameras, and a myriad of other physical devices that operate asynchronously from the workstations. The Internal Device Interface Layer provides a capability to attach to devices other than workstations or external systems. Each of these devices is likely to have its own proprietary interface and protocol that will need to be used. If the device only attaches to a single system, i.e., the system that controls the device, it will not be necessary for this interface to be a service, as the device will be allowed to directly attached to the system.

- External Device Interface layer

This layer would be used to connect external devices to VA systems. The typical external devices would be special devices used to support non-TCP/IP communications links where special purpose devices are required to communicate over the link with a legacy system external to VA. To the extent that such links exist, it is unlikely that they can readily be changed, and thus they will need to be interfaced with. These special purpose devices would be separated from the rest of the environment through the use of the service facades so that enterprise services would see the special devices as standard services.

3.6.1.3. External System Interface Layer

External systems shall not be allowed direct access to VA internal functional services and will need to be processed through an interface layer that provides the security services provided by the presentation layer for end users. The External System Interface Layer supports system-to-system and automated (e.g., HL7, NIEM, COTS Messaging, Web services etc.) interfaces. For COTS messaging interfaces with external users, this level would be the queues to which the messages first arrive.

For systems that have a services interface (e.g., Web services) there will be a services interface in the External Interface Level. It would be preferable for these links to conform to a services approach, but because of the legacy nature of the links, some may need to be direct links governed by Interconnection Security Agreements (ISAs), Memorandums of Understanding (MOUs), and / or Interface Controls Documents (ICDs). Where the interface cannot be directly based, there will need to be a service facade for the interface to expose the interface as a service.

The External Systems Interface Layer will also allow interfaces with external systems using Web services and will provide the external Web services interface to the Business Layer service. Just like the services in the presentation layer, services at this level may only support users at one security level, although they may support multiple service requestors belonging to differing privacy groups.

3.6.2. Business Logic Layer

The Business Logic Layer includes the core computational services that perform the bulk of the work for VA systems. It is at this level that the core business processes will be implemented and performed and all of the services except for externally exposed Web services, presentation services, data services, and external interfaces will reside.

3.6.2.1. Native Service Layer

Modernized applications will be developed to be compliant with this SOA and expose all interfaces as services in a native manner or will migrate to that approach over time. It is important that all future releases of these systems conform to this SOA. Business logic services will be implemented at this level. This will include both atomic and composite services that perform the VA business logic and implement VA business processes. They may either be written in Business Process Markup Notation (BPMN) and executed by a BPEL engine or be developed directly in a modern language such as Java or C#. These will be native, modernized services and will be the basis for VA applications. Applications being built at this level will not interact with external applications directly, but will do so through the presentation or external systems interface layer services. This includes links such as queues for COTS based messaging products or other interfaces to external systems.

3.6.2.2. Non-Native Service Layer

The functionality of legacy systems that do not conform to this SOA will be exposed through a series of adapters. Those adapters will provide access to legacy systems data and business logic but appear as services to the other services. These adapters will reside in the business logic layer with the other services rather than on the legacy system side of the firewall. Therefore, all complexity due to the need to cross a firewall and different network segments will be between the adapter and the legacy systems and the adapter will appear to the other services as a standard service at the same architectural layer at which they reside.

Code refactoring is the disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior undertaken in order to improve some of the nonfunctional attributes of the software. Refactoring can be used to replace the legacy code that sits behind a service adapter with native services, replacing both the legacy code and the service adapters.

However, there is currently no budget or other resources to modernize legacy code, just to replace legacy code with services, but as the legacy code needs to be modified for technical or programmatic reasons, the legacy code is to be replaced with native services code.

Current legacy system links with external systems will be retained and will not be required to go through the presentation layer, as would new links for modernized systems, until such time as they are modernized. There is currently no funding, nor is likely to be any funding in the future for the replacement of legacy interfaces with service interfaces just to replace the legacy interfaces, but as they interfaces are modernized for technical or programmatic reasons, the replacement interfaces must be service based.

3.6.3. Data Layer

The data layer will be maintained in the SOA with special services providing data access. In a non-SOA environment, there is a concept of an application owning a database and controlling access to that database, this distinction can be lost in an SOA. Since an application is a collection of services, some of the services will be accessing data, others will be processing data, and others will be validating potential changes to the database. As the boundaries of an application are relaxed, the need for well-defined services for data access must be strengthened. There must be strict controls as to which services can read specific databases tables and which services can create, update, or delete records. In this SOA, these functions are the responsibility of specific data access and data interchange services that will strictly control services access to data. See Section 7 Data Architecture for a more thorough discussion of this topic.

There will be two types of data services that will need to be supported. First, there will be data services that provide data to the business services. These data services will exist in the SOA services layer and will support the separation of business logic and data structures that will provide access to standard collections of data and will be independent of the underlying data structures allowing the structure of the tables and databases to change without impact to the these data services or the business services that they support. Second, there will be a set of services in the data layer that will be explicitly dependent upon physical structure of the data and will access the physical data structures and assemble the data for use by the SOA layer data services.

Access to all “privacy” e.g., PII or PHI data will need to be restricted so that all uses of the data are consistent with the purpose for which the data was collected. The functional steward for the data – the organization providing access to the data, must ensure that the use of the data by the new service is consistent with the existing purpose of the data that the service accesses.

3.6.4. Mapping of SOA Layers to Application Architecture Layers

The previous discussion described the VA SOA in terms of a number of layers. These layers were used to describe the types of services that would be developed based on this SOA. However, it should be noted that the OneVA Enterprise Application Architecture (EAA) is also based on a series of layers and that those layers do not correspond to those described above. In fact, there are SOA-related layers throughout the VA SOA stack as seen in Figure 4. Although most of the user written services will occur in the highest layer of the EAA stack – the SOA Services Level, there are aspects of the SOA throughout the OneVA EAA stack.

The presentation, business, and data services and messages described above will be implemented in the SOA Services and Message Layer. The external system and device interfaces are implemented at the Interface Layer of the OneVA EAA stack. The data services discussed above are implemented in the Data Logic Sub-Layer of the SOA Services Layer. Section 7.2 Data Services talks about services implemented at the EAA Data Layer. The Enterprise Software Sub-Layer specifically the ESB will be discussed below. That discussion includes a discussion of the Message Oriented Middleware (MOM) software that comprises the transport layer.

3.6.5. Enterprise Service Bus and Supporting Infrastructure Layer

The Enterprise Service Bus Layer includes virtually all of the infrastructure support for the SOA including service orchestration and choreography and transport and format transformations. Security gateways and logging as well as authentication also occur in this layer, as well as support for external interfaces and external services. A more complete description of the ESB is provided in Section 3.7.

3.7. Enterprise Service Bus

ESB is a logical entity describing a set of capabilities, not a vendor product – although the capabilities may be provided by a number of vendor products. The ESB will need to support services being provided in each layer of the multi-tier architecture and may be distributed across each of the layers of that architecture. In previous types of architectures, the software in the ESB’s place in the architecture was the magical glue that performed all of the integration (e.g., data integration) functions between systems. There may be a message routing mechanism as well as transport and format conversion, but not syntactic or semantic conversion. Also, orchestration mechanisms may exist at multiple locations in the architecture. Because some services may only be exposed within an application, or a closely related group of applications, orchestration may need to occur within that application or application group.

The ESB capabilities will be built using multiple system components – system components including a variety of COTS packages, processors, and gateways etc. that will collectively be called the VA ESB.

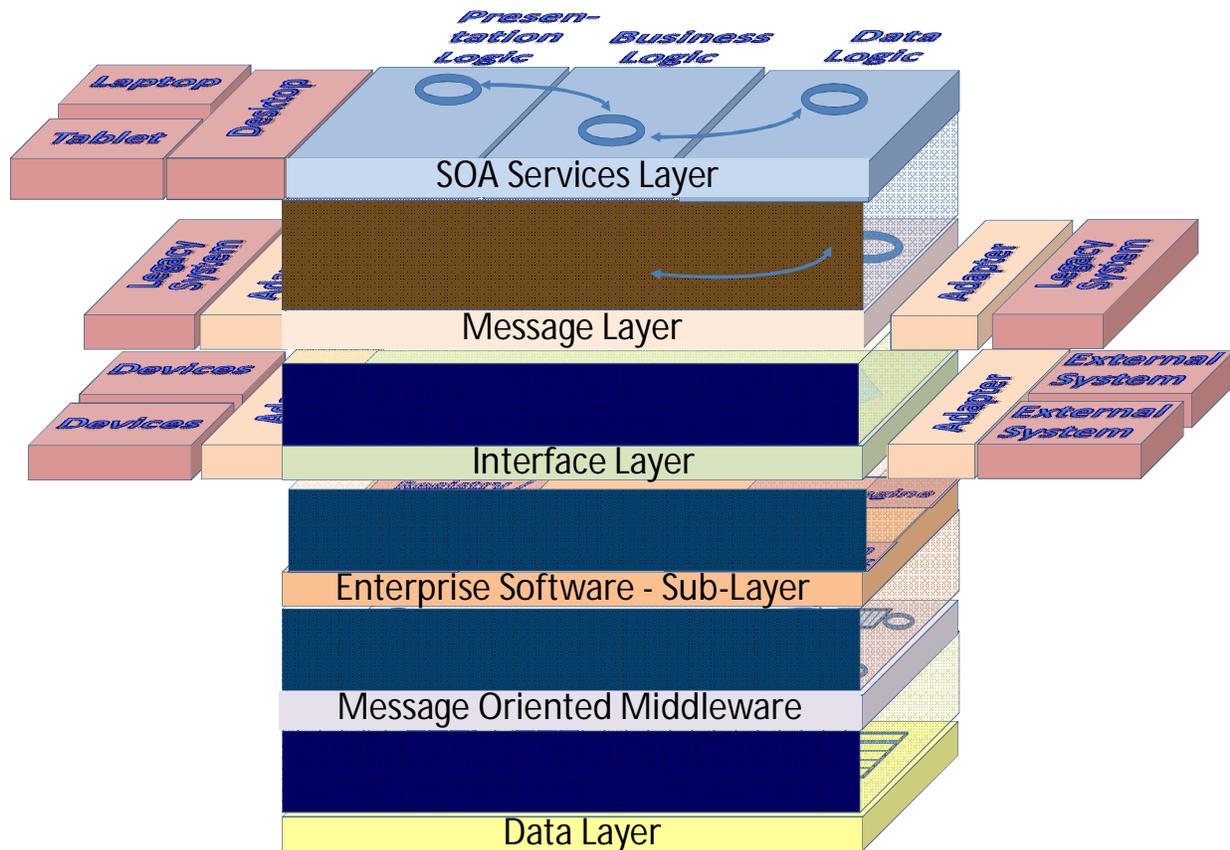


Figure 4 - SOA Stack in the Enterprise Application Architecture

There are a number of other SOA infrastructure components, such as a registry / repository that are required for operation of the SOA but which are not normally parts of an ESB, but are included in the VA ESB. Therefore, the ESB is a logical construct that includes capabilities of multiple hardware and software elements distributed across the enterprise. In addition to the VA ESB, there may be ESBs within business areas (i.e., MIs, segments, or business areas) and applications that implement this SOA. Both the VA ESB and these other ESBs will be implemented within security domains (see Section 8.1 Domain Model). That the ESB is a set of capabilities, rather than physical hardware or software components, is of critical importance, notwithstanding the fact that all capabilities will be instantiated through software and will execute on hardware.

3.7.1. ESB and Integration

The integration of information flows between systems is accomplished through architectural means rather than through a complex piece of software. Unlike previous integration approaches that were product-based – integration in this SOA is through the architecture, not through products. One major reason that previous architectures were unable to provide the level of integration desired was that they depended upon a piece of software to accomplish complex data and systems integration. With this SOA, most of the integration is performed through the development of the ELDM, the harmonization

of message (data element) syntax and semantics, and the use of standard messages. The ESB becomes a much simpler message hub and location for the orchestration of services.

SOAs are not conceptually new; they have been implemented for over thirty years. What is new is that there are now industry accepted standards for communications between application systems and an understanding that data integration must be done in the applications, not in a piece of external software. What the ESB does is:

1. Provide interoperability between a limited number of data formats and transports.
2. Provide communication and routing.
3. Provide certain process and service management capabilities.

The ESB performs format transformations (e.g., from J2EE RMI to DCOM) but does not perform syntactic and semantic transformations (e.g., conversion of data definitions as might normally be done during an “Extract, Transform, and Load” (ETL) process).

Ideally, in the SOA much of the data integration work has been removed because communication between services is based on data definitions contained in the entities from the ELDM. However, for the foreseeable future many of the data exchanges will not be based on native use of the ELDM entities, there will need to be a mechanism that will perform data translation and transformation. This data transformation cannot be done on a point-to-point basis. It must be done from the application’s native data definitions to the standard data definitions and from the standard data definitions to the data definitions required by other applications. Using this approach each application need only transform and / or convert data once and this transformation can be used by any application that has an adapter to allow it to talk the standard definitions. As noted earlier, these syntactic and semantic data transformations are not part of the ESB, but will be the responsibility of the applications. Each application will need to convert from its native data definitions to and from the enterprise native definitions as specified in the ELDM.

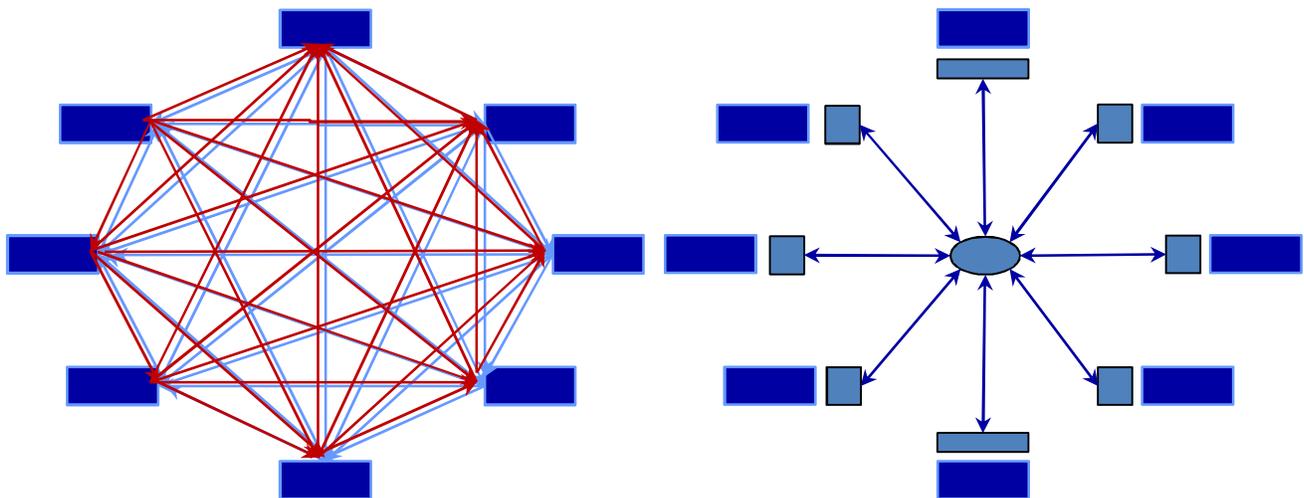


Figure 5 Point to Point vs. Standard Messages with Hub

Essentially, the ESB is designed to provide connectivity between services and provide the supporting infrastructure to support and manage the flow of service requests and responses. Therefore, the ESB

will need to perform a number of specific functions at various levels in the architecture. Some of the major services provided by the ESB will include:

- Messaging services or other means of ensuring the service requests and service responses are delivered to their intended recipients.
- Enterprise infrastructure services (technical services) need to support the delivery and management of functionally oriented services.
- Integration services to ensure that information sent by one service can be read by other services (i.e., format and transport transformation).
- Business process orchestration and workflow management to allow business analysts to change service sequencing to reflect new or modified business processes.

The ESB will need to be able to support a hierarchy of services, each layer of services being built upon the underlying layers. These layers will range from very fine-grained functional services to very high-level, coarse grained functional services. High-level functional services will be composed of lower level services and the ESB will need to efficiently support the large number of subsidiary service calls that result from the invocation of the single high-level service. As will be discussed during the discussion of Security requirements in Chapter 8 “SOA Security,” security and access control requirements will influence the permitted granularity of the services¹⁵.

¹⁵ Essentially there are two security approaches – 1) barrier security, which is an OSI Level 3 approach that physically limits access to a set of services and performs authentication once, at the barrier and 2) authentication-based approaches that ensure the identity and authorization of persons accessing a service. If authentication based approaches are used then some validation of the credentials must be done on every service call and services must be relatively coarse-grained to limit authentication and authorization overhead. The barrier approach which is used in this SOA is more complex to implement but allows significant operational efficiency.

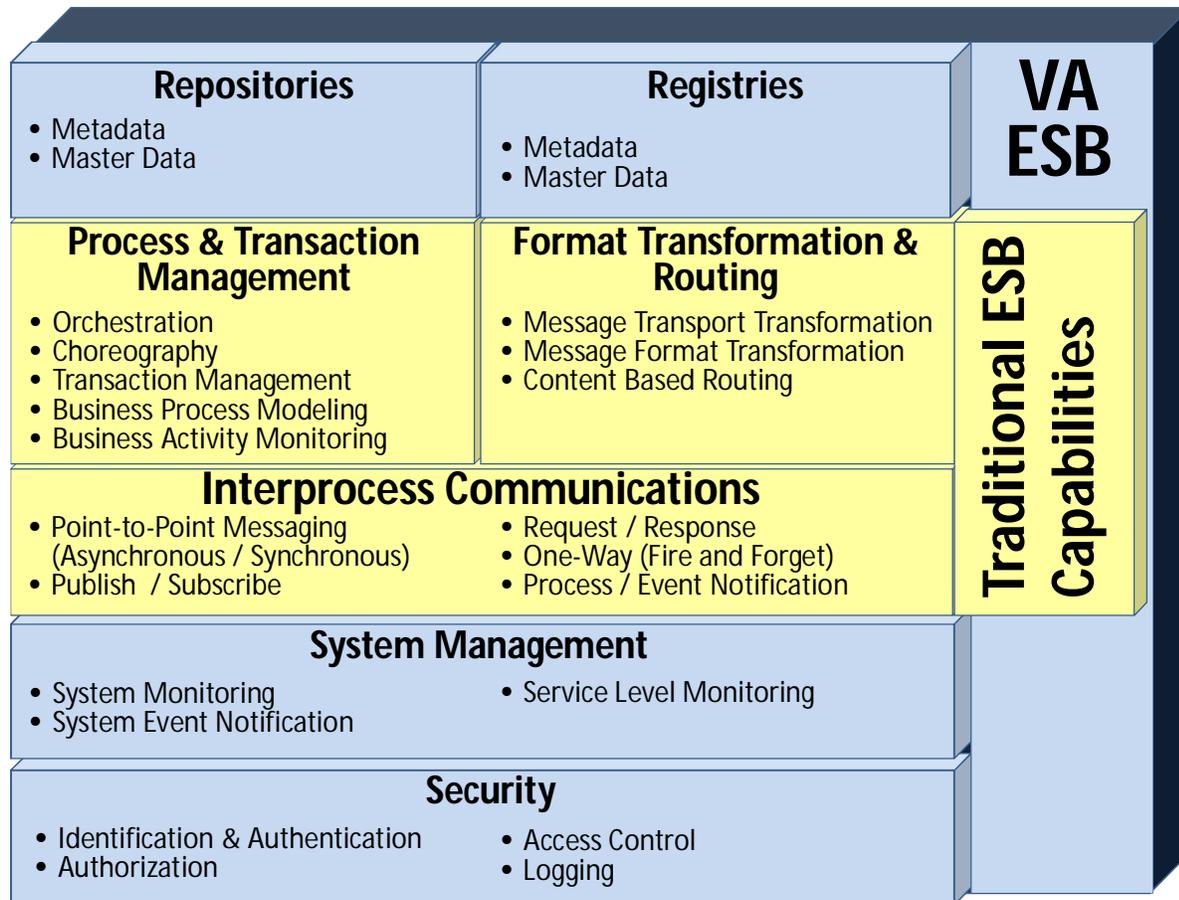


Figure 6 - ESB Capabilities

3.7.2. ESB Mechanisms

One of the basic functions of the ESB is to allow and support communication between services developed by various VA organizations and applications. The communication mechanisms supported by the ESB include **AND ARE LIMITED TO:**

- J2EE Remote Method Invocation (RMI).
- J2EE JMS.
- DCOM
- COTS Messaging.
- Web services (SOAP over HTTP or SOAP /JMS over COTS messaging).

Each service will support interface(s) to the ESB using one or more of the above mechanisms. Services are not limited in the number of the specified mechanisms that they can support. The ESB will perform the transformations between the chosen mechanism and any of the other mechanisms. In choosing a mechanism to support, the developer should be aware of the likely usage of the service and the performance requirements of the service consumers. While the ESB can transform a high performance mechanism to a lower performance mechanism, if a service is implemented using a low performance mechanism (e.g., Web services) the performance will be that of the lower performance

mechanism and the ESB will not be able to provide a high level of performance to the service consumers.

3.7.3. ESB Capabilities

The ESB will provide a number of capabilities. These range from simple communication and routing services to the much more complex process choreography and transaction management services.

3.7.3.1. Interprocess Communication

The most basic capability of the ESB will be to transfer messages between services. This can either be based on synchronous or asynchronous communication approaches. Messages can either be guaranteed delivery or not. Guaranteed delivery assumes that messages are persistent and will eventually be delivered even if the receiving system is not operational at the time that the message is sent.

- **Point-to-Point Messaging** – Ensures that only one receiver consumes any given message. If the channel has multiple receivers, only one of them can successfully consume a particular message. If multiple receivers try to consume a single message, the channel ensures that only one of them succeeds, so the receivers do not have to coordinate with each other. The channel can still have multiple receivers of a single type to consume multiple messages concurrently, but only a single receiver consumes any one message.
- **One Way (Fire and Forget with Guaranteed Delivery)** – A message is sent from a single sender to a single or multiple receivers. This Interprocess Communication method is built upon the point-to-point communications described above. However, at each step in the transfer the message is copied to persistent storage and the message is not considered to be successfully delivered until processed (e.g., moved to local storage) by the receiver.
- **Publish / Subscribe** – Allows multiple receivers to register (subscribe) to receive notification of some event or to receive updates to some data store. It has one input channel that splits into multiple output channels, one for each subscriber. When an event is published into the channel, the *Publish-Subscribe Channel* delivers a copy of the message to each of the output channels. Each output channel has only one subscriber, which is only allowed to consume a message once. In this way, each subscriber only gets the message once and consumed copies disappear from their channels.
- **Request / Response** – Is composed of a pair of one-way messages. In this case the original sender sends a message to the original receiver. The receiver performs some work and then creates a second message, which it sends to the original sender. Request / reply consist of two independent messages and is not a simple “ack” of the original message.
- **Process – Event Notification** – Allows a request to be registered to be informed when some process completes or fails or when some specified event occurs. This allows for asynchronous processing as it allows a process to be notified when previously initiated processing is complete.
- **Event Driven Messaging** – Is used to enable the service consumers, which are interested in events that occur to get notifications about those events as and when they occur. Event-Driven Messaging uses the publish / subscribe mechanism that ensures timely notification of event related data to the service consumer. The event manager automatically notifies all the

interested service consumers about the occurrence of a particular event the moment it actually happens. Event-Driven Messaging requires an event manager, which could be the COTS messaging product that provides the publish / subscribe mechanism, with whom the service provider registers its events. The service consumers then register their interest in few or all of the advertised events. Upon the occurrence of an event, the service provider informs the event manager that then notifies all of the registered service consumers. This communication mechanism shares its roots with the Observer pattern applied traditionally within the object-oriented world. The actual implementation of such a publisher-subscriber based communication mechanism requires a mature message tracking and forwarding mechanism such as the COTS messaging capability included as part of the ESB. The application of this pattern helps to further decouple the service consumers from the service providers and increases the overall reliability of a service composition. For which it seeks notification and to be notified when those events occur.

3.7.3.2. Process & Transaction Management

Process and Transaction Management provides a higher layer of services above the Interprocess Communication layer described above. This layer deals with how business processes are executed which may include the execution of a number of automated and manual sub-processes. Process management capabilities include:

- **Orchestration** – Refers to composing services into an overall process, for example, a higher-level service, which is internally made up of a series of lower-level (finer-grained) processes sequenced together. The internal sequencing could be a simple linear sequence of service invocations, or could include branches, conditional paths, human interaction steps, or data transformation steps. A central controller controls process sequencing, from start to finish. From that perspective, it can be thought of as a command and control style of service invocation and execution.
- **Choreography** – Refers to ways of defining how multiple applications collaborate to accomplish an overall business objective, by exchanging messages in a peer-to-peer style. There is no particular central point of control. Rather, execution of an overall business flow is shared and distributed across all the applications participating in the collaboration. This type of interaction might be found in cross-enterprise business flows where there is no central controller that spans all of the enterprises. This would occur when a single business process requires the execution of services in multiple organizations. For example, in a Business-to-Business (B2B) process interaction, choreography would specify that an order require a credit check with an outside organization and access to information from a shipping company for delivery information.
- **Business Process Modeling** – When business process orchestration tools are used, the business process model that is created can be used to predict the cost, performance, and response time of the business process. This is accomplished by assigning resource and time requirements to each step of the process and using queuing theory or simulation models.
- **Business Activity Monitoring (BAM)** – Like systems models which monitor the performance of systems hardware and software components, business process monitors monitor, measure, and report the performance of business processes. The use of functionally-oriented services that align with major segments of a business process and the use of the

business process orchestration tools allow monitoring to be performed at the business process level. BAM provides this capability at the business process level and provides real-time management information on the performance of its business processes, which can then be reported through a graphical user interface. For example, payment of a Veteran's college tuition would be a business process that starts with the Veteran submitting an application for benefits and the delivery of a check to the institution months later. BAM would be able to provide statistics on the number of times this process was initiated over a period of time, the number of executions that completed successfully, and the status is those that have not yet completed successfully.

3.7.3.3. Transaction Management

In the context of the SOA, a transaction management is performed within the context of an orchestration or choreography and not only deals with the control of the process flow through the orchestration, but includes the exception processing related to failed transactions and the correlation of transactions (services).

3.7.3.3.1. Compensating Transactions

In the context of an SOA, a transaction can be a long running series of orchestrated services. Online Transaction Processing (OLTP) use "two-phase commit" to assure that either all database updates related to a transaction are made or, in the event of a failure of the transactions, that none of the database records are updated. One requirement to be able to perform the two phase commit is the ability to lock the all of the database records that will be changed for the duration of the transaction. It is not necessarily possible to do this with long running SOA services.

Transactions that use two phase commit are sometimes referred to as ACID (atomic, consistent, isolated, and durable) transactions and are commonly used in online transaction processing systems to maintain consistency of databases. Because an SOA "transaction" may run for minutes, hours, days, or even longer, locking databases and implementing a two phase commit for database updates may not be feasible. For example, a single orchestration may run from the time that a Veteran checks into a VAMC and run until the time that the Veteran is released from the VAMC. Locking a database record for the duration of the Veteran's stay in the VAMC is not feasible as the locked records are unavailable to any other processing for the duration of the lock.

In an SOA, backing-out a transaction is normally accomplished through the execution of one or more compensating transactions – transactions specifically run to put the databases back into the proper state. Transaction management is the set of services required to support the management of transactions. ACID transactions may be required where the service is integrated with existing two-phase commit-based transactions since existing two-phase commit transactions may require two-phase commit participation. Because they are only invoked when there is a problem, compensation transactions greatly reduce the runtime overhead that is inherent with a two-phase commit. However, the development time for a two-phase commit transaction is much shorter and less problematic than the development of compensation transactions. When developing compensating transactions, all possible error modes must be identified to allow identification of all possible inconsistent database states so that transactions can be developed to put the database back into a consistent state. Of course these transactions may also fail. These compensating transactions must be incorporated into orchestrations.

3.7.3.3.2. Exception Processing

Service orchestration makes exception processing for services in an SOA much more complicated than exception processing for OLTP transactions. Consider the two possible locations for a service failure illustrated in Figure 7.

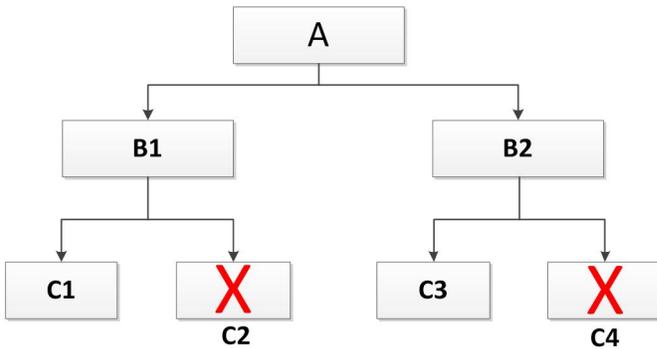


Figure 7 -- Exception Processing for Orchestrated Services

failure illustrated in Figure 7. If the service C2 fails the effects are more complex because it is part of a larger orchestrated service (B1 and A). When C2 fails B1 must fail, or at least be notified of the failure and then compensating services to undo any database or other changes that were made by service C1 must be executed. If C2 were not to fail, but C4 fails, the situation is even more dire as B2 must be failed, which causes C3 and A to fail which in turn causes B1 and then C1 and C2 to fail. Compensating transactions may need to run to undo the

effects of A, B1, B2 C1, C2, and C3. Further complicating issues is the fact that compensating services may need to be run for services that have long since completed. Exceptions will need to be reported both up and down the tree alerting services both above and below the service that fails.

The fact that a service fails does not necessarily mean that compensating services must be run. The services may not have made changes to databases that need to be undone. Further, a service may call a number of services and be able to continue operation even in the event that some service fails. A service could make a request to a number of services and then proceed based on the responses that it receives in a fixed amount of time (e.g., if the service is responding to an online request), but if this is the case it must be specifically designed for.

3.7.3.3.3. Service Correlation

In high volume systems a second or even third request for a service may be made before the first service request has been completed. For example multiple requests for patient records may be made in the time required to retrieve a single patient record. It cannot be assumed that the service responses will be returned in the same order in which they were issued. Therefore there must be a process that determines which service response corresponds to which service request. This process is called service correlation and is the alignment of service responses with the service requests that generated them.

3.7.3.4. Transformation and Routing

Transformation and routing functions are higher-level communication services – above the Interprocess Communication layer discussed above.

- **Message Transport / Format Transformation** – Provides communication services that support different message transports (e.g., HTTP vs. COTS messaging) or formats (e.g., SOAP HTTP vs. J2EE RMI) and converts between them as required (Note: overall performance will be determined by the performance of the slowest transport / format).
- **Content-Based Routing** – The Content-Based Router examines the message content and routes the message onto a different channel based on data contained in the message. The routing can be based on a number of criteria such as existence of fields or specific field values, etc. When implementing a Content-Based Router, special caution should be taken to

make the routing function easy to maintain, as the router can become a point of frequent maintenance. In more sophisticated integration scenarios, the Content-Based Router can take on the form of a configurable rules engine that computes the destination channel based on a set of configurable rules. Content-based routing can cause additional security and privacy issues as payload data is processed and the results of that processing are be logged. Therefore, both security and privacy reviews will need to consider any content-based routing rules as well as the log data that is produced.

- **Syntactic and Semantic Transformation** – Syntactic and semantic data transformation are not performed by the ESB, but are performed by a data transformation adapter that is owned by the application. The message transport / format transformation capabilities of the ESB assume that the messages passing through the ESB use VA standard data definitions (syntax and semantics) which are to be based on the data definitions in the VA ELDM.
- **External Message Transformation** – While VA can specify that internal messages conform to VA standard syntax and semantics, this is not so for messages to or from external entities. Just as legacy applications will be required to provide adapters to convert their internal messages to the VA standard, owners of external interfaces will be required to transform messages that pass over external interfaces to the VA standard before they are transmitted through the VA. As noted earlier, legacy interfaces are allowed to continue, but as they are updated for technical or functional reasons, they must be updated to use the adapters identified in this section.

Making the transformation of local data formats to the ELDM standard ensures that each application only needs to understand two data formats - its own internal data format and the enterprise standard. Further, the owner of the application best understands the application's internal formats and is best able to do the transformation. This allows the ESB to be developed and maintained by an enterprise services group without requiring knowledge of the detailed application formats.

Further, performing the data transformations in the ESB could potentially require $n*(n-1)$ transformations and seriously reduce scalability as the number of applications passing data through the ESB increases.

3.7.3.5. Registries and Repositories

In the Web services world, there is a concept of “service discovery” in which a user can go to a central location to identify which service providers are providing a service that meets specified requirements. This is done so that the Web service user can choose which of the services to access. Because of the sensitivity of VA's data and the nature of its operations, VA's approach to publishing services will be markedly different. First, applications will not allow unknown entities to connect to its systems or to browse for or access its services without specific authorization. This authorization should include ensuring that all security and privacy requirements related to service and the data that it exposes continue to be satisfied. All services provided to entities external to VA will be governed by ISAs, ICDs, and MOUs between VA and each such entity. Second, only a single instance of specific services will be published so as to attain VA service reuse goals.

Design time registries help developers locate assets, make decisions about which ones are best to use among many that might be similarly appropriate or adequate, and understand the various costs involved in their consumption. Runtime registries help systems make automated, policy-based

decisions about service selection. However, there is no such design time vs. runtime delineation for repositories. A repository is simply a repository – to be used at both design time and runtime.

Design time registries help developers locate assets, make decisions about which ones are best to use among many that might be similarly appropriate or adequate, and understand the various costs involved in their consumption. Runtime registries help systems make automated, policy-based decisions about service selection. However, there is no such design time vs. runtime delineation for repositories. A repository is simply a repository – to be used at both design time and runtime.

It is important to distinguish between two basically different concepts for the discovery of Web services. The first is essentially a manual, design time activity, where developers (or other people who want to find a particular web service) search for information with some kind of service discovery tool. The second is basically an automatic, runtime activity, where a service consumer (which is a piece of software) has an indirect reference to the service it requires and looks up that reference in a runtime registry to find the service contract (e.g., WSDL file) that tells the consumer where to find the service and how to bind to it, in a manner similar to how the internet Domain Name Service (DNS) binds names of Internet addresses to the physical IP address of a requested server.

This manual, design time sense of service discovery is best handled by an asset management or metadata management tool. Such tools may support Universal Description, Discovery and Integration (UDDI) as part of how they provide for the discovery of services, but the information that might be contained in an UDDI-enabled registry is only part of the metadata that asset management tools must deal with. In other words, asset management tools are metadata management tools that may or may not support the UDDI standard, but in any case, must offer users far more information than what would normally be included in a UDDI registry. This includes metadata about Service levels, support for Service consumers, security and policy requirements and information about what the service definitions are, as well as how to use them. Asset management is essential to how people use an SOA to build loosely-coupled, coarse-grained systems composed of reusable service assets, but is only indirectly related to the runtime operation of the architecture. As part of this SOA that there will be a central VA level directory.

In contrast, the automatic, runtime sense of service discovery is essential to the proper operation of an SOA. To understand why runtime discovery is so important, consider that a service consumer should query the registry every time a particular service is requested. The registry might conceivably return a different service contract from one request to the next. For example, a Service-Oriented (SO) Management application that incorporates an UDDI-based service registry might route a request to a different server to provide better scalability or availability, support different service versions or service levels, it may handle protocol translation, or may even convert a synchronous request to an asynchronous one.

In addition, service consumers may not be aware of security or policy changes that may require a change in the way that they access a given service. In these cases, there may be a service negotiation where service consumers communicate first with a registry to determine how best to access particular functionality. Regardless, all of this management activity takes place behind the scenes, in that the user of the service need be none the wiser about the details of the service contract or that the underlying service provider is different from one request to the next.

3.7.3.6. Service Versioning

Service versioning is required when changes are made to an existing service and either the new service or both the new and old service must be made available.

3.7.3.6.1. Versioning for Externally Exposed Services

Any changes to services visible to external users would need to be publicized well in advance and only be implemented after due notice has been provided. It is unlikely that all, all private users, non-VA government users, and even all VA applications will ever be able to upgrade to new versions of existing services or to new services on the same date. Therefore, it will be likely that applications will need to support multiple versions of the same service for some period of time. This implies that there would be a need for UDDI-based registry services to support external services even though service discovery would not be allowed in the traditional sense.

Just as in the case of the external users, an UDDI-based registry will be needed to support internal users as well. Although, a major feature of the SOA is that services are developed once and are reused throughout the enterprise, there will still be a need for a discoverable registry. While users requiring a specific service will find only one provider of that service, multiple versions of the service may exist. A UDDI-based directory will allow for the location (i.e., URL) or other characteristics of a service to change and for these changes to be hidden from the applications that access that service.

3.7.3.6.2. Versioning for Internal Services

The intention of this SOA is that once published services do not change, either they are deleted or a new version of the service is created and included in an orchestration. Each new version of the software would be added to meet some specific new requirements. New versions of services can be localized if the services are fine grained so when requirements change the bulk of the services implementing the requirement remain the same and those parts that need to change are segmented into new finer grained services. The users for whom the requirements did not change would see no change and the only change impacting them would be that what was formerly an atomic service becomes an orchestrated service. This leads to finer grain services and an increase in the number of small, independent orchestrations rather than the increasingly complex, interrelated control logic the grows as new requirements are added to monolithic systems.

3.7.3.7. Directory Services

UDDI directory services will be provided to support relocation of services and to allow multiple versions of a service to co-exist as when services are updated, it may not be possible for all users to use move to the updated service at the same time. The directory services are included to ensure that there is no need to hard code service addresses or locations into applications or services. The UDDI directory will be one component of the metadata registry / repository that is used to support services management.

4. ESB Conceptual Model

The Conceptual Model Diagram identifies the major layers, towers, and components of the ESB. The components depicted in this diagram are conceptual in nature and there should be no assumption that there is a one-to-one mapping between these components and physical servers or software products. Following this conceptual description is a logical description of the ESB to include the identification of the logical services and services that will comprise the ESB and to provide the capabilities described in the Conceptual Model.

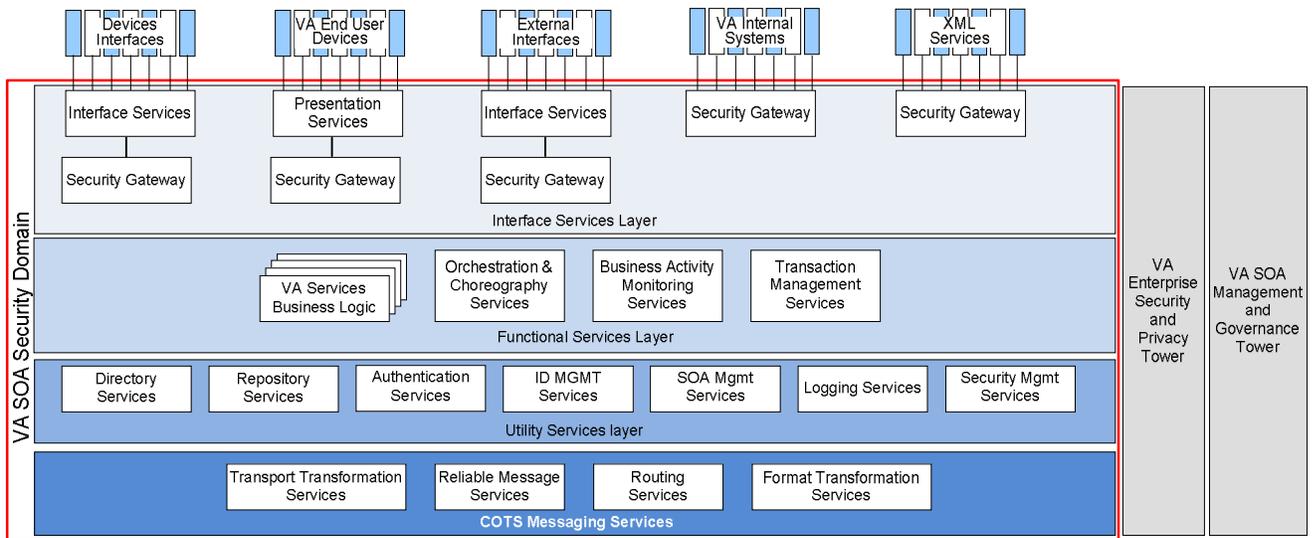


Figure 8 - ESB Conceptual Model

The conceptual design diagram has five layers, four of which are included in this design and one of which identifies the entities with which the ESB communicates. Of the four layers included in this design, two layers are associated ESB capabilities – the COTS Messaging Services Layer and the Functional Services Layer discussed above. The other two layers – the Utility Services Layer and the Interface Services Layer – provide capabilities that form a necessary part of the VA SOA infrastructure and provide support for the ESB capabilities.

In addition to the layers in the conceptual design diagram, there is the VA Enterprise Security and Privacy tower that crosses all layers of the conceptual design, but which is not part of the VA SOA security domain, as this tower provides the policy, procedures and tools that are required to support the security and privacy capabilities provided within the SOA. The VA Enterprise Security and Privacy tower is not explicitly discussed in the conceptual model, but security aspects are discussed in the logical model of the system that will need to provide the required services and later in the document with respect to SOA and ESB security (see Section 8 **SOA Security**). The hardware and software that support SOA security are included inside of the ESB security domain and thus, are not part of the external tower. The Security and Privacy Tower refers to the set of policies, guidelines, and processes and procedures that are an integral part of the design and implementation of the VA SOA infrastructure. By the same token, the SOA Management and Governance Tower relates to the management and governance:

- Policies,
- Guidelines,
- Processes and procedures,
- Metadata related to the SOA, and
- Life cycle information about services regardless of their life cycle stage related to the SOA

rather than a set of hardware, software or services.

It should be noted that the data logic server does not appear in any of the layers in the ESB conceptual model. That is because the data server is located outside of the ESB and thus, is not part of the ESB and therefore is not discussed in this section.

4.1. **Interface Layers**

The interface layer provides access to the VA ESB capabilities by entities external to the VA SOA security domain.

4.1.1. **External Interfaces**

External interfaces are interfaces with entities external to the VA and include a wide variety of interfaces and transports. These interfaces may be over dedicated lines or channels or over the public internet.

All external service requests that enter the ESB Security Domain must pass through a security gateway and an interface layer service where the identity of the message source, the authenticity of the message, and the well-formed nature and legality of the message are ensured. The security gateway will perform logging of all inbound and outbound messages. The interface and presentation level services will be developed and provided by the interface owner and the service developer. The presentation and interface services are required to be on subnetworks that are isolated from the rest of the VA SOA security domain and thus external to the VA ESB security gateway.

4.1.2. **Internal Interfaces**

Internal interfaces are interfaces between the services and applications operating within VA and on the VA network. Service requests crossing the internal interfaces will pass through a security gateway so that the authentication and authorization can be performed prior to entry into the ESB security domain.

4.2. **Functional Services Layer**

The functional services layer provides the capabilities used by the service providers to implement the business services. These capabilities are provided through the following services:

- **Orchestration and Choreography Services** – Refers to composing services into an overall process, for example, a higher-level service which internally consists of a series of lower-level (finer-grained) processes sequenced together. Orchestration is the sequencing of services within an organization and choreography is the sequencing of services across organizations. A service consisting of an orchestration of other lower level services can itself be orchestrated into to a higher level service.
- **Transaction Management Services** – Are the set of services required to support the management of executing services. In an SOA, backing out a service is normally accomplished through the execution of one or more compensating service requests to put the databases in the proper state because the long run time of many services and the composition of many low level services makes locking database records for the duration of a service infeasible. Transaction management services are the services required to support service execution and the initiation and management of compensating services when they must be initiated due to a service failure which left data stores in an inconsistent or incorrect state.
- **Business Activity Monitoring Services** – Monitors, measures, and reports the performance of business processes.

- **Business Logic Services** – Are the application servers and services that allow and support execution of custom developed services.

4.3. Utility Services Layer

Utility services are services required to support the VA SOA and provide the ESB capabilities that are to be provided. These services are necessary for the operation of the ESB. The hardware and software configured to provide these capabilities are in the VA SOA security domain. These utilities include:

- **Directory / Registry Services** – will provide a superset of UDDI capabilities and will be the repository for location information for all exposed services. The registry will include location and service description information for services in each phase of the life cycle.
- **Repository Services** – Maintains the artifacts that were developed during the service specification, design, and development. The repository services will maintain metadata for production services. This metadata repository will be in addition to the service management and governance information that is kept external to the SOA security domain. The repository services are also used by the dehydration process.
- **Authentication Services** – Each user or system requesting services, each service, and many of the ESB components will have a certificate issued by a certificate authority subordinate to the Federal Bridge which will be used to authenticate users and to digitally sign and verify service requests and responses.
- **ID Management Services** – VA is currently in the process of designing and developing an enterprise-wide Identity Management solution that will result in VA personnel and Veterans being issued a single identity for use across the VA and for all VA systems. A single user identity will be associated with one or more roles. These roles will be used to support the authorization for access to data from services and systems throughout VA. Once the VA standard solution and the associated product set are selected they will be implemented as part of the ESB supporting infrastructure.
- **SOA Management Services** – Will maintain metadata about the service such as specified Service Level Agreements (SLAs) and the level of service that has been / is being provided.
- **Logging Services** – Creates a log for use by security and internal affairs as well as user requested audits to verify the receipt of messages. Two logging services are provided one for event based logging – e.g., authentication decisions, and a second to support payload logging should it be needed. Both log systems will aggregate logs generated by each of the system components that log information.
- **Security Management Services** – Provide the support for the security capabilities that are included in the implementation including access control lists (ACLs) and access rules specifying the access privileges of authenticated and authorized personnel. These include the level of access and the types of transactions and functions that are permitted (e.g., read, write, execute, delete, create, and search).

4.4. COTS Messaging Layer

The COTS Messaging Layer will provide a range of services that provide basic bus functions including communications and syntactic message transformation services:

- **Reliable Message Services** – Ensures that messages are delivered reliably between sender and receiver(s) in the presence of software component, system, or network failures.
- **Routing Services** – Examines the message and routes the message onto a different channel based on the address or data contained in the message.
- **Format Transformations** – Provides communication services that support different message formats (e.g., SOAP HTTP vs. J2EE RMI) and converts between them as required. This transformation is a format transformation only and does not include semantic transformations (e.g., from application native definitions to VA Standard Messages).
- **Transport Transformation** – Provides communication services that support different message transports (e.g., HTTP vs. COTS messaging) and converts between them as required. This transformation is a format transformation only and does not include syntac or semantic transformations (e.g., from application native definitions to VA Standard Messages).

The most basic capability of the ESB will be to transfer messages between services. The COTS messaging layer will provide a number of forms of messaging including:

- **Point-to-Point Messaging** – Ensures that only one receiver consumes any given message. If the channel has multiple receivers, only one of them can successfully consume a particular message.
- **One Way (Fire and Forget with Guaranteed Delivery)** – A message is sent from a single sender to a single or multiple receivers. However, at each step in the transfer the message is copied to persistent storage and the message is not considered to be successfully delivered until it is processed (e.g., moved to local storage) by the receiver.
- **Publish / Subscribe** – Allows multiple receivers to register (subscribe) to receive notification of some event or to receive updates to some data store.
- **Request / Response** – Is composed of a pair of one way messages. Request / reply consist of two independent messages and is not a simple “ack” of the original message by the sender. This can be in two flavors, synchronous and asynchronous.

COTS message queues are an allowed mechanism for service requests in the VA SOA. The COTS messaging layer will provide COTS message based communication both with the COTS messaging services at the applications and to allow service requests to be initiated through placing messages on a COTS message product queue.

4.5. VA SOA Management and Governance Tower

In addition to the SOA Management Services included in the Utility Services Layer, there will be another set of capabilities to be provided. Where the SOA Management Services in the Utilities layer provide runtime management, the VA Management and Governance Tower includes policy information and controls such aspects of the operation. These relate to the management of the policies and procedures governing operation of the SOA and include items such as the governance processes related to promoting services from development to test and then to production etc.

4.6. VA SOA Security and Privacy Tower

Security will be managed by a series of security policies that will be applied by the security gateways. Security policy management services, which are included as part of the SOA Security and Privacy

tower, are the services that are used to specify and maintain those policies. The security management services in the Utilities Layer deals with subjects such as the implementation and management of ACLs while the Security and Privacy Tower manage who is allowed to update ACLs.

5. ESB Logical Model

The ESB capabilities described above are implemented as a series of services (although not SOA services) and these services are instantiated using a set of hardware and software resources. This logical model contains a description of the logical services used to implement the capabilities discussed as part of the conceptual model. The logical model describes the basic components of the system, their functions, and the interconnections between them. The logical model provides details on the capabilities provided, the logical components used to provide those components, and the implementation of the system functions. This is a logical, not a physical description, of the ESB and discusses logical components, not physical devices. The fact that two components are described as distinct does not imply that they must be implemented in a single physical device or the fact that a component is described as having a collection of capabilities imply that all of the capabilities are provided in a single physical device.

5.1. Interface Level Services

The SOA Framework which defines the architecture on which the VA ESB is being designed and built requires the use of interface level services to segregate external users and external system interfaces from the business logic services that are a part of the ESB. Until such time as VA applications are modernized using services, external input to the ESB will be over system-to-system interfaces rather than through workstation input. Users will access the current VA applications that will access services in the ESB. It is expected that these applications will enforce the appropriate security rules such that only authorized users will be permitted to initiate service requests.

For workstations the interface level services will include presentation level services that will be implemented using externally supplied web servers and web application servers that control the material that is presented to the user. These interface and presentation services are created by the business service developer. These presentation services receive the user input and control what is output to the users.

Interface layer services are the first landing for any service requests entering the VA SOA security domain. The interface layer includes four components:

- **External Device Interface** – Provides the physical interface for devices interfacing to the ESB.
- **Gateways** – Provide security policy enforcement, authentication, and auditing of service requests and responses. They are able to provide XML schema validation to ensure that incoming messages are properly formed.
- **Interface Services** – Provide access to an external interface through a NIEM and SOA compliant service.
- **Presentation Service** – Which provides browser access to VA level services where required.

The interface level services are responsible for determining that an incoming service request is syntactically valid, i.e., if the request is for a Web service that the XML describing the service is well formed. Likewise, for information coming across external interfaces, the interface level service will

validate the authenticity of the sender and the validity of the received message. Preferably this authentication will be made using Public Key Infrastructure (PKI) certificates issued by a Certificate Authority that is subordinate to, or supports interoperability with, the Federal Bridge Certification Authority (FBCA).

5.1.1. External Device Interfaces

There are two instances in which the external device interface will be required. First, some applications require input from specialized input devices. The second instance in which external device interfaces are required is to support non-TCP/IP legacy interfaces.

The VA SOA requires that all external interfaces be exposed as VA level services using VA standard compliant data element definitions. It is unlikely that all of the VA external interface partners will ever concert to the use of VA standard messages. The owner of the interface will need to provide the adapter to transform the message to VA message standard.

5.1.2. Gateway Services

The operational concept for the SOA infrastructure is that it will be implemented as a logically isolated network segment (i.e., either a physically isolated segment or a VPN segment). The presentation and interface servers will be on a separate VPN(s) or network segment(s) from the business logic servers and other ESB components. Further, authentication will be performed on entry to the network segment (security domain) that provides the VA ESB capabilities.

The SOA design requires the separation of the ESB components business logic from the presentation layer. For the Web services entry point to the ESB, the gateway processors will serve as the interface layer as specified in the architecture.

5.1.3. External Gateways

The external gateways perform two functions. First, is a security function as the external gateways will separate the ESB network segment from networks external to VA. Second, the external gateways will provide presentation level services related to the validation of the correctness of the input messages. As noted, the security function separates the VA ESB network segment from external network. The external network links will typically be dedicated links or other links for which an Interface Security Agreement exists and are already used to transfer information.

The gateways will be required to perform a variety of services including:

- User authentication.
- Logging of user requests and responses.
- Security policy enforcement.
- Access control.
- Input message validation.

5.1.4. Internal Gateway

The Internal Gateway provides a gateway between the SOA security domain (network segment / VLANs) and other VA security domains (networks / network segments / VLANs). The architecture specifies that service requests that originate elsewhere in VA will have been authenticated and signed

by the application based on the same root certificate authority as used for authentication in the SOA security domain and thus, they do not need to be further authenticated.

5.2. Web Services / Services

One mode of operation will be the use of orchestrated, XML-based services that conform to the Web services standards – Web services. These services will be and can be orchestrated to provide composite services. Web services will be the preferred interface with external entities although other types of services will be supported. The ESB will be able expose services to external users through the Web services interface.

The VA SOA will support services other than Web services. These may include both XML based messages over a wide variety of transports other than HTTP as well as non-XML messages over that set of transports. In addition to supporting SOAP over HTTP the VA SOA supports RMI, JMS, and COTS messaging systems.

5.2.1. BPEL Processing

The ESB will include a BPEL processor. The BPEL processor is used to orchestrate the execution of services – both Web services and non-Web services. However, BPEL processors can only orchestrate services whose orchestration is described in BPEL – which is XML based. The ESB will perform Extensible Stylesheet Language Transformations (XSLT) transformations and support XML Schema Definition XSD and XPath on XML formatted service requests. In addition to orchestrating Web services, the BPEL orchestration service will allow the orchestration of non-XML based Web services, including some custom business logic services. This additional business logic can be either Java or .Net based, although Java based services will be preferred and are expected to be by far the more common.

5.2.2. Business Activity Monitoring

Business Activity Monitoring (BAM) allows for the collection, analysis, and reporting of data obtained from the BPEL orchestration. The orchestrations can be instrumented to allow collection of information regarding the number of time individual services have been executed, the processing time of individual services, and other characteristics of the services and the orchestration. BAM provides information about the execution of the business process that is just not typically available through system based runtime performance models. The emphasis is on reporting business performance rather than service performance.

BAM information is stored in a dedicated database for analysis and reporting. This data will be made available to users in VA business areas through a set of Web services. This data may then be exposed through management dashboards or other reporting tools. The BAM capabilities will also be used to report service SLAs to VA business areas. BAM collects business level performance information from inside an orchestration.

5.2.3. COTS Messaging Transformation

Most of these interfaces are of an externally specified or agreed to format and cannot be changed without agreement of the interface partners. Some of these interface specifications are set by federal law or regulation, industry standard groups, or merely the fact that potentially tens of thousands of external users use that interface and are not likely to convert to XML any time in the foreseeable future.

The ESB supports a number of mechanisms over the COTS messaging products such as native formats, XML, JMS, or RMI. It is expected that much of the communications between will continue to be COTS message based. The ESB provides direct support for COTS messaging and provides for the format conversion of COTS messages by converting between the various message formats that can be used over the COTS messaging transports. This capability provides transformation for COTS messaging based messages without requiring use of an XML based intermediary format.

5.2.4. Security Services

Security for the ESB is provided in two fundamental ways 1) through the implementation of security domains and 2) access control through the use of role based security and PKI based authentication. The VA SOA design separates authentication from authorization. The ESB performs authentication on entry to its security domain and accepts authentications, but, the only authorization that the ESB performs is to authorize entry into the ESB security domain and by extension authorization to those services inside the security domain that do not require additional authorization. Services requests (messages) are required to be authenticated and authorized to enter the security domain and in addition may require additional authorization to access any specific service. This access may be provided on an individual or role basis.

5.2.4.1. Domain Security Services

The ESB security domain will be separated from other elements of the VA network and subnetworks either by being on a logically isolated VLAN or on a physically separate network segments. Entrance and exit from the VA SOA security domain will be through gateway services that will perform PKI-based user authentication that will read and verify digital signatures. In addition, the presentation and interface services that are exposed to external users will be on a separate network segment or VPN than the rest of the ESB infrastructure.

5.2.4.2. Role Based Security

Gateway services will manage access to the ESB security domain. On entering the ESB security domain through the gateway the user will be authenticated with respect to both their identity and role. Role based security will be enforced at the gateway and through the identity management services to be described later. Access to specific services may be controlled based on the user role.

5.2.4.3. Logging Services

Logging is performed primarily for two purposes – to support auditing and to support recovery.

5.2.4.3.1. Audit Logging

The audit logging service shall log all service requests that enter or leave the security domain and which therefore, interact with the ESB. In addition, logging shall be performed at any place where there is a request for authentication or an authorization decision is made.

5.2.4.3.2. Logging for Security, Privacy, and Internal Affairs

Security, privacy, and Internal Affairs logging is required for any user, system administrator, application administrator, or any other person, system, or agent that accesses any server or service that accesses any data or that has the ability to impact any user access rights or authentication decisions. Any action performed by any of those entities while signed on whether or not they do or do not impact any authorization or authentication actions will also be logged. Special attention will be accorded to any service requests or data accesses that access PII or PHI. This specifically includes

access to log data which may include both the requestor's identity as well as the personal data relating to the subject of the service request.

5.2.4.3.3. **Recovery Logging**

Individual applications or services may perform such additional logging as they determine is required for error recovery, to support the selection of compensating transactions to run in the event of a service failure, or for application or service error diagnosis. The level of application or service logging will be at the discretion of the application or service and can be changed as desired by the application. These logs will be generated, maintained, and deleted by the applications or services. The services will determine the amount of time for which those logs must be maintained.

5.3. **Services Management**

The VA ESB will provide a services metadata repository to support the management and governance of services exposed at the VA level. This metadata repository will include, but not be limited to, copies of the:

- **Service Package** – The information package developed by the consumer and provider of a service specifying the functionality, inputs, outputs, SLAs and other requirements for the service.
- **Service Level Agreements** – A written agreement between the service provider and service consumer specifying expected workload, performance, and availability etc.
- **Service Contracts** – MOU, MOA or other agreements between parties governing the use of information available through the service.
- **Service Dependencies** – Specifies on which services, if any, the given service is dependent upon.
- **Service Performance** – The measured response time and resource usage of the service.
- **Service Usage** – Counts of the number of times that a service has been requested and the number of times that it completed successfully.

This repository will be used to support services management and governance.

5.3.1. **Data Validation**

When an XML message is sent by a service or application, the ESB shall validate the XML according to the XSD for the service before transmitting the message. This validation shall be performed upon entry to the VA security domain and will not need to be repeatedly validated within the VA SOA security domain. No further ESB validation is required for messages generated internally using techniques that relied on the XSD. When the ESB converts a message from a non-XML format to XML, the XML shall conform to the appropriate XSD.

5.3.2. **Format and Transport Transformation**

A major function of the ESB is to perform format and transport transformations to allow the use of multiple transports.

5.3.3. Format Transformation

Format transformation involves the transformation of message contents between any of the allowable message formats (e.g., XML or native COTS messaging formats) without changing the syntax or semantics of the message. Even when both the provider and consumer are using XML there may be a requirement for XSLT transformations to convert the data from one XML schema to another, for example when the a composite services must deliver a response based on the results of multiple other XML based services.

5.3.4. Transport Transformation

The ESB will perform conversion between multiple message transports, such as HTTP, COTS messaging product etc. The transport transformation will allow the service provider and consumer to each select whichever of the allowable transports that they desire. The preference is for external services to conform to Web services standards using SOAP over HTTP. In addition to SOAP over HTTP, allowable mechanisms are:

- JMS.
- RMI.
- DCOM.
- Native COTS message formats.

Each service will support interface(s) to the ESB using one or more of the above mechanisms. Services are not limited in the number of the specified mechanisms that they can support. The ESB will perform the transformations between the chosen mechanism and any of the other allowed mechanisms.

5.4. Logical Model Component Descriptions

The VA ESB and supporting infrastructure consists of the following major system components:

- Security Gateways.
- Interfaces Support Services.
 - Device Interface Server.
 - Presentation Server.
 - Interface Server.
- Services Support (Web Services and Non-Web Services).
 - BPEL System.
 - BAM System.
- COTS Messaging System.
 - COTS Messaging Engine.
 - COTS Message Broker.
- SOA Supporting Infrastructure.
 - ID Management.

- Lightweight Directory Access Protocol (LDAP).
- Logging.
- UDDI Directory / Service Repository.
- SOA Runtime Management.
- Business Logic Services

In addition, to the VA ESB components identified above the VA ESB will make use of a number of VA shared infrastructure components. These include:

- Firewalls.
- Communications Network Switches.
- Management Network Switches.
- Storage Units.
- Storage Area Network (SAN) Switches.

Since these additional components are not integral to the SOA or part of the ESB, and therefore not described in this document.

The VA ESB is used primarily to expose services to internal and external systems and not to manage the user populations of VA systems. Application workstations will be attached to the applications initially and will be attached through presentation services running in a portal later as the application is fully migrated to a services environment.

5.4.1. Security Gateways

Gateway systems are the systems that provide the interfaces between the components inside the Security Domain and those outside the Security Domain – either external to VA or in other VA security domains.

The security gateway services provide gateway services for XML (Web) services interfaces, external systems interfaces, and user workstations. The gateways allow central definition of policies that govern Web services operations such as access policy, logging policy, and content validation, and then apply these policies to service requests entering the gateway. They also collect monitoring statistics to ensure service levels and security, and displays them in a web dashboard. The security gateway is used for the Web services interfaces as well as for the special legacy type (i.e., non-Web service) interfaces.

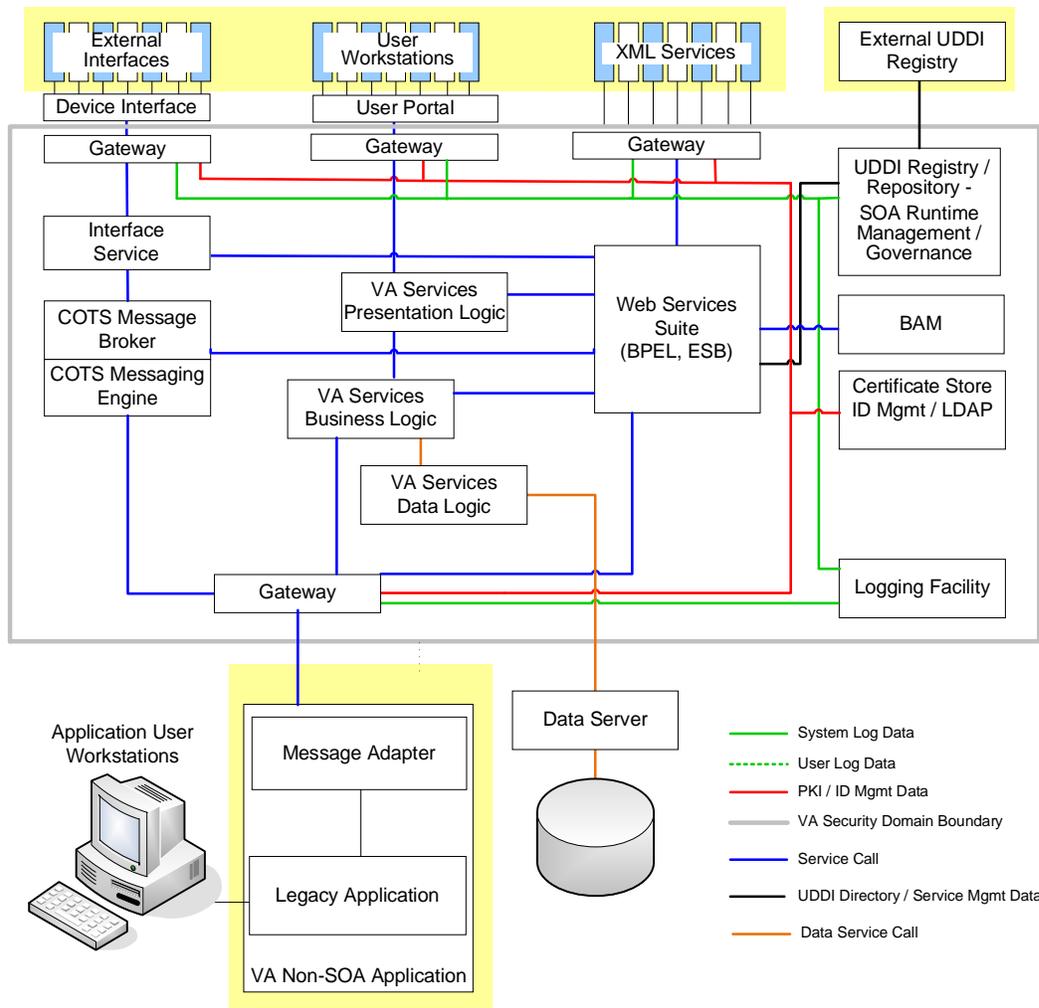


Figure 9 - ESB Logical Model

The security gateway needs to be implemented as a wire speed appliance that performs a wide variety of message filtering and message processing functions. The appliance will be used for each of the non-Web services interfaces between the ESB security domain and external entities. It is capable of providing any-to-any transformation and can parse, validate, and transform arbitrary binary, flat text messages as well as many industry standard and COTS message formats and multiple synchronous and asynchronous transport protocols¹⁶. It also provides integrated message level security, including message-level authentication and access control functionality. Authentication is performed by digital

¹⁶ Although the security gateway is able to perform message transformation, most of the format transformations will be performed by other components of the architecture. The security gateway must be able parse incoming messages to the extent necessary in order to perform certain logging and content based routing and logging functions. Semantic and syntactic transformation will not be performed in the ESB infrastructure, but by the applications that communicate across the ESB.

signature verification. Access control functionality is provided by ACLs that are maintained and used by the security gateway. Messages will be filtered, validated, encrypted, and signed. The appliance provides sophisticated multi-step message routing, filtering, and processing as well as detailed logging and audit trails.

The security gateway is also used as the gateway between the ESB security domain and the other VA security domains and VA applications. This gateway will be used to provide logging of service requests sent to, and received from the business area SOAs.

The underlying SOA architecture that this infrastructure implements is based on the use of security domains so that messages can be authenticated once they enter the domain and do not need to be authenticated at each step in the processing (i.e., where security domains such as described here used each service in an orchestration might need to individually authenticate each user). Therefore, external messages will either be digitally signed by the external sender and verified by the security gateway, authenticated by an interface process outside the gateway and digitally signed by that process and authenticated by the security gateway, or for Web services – be authenticated at the gateway using the Security Assertion Markup Language (SAML) and then being digitally signed by the security gateway.

5.4.2. Interfaces Support Services

5.4.2.1. Device Interface

The Device Interface is only when required for non-TCP/IP based communications or where non-standard interface devices are required to support an existing interface. The device interface is used to support the existing interfaces – in whatever form or format that information is currently transferred over those interfaces. The Device Interface will perform those current authentication processes and / or use those current authentication mechanisms as currently exist. The authentication may be that the message has arrived over a particular dedicated line or because the message arrives in a particular queue. The device interface is used because of the peculiar nature of the external interfaces. The Device Interface will need to log the record that was received and modify the record to include user identification information, such as the queue from which it was received, the dedicated line over which it was received, or whatever other information that it might acquire as part of the message acquisition, interface, and authentication.

5.4.2.2. Presentation Server

Presentation logic will be required to support workstations and other end user devices. The presentation servers will be outside of the VA ESB security domain. The presentation server will act as a web server for the ESB attached workstations and provide a web browser interface. The web browser interface will display web pages for the browser will provide user logon services, user authentication services, and provide users access to services that they are authorized to access. The presentation server will also provide the user the ability to use his personal certificate to sign service requests.

5.4.2.3. Interface Server

The Interface Server is inside the VA SOA security domain and is used to expose external interfaces as services to be made available to authorized users throughout VA. The interfaces exposed as services will use VA standard message compliant data elements. The native interface to VA standard data element conversions will be performed by code provided by the interface owner. The Interface

server will pass through the native interface messages to the system that owns the interface. The Interface Server will be a business logic server that may participate in service orchestrations.

5.4.2.4. Web Services Support

Web services are used to provide a services interface for external users.

5.4.3. BPEL / “ESB” System Servers

This segment includes ESB, BPEL, and BAM products and provides the core capabilities for the VA SOA infrastructure.

5.4.3.1. BPEL Processor

The BPEL processor provides the basic BPEL processing. The ESB has no local, permanent data storage; the data is stored in the Data Layer in the VA Application Architecture and will be access by data services accessible in the ESB. The BPEL engine will be the core of the SOA capability. The BPEL processor will provide one of the primary SOA capabilities.

5.4.3.2. ESB Engine

The ESB capabilities include multi-protocol transport (JMS, native, HTTP, File/File Transport Protocol (FTP), SOAP, Representational State Transfer (REST), etc.), routing, transformation, connectivity to back-end systems (adapters), standard – XML, Web services, WSDL processing, XSLT and XSD processing. It is used in conjunction with the BPEL Processor. The ESB provides many of the transport and syntactic conversions – particularly to and from Web services (i.e., XML and SOAP over HTTP).

5.4.3.3. BPEL Dehydration Database

The BPEL Dehydration database is used to improve the efficiency and effectiveness of the BPEL processing. Because some of the BPEL orchestrations can run for very long periods of time, there is a requirement to store process state in a persistent data store. The dehydration database stores this BPEL orchestration and state data in binary XML allowing rapid serialization / deserialization. This data will be stored in a relational database. This is required even though services are to be stateless. As normally used, a stateful service would maintain state from service to service and state would be visible outside of the service. In the case of the BPEL Dehydration Database, the state being referred to is the internal state of the service. For example, if a service orchestration issues service requests to a number of other services, the internal state would describe which of these subsidiary service requests have completed and which are still awaiting a response.

5.4.3.4. Business Activity Monitoring System

The Business Activity Monitoring System (BAM) has two components, a processing component and a database component.

5.4.3.4.1. BAM Processor

The BAM process performs the data collection and processing based on data collected from monitors imbedded in the BPEL engine used to run the orchestrations. The BAM processor provides information on business process performance that cannot be captured by traditional performance monitoring tools.

5.4.3.4.2. BAM Database

The BAM Database holds the data collected by the BAM processor.

5.4.3.5. COTS Messaging

Several components of the system deal with processing messages and services requests through COTS Messaging products in other than Web services formats. The ESB Server is able to handle the processing of service requests where the source and destination both use any one of these COTS messaging products. A separate COTS messaging path has been provided to allow currently exposed messaging queues to continue to be used and allowing those queues to be exposed as services.

5.4.3.5.1. COTS Messaging Engine

A native COTS capability is provided in addition to the messaging capabilities provided in the core Web services stack because a large portion of the current message traffic both will be COTS messaging based. The native facility is to provide a more efficient path for service requests and messages that only require transformations between COTS messaging systems and for which at most only format translation needs to be performed. A separate messaging engine may be needed for each COTS format that is natively supported.

5.4.3.5.2. COTS Message Broker

The COTS Message Broker will support the processing of COTS messages and will allow the ESB to most efficiently handle MQ requests and to perform format transformations of COTS messages. These systems will allow incorporation of existing COTS message queues into the ESB and to allow orchestrations to access existing COTS message queues. The COTS Message Broker will perform transformations between the various COTS formats messages. The COTS Message Broker will not be used to perform orchestrations as orchestrations are performed by the BPEL processor.

5.4.4. SOA Supporting Infrastructure

In addition to the “ESB” capabilities specified in the VA Service Oriented Architecture Framework, there are a number of other capabilities that must be provided to allow the VA SOA to be instantiated and to operate. These capabilities are described in this section.

5.4.4.1. Identity Management / LDAP

Initially the ID management functionality will be provided by a simple LDAP directory. While this capability is not as rich as would be provided by a full identify management solution, it will suffice initially. Identity Management requirements will be kept to a minimum initially due to the fact that service requests will primarily be from legacy applications and not individual users. Since the LDAP will be replaced with a more robust identity management solution, only the minimum required information will be stored in the LDAP.

5.4.4.2. Logging Servers

There are two logging services, event logging and payload logging. The event logging service buffers records on the security gateway, which are then off loaded either on a size or time basis to persistent store via secure ftp. Payload logging, when it is required, is performed by the security gateway appliance. The payload logging service sends messages to an external logging service. This service is configurable to rotate log files (i.e., create new files) either on a size or time basis.

While the logging servers will be capable of full payload logging, the amount of data that this could generate should limit the amount of payload logging that is performed to short time intervals, infrequently used services, or services whose messages have a very limited payload – or all three.

The logging service will receive XML formatted requests from logging clients that will be sent via a transport that supports reliable and persistent message delivery; and store them in a database table. The XML requests will have a limited number of mandatory fixed elements and may include a variable number of optional elements. The format of the variable elements can be different for each message / client. The logging service will transform the XML request sent by the clients into a fixed format which matches the log database schema using the XSL style sheet whose name is specified in the XML request.

5.4.4.3. Directory / SOA Governance Repository Servers

The UDDI directory will provide a services directory to allow services to avoid hard coding of service URLs and to allow multiple versions of services to exist at the same time. This component will also store service metadata to support SOA governance functions. While much of the information in the Repository will provide operational support to the SOA, additional information in the repository will deal with the overall management of the SOA, such as information collected during the development cycle of the service. The UDDI directory, metadata repository, management data will all be supported by the single server or set of servers. While UDDI is a Web Services directory and not all services will be Web services, all services can be described in XML and entered into a UDDI directory.

5.4.4.3.1. Internal UDDI Directory

The UDDI repository provides a machine readable directory which stores the location of all services so that their location does not need to be hard coded into services or application. This allows the location of services to be changed, multiple versions to be supported concurrently, and user to be seamlessly transitioned from one version to another. The UDDI capability is standard and a commodity. The true added value is comes from the repository functions. Separate instances of the UDDI directory will be instantiated for the development, test, and production environments.

5.4.4.3.2. External UDDI Directory

The External UDDI directory will not be inside of the VA ESB security domain and will, not be used by VA systems or services. The External UDDI Directory will contain meta information regarding services that are to be made available to users external to VA. It is outside of the ESB security domain as it will be directly accessed by systems external to VA. The External UDDI Directory will only contain information on services to be made available to entities external to VA. It will not contain any information on any services that are not to be exposed external to VA.

5.4.4.3.3. SOA Repository

The SOA repository will contain artifacts related to the design, development, testing, management, and governance of the services. The SOA repository will store all metadata related to the services, as well as policies, performance management information etc. for services. It will serve as the single integrated store for all information about the services. The SOA repository will also serve as the data store for services SLA and performance data.

5.4.4.4. SOA Runtime Management Server

The Runtime Management Server provides a variety of runtime services that support production operation of the ESB and provides management for deployed services. The Runtime Management Server allows creation and publishing one or more managed endpoints for each service, creating special purpose endpoints for each type of usage (e.g. secured and unsecured), load-balancing across

endpoints to ensure high availability, providing failover across endpoints to ensure fault-tolerance, and monitoring each endpoint for instantaneous health and availability.

The Runtime Management Server also provides non-intrusive evolution of production systems by allowing publication of multiple versions of the same service simultaneously, transparent rolling upgrades to published services, ensuring backward compatibility for new versions, version-based routing of requests to services, and manual or scheduled version deprecation. It also provides auto-enforcement of policies on new services and auto-provisioning of policies.

5.4.4.5. Business Logic Server

The business logic service processor(s) are provided to allow for processing non-BPEL service code. It is highly likely that some services will need to perform functions that cannot be handled exclusively through the use of BPEL, XSLT, XSD, and XPath and will require code to be developed. The business logic server will provide an applications server to allow the development of J2EE applications code that can be orchestrated with other services. If there is sufficient demand a .Net business logic server would be provided.

5.4.4.6. Data Logic Server

SOA level data services – the data services that reside at the SOA level of the VA Enterprise Application Architecture. This server will provide SOA services access to data, but will not actually store the data. There will be no data storage inside of the ESB Security domain.

5.4.4.7. Data Server

The Data Server is not a part of the ESB and is located outside of the ESB security domain. Access from the data logic server, which is within the ESB security domain, to the data server will be through either an isolated VLAN or isolated physical network segment. There will be no user access, application access, or external access to the data server. Since the data server will be on an isolated network segment or isolated VLAN there will be no requirement for further authentication for the service access to the data.

5.5. Operational Overview

The previous section described the components that comprise the ESB and its supporting infrastructure. This section describes some aspects of the operation of the ESB and the infrastructure required to support those operations and management.

5.5.1. Synchronous versus Asynchronous Services

Some service orchestrations may be able to operate in a completely asynchronous manner with no regard to the order in which service requests are received, processed, or completed. Some applications underlying other services may require that services be performed or completed in a specific order.

5.5.1.1. Asynchronous Services

There are service requests that are asynchronous in that they have no inherent ordering in them and which therefore may be received and processed in any order by the ESB without ill effects. This occurs when the service requests are unrelated or if they are related have no dependencies. For example a Veteran may request an educational benefit and arrives at a VAMC for treatment. Although they relate to the same Veteran, there would be no dependencies between the request for an

educational benefit and the hospital admission and therefore, the ESB could process the service requests in any order without causing an issue.

5.5.1.2. Synchronous Services

Synchronicity can be expressed in two ways. First, the service may be composed of one or more requests – response pairs in which the service makes a request and waits for a response before making another request. The ESB will support this type of synchronous processing.

In other cases, the synchronicity may express an arbitrarily long sequence of requests in which the order in which services are performed is critical and the time over which this sequence is executed is arbitrarily long. For example, consider three service requests – one to create a record, one to update it, and then another to update the previously updated information. If the two updates are to different fields then there may be no issue, but neither can be processed prior to the initial request to establish the record. However, if the two updates are to the same field there is a serious problem. This problem cannot be resolved in the ESB as the required prior request could have come an arbitrarily long time ago, requiring that the ESB would not only need to know what updates were required to happen sequentially, but would require the ESB to maintain a full history of updates to know which have been applied. Since the requests may have come from separate service requesters, there can be no assurance that the requester can handle the problem either. In the interim, in which there are a mix of services accessing and updating data, the ESB may not even be a party to some of the transactions in the sequence.

Therefore, the ESB will support the first type of synchronicity but not the second, since it will be the service (ultimately the application of which the service is a part) that will need to control the flow and ensure proper sequence or service requests – not the ESB. It may do this by a dialog with its users, assigning serial numbers to requests for updates to ensure that they are posted in the correct sequence or it may request that each service requestor include a unique requestor ID and a locally unique sequence number to each service request or some other mechanism. Service developers need to be aware of this issue and develop the underlying called services with this in mind.

5.5.2. Service Failover

Application servers will be configured in an n+1 configuration where n instances of the server are configured for each function to meet capacity requirements. The additional server is used to provide a replacement for any failed system in the functional group and to allow the system to continue to provide the required capacity and to meet performance requirements. A pool of additional application servers will eventually be provided to allow additional surge capacity if so required. The n+1 servers will be configured in a cluster using a clustered file system so that all servers in the group share a common file system so data available to one is available to all. Such a clustered file system will store all of the data so that failure of a processor does not cause loss of data. As a result, each of the individual systems will be configured in a diskless configuration.

Recovery in the event of a failed service will be to the beginning of the service request so full redundancy (i.e., HACMP configurations) will not be required. If a service that is a part of an orchestration fails, that service request can be retried before the entire orchestration has to be failed. For asynchronous orchestrations, the BPEL engine stores state information for each of the executing orchestrations. BPEL calls this saving of process state dehydration and storing the dehydrated processes allows assignment of the orchestration to any orchestration server. This provides fault

tolerance for orchestrations and allows orchestrations to be resumed up to the point of the last service invocation on a processor other than the one on which it started.

5.5.2.1. Backup Strategy

Since there is no long term user data storage within the ESB, there are minimal requirements for user backup. Recall that the service data servers are outside of the ESB and thus, are not part of the ESB backup. They will be backed up as part of normal data center operations and backup. The only user data to be stored in the ESB security domain will be the state information for orchestrations and services while they are in execution (i.e., the information in the dehydration data server) and whatever temporary storage the services require.

Therefore, the data to be backed up will be systems related files and information (e.g., ID management data, BAM data, etc.), orchestration and service state information, logs, and service temporary storage.

5.5.2.2. Data Archiving Strategy

Except for log data there is currently no expectation that there will be any ESB data that needs to be archived. Since Internal Affairs (IA) will be the responsible agency for internal audit, the log data will be provided to IA and they will determine the logging and retention requirements for the data.

5.5.2.3. Operational Surveillance and Systems Management

Operational surveillance and systems management will be provided using the suite of systems management products in place at each of the production datacenters. Each processor and component provided as part of the VA ESB will be configured with an End Point for those systems management tools and will be operated and managed in accordance with established data center policies, procedures, and configurations. Further, the ESB runtime management server will collect services oriented operational information and will be integrated with the installed systems management products to allow the ESB components to be monitored and controlled through the same systems and screens that are used for other production systems in the data center. The ESB runtime management system will also require endpoints on each of the ESB services and SOA components.

6. Governance

An SOA requires proper governance of services since the overall success of an SOA effort may be more dependent upon management factors than technical factors, if only because the technical factors are better recognized, better understood, and there are multiple product and service vendors lined up to assist in the technical implementation. SOA mandates a connection between business requirements and IT capabilities through the power of abstraction, loose coupling, and composition. In order to make the business goals a reality, however, companies require governance to make sure they build and compose the *right* services in the *right* way. Providing these capabilities for SOA initiatives is particularly critical for the success of those initiatives. Without such governance capabilities, any SOA initiative will likely succumb to the "silo effect," where different groups within IT handle issues of management in separate ways, thus negating the benefits of an SOA. Another benefit of preventing the "silo effect" is that a centralized governance model can ensure that all uses of each service meets the same policy and procedural requirements such as security and privacy compliance requirements.

The use of a SOA within an organization is fundamentally different from the implementation of standard Web services to support external users. In the latter case, services are developed based on a

perception of the needs of the external users and the services are “published” so that they can be found by those potential users. Because internal users will be required to use the specified services, rather than developing their own capabilities, developing services for use by internal users requires that the services are defined so that the services meet all of the needs of those users.

An SOA is based on the concept that a service is developed once, by a single application, project or organization, for use across the entire enterprise. The cost-effectiveness of an SOA is based on developing services once and using them multiple times. Therefore, unlike Web services in the commercial market, where a service consumer may identify multiple providers of similar services, within an organization there should be exactly one provider of each service and that service should be used by all service consumers within the organization.

The requirement that there can only be a single service provided for each required function introduces a large number of cross-project dependencies between applications (projects) that provide services and those that consume services. These cross-project dependencies include functionality – what the services will do, schedule – when the services will be available and service level objective – how fast and available the service will be.

There are two approaches that can be taken to the implementation of services. The first is to allow any project or application to develop and publish whatever services it desires (or that it believes its customers desire) and determine what functionality those services will provide. All service providers publish their services and users discover those services and select the ones they believe are most appropriate for their particular effort. This essentially is the model for Web services popularized by the many “Web services” vendors.

The second approach, is to have a governance process in place that determines which services will be developed and published at the enterprise level, what they should do, who should develop them, and require their use by all internal applications. A centralized governance process also ensures that all services and all requested uses of services are reviewed for privacy compliance, so that all uses of personal information and all uses of technology that could impact an individual are privacy compliant. Further, business areas, MIs, and systems are not allowed to independently develop services (even for internal use) that duplicate functionality of services published to the enterprise level by another system. This SOA is based on the second approach.

The SOA Governance Process will need to be robust enough to identify the range of services that will be required, define the level of functionality to be provided by each service, develop a schedule for the implementation of each service, and identify who will implement each service. Since a service will need to meet the requirements of all enterprise users, the needs of all enterprise users will need to be identified and considered. In addition to being concerned about the functionality of the services, projects expecting to use services will be concerned with the scheduled delivery dates for those services. The schedules for the development of the service consuming application and the scheduled delivery date of the service by the service providing applications will need to be tied together.

Determining who will develop specified services might be problematic. There are many VA systems that collect overlapping information and perform overlapping functions. A key goal of the SOA will be to allow components to rationalize and modernize those systems without impacting users of the information. Therefore, it is likely that the requirement will be for one service to accept and record that data (even though it may then distribute it to multiple current databases) and one service to perform the processing (even if it must query multiple services). Since each of these two services

integrate data across multiple legacy systems, a determination will need to be made as to which, if any, of those systems should be the implementer(s) of those services.

Once a service is developed, issues arise as to who has the authority to change the functionality of that service. Questions also arise as to whether consumers of the service can request changes; which if any of the customer requested changes to the service will be implemented by the provider, and who will pay for the changes? Can the provider change the nature of the service, and when can changes be made? How should the QA process ensure the integrity of the changes? The governance process must resolve each of these issues.

6.1. Use of the OneVA EA as the Basis for SOA Governance

Agencies are already being required to put an enterprise-wide governance process in place for their EAs. The OMB and Government Accountability Office (GAO) maturity models for EAs measure integration of the Application Portfolio Management, Enterprise Architecture, and capital budgeting processes. The maturity models also require senior management signoff of the architecture and strong involvement of the business units in the architecture. Furthermore, the EA is already required to include service components that forms the basis for the services built in response to the SOA, so very high level services – or at least service areas are already defined in the EA. In theory, an entire SOA governance structure could be developed parallel to that already put in place for the EA. However, given the level of difficulty involved in getting the EA governance process in place, such a duplicate effort is unlikely to yield positive results.

The EAs already tie required business functionality to projects and applications. It can be used to assign service functionality to projects and to mandate the scope, functionality, and schedule for those services. The ties between the EA and the Capital Planning and Investment Control (CPIC) process should ensure that funds are available for implementing the services if they are required by the applications. If the EA is used as the basis for the SOA governance, changes to the definition of services become changes to the EA and can be handled in the standard manner for EA changes. Therefore, the current EA configuration management, change management and other governance processes can be used for the corresponding SOA requirements.

Another factor in SOA governance is that the process must be kept relatively simple so that the cost of developing and exposing a service is on a par with the cost of defining, documenting and implementing an interface. When there is a requirement for one application to access data owned by another application, there should be a strong incentive – or at least the removal of a cost disincentive – to develop a reusable service rather than developing a custom interface. While development of the service rather than the interface will be mandated, the approach will be more widely accepted if it does not incur substantial additional costs. The costs of services can be minimized when the interfaces are required to conform to a set of standards.

If, rather than allowing a service consumer to request any set of data elements as part of service definition, – which then may need to be renegotiated for the next potential service consumer – the service provider is only required to expose entire data classes from the Enterprise Logical Data Model (ELDM), the cost of negotiating the interface becomes insignificant. If the data class contains too much information, then the data class can be decomposed and those changes incorporated into the ELDM and the EA. The only factors that need to be negotiated will be SLAs for the services.

6.2. Portfolio Management

In a stovepipe environment each VA element – VA business area, MI, or project etc. would make the decision as to what systems or services it would develop based on its own priorities and resources. As the OneVA EA becomes more and more a part of VA IT planning, this SOA places much higher emphasis on common business and infrastructure services, and the CPIC process pushes VA more towards a portfolio management approach to systems development. VA portfolio management processes as they mature will be the base for the management of SOA services across the VA enterprise and deciding which enterprise services will be developed and who will develop them.

6.3. Development, Specification, and Modification of Services

Services to be developed as part of the SOA will be documented as part of the OneVA EA. This will include the service contract that is the basis for the service. The OneVA EA will map the SOA services to the service areas and service components. Initially the OneVA EA will contain lists of high-level services that are mapped to the respective service areas. VA level services will be assigned to applications and those high-level application services will be assigned to projects for development. The projects will decompose the high level services provided to a level such that each resulting service can be implemented by that project.

Since all exposed services are documented in the OneVA EA, any addition of new services, modifications of existing services, or deletion of old services are changes to OneVA EA and thus are considered to be changes to the OneVA EA. Processes and procedures for proposing, evaluating, approving, and implementing changes to the OneVA EAs currently exist.

6.3.1. Services and Agile Development

VA is moving to an environment in which an Agile approach is used for all application development efforts. The Agile approach emphasizes larger numbers of smaller increments resulting in very frequent delivery of small increments of application functionality. The SOA services that are assigned to a project will need to be decomposed to a level that each service is sufficiently fine grained to be developed in a single incremental release.

One of the issues with using Agile development techniques is the identification of useful, independent pieces of application functionality that can be implemented in a single Scrum. The development of services is ideally suited to the Agile approach because services are by definition small, independent units of functionality. Each Scrum will implement a set of services.

6.3.2. Criteria for Determining the Level at which Services are Developed, Hosted, and Published

There are four fundamental types of services that will be developed:

- Presentation Services
- Data Services.
- Functional Services.
- Infrastructure Services.

Not all infrastructure, integration, and interfaces will be able to be exposed as services, because they may comprise the infrastructure on which the core services are built. There will be differences in how each of these types of services is developed, where they are hosted and the level to which they

are published. Each of the service types is described in more detail below. Note that even though a service may be published at the VA level for use by all applications, only authorized users or systems would actually be given access. Individual users or application systems would need to be authorized for access to the services or data.

6.3.2.1. Presentation Services

Presentation services can be developed as relatively standard services using the ESB, and some of them will be developed in that manner, and these services will be an important part of the presentation layer. However, it is expected that much of the presentation will be implemented as JSR168 portlets that are essentially portal services that communicate between themselves in much the same ways that business logic services communicate within the ESB. JSR 168 Portlets –enables interoperability for portlets between different web portals. This specification defines a set of APIs for interaction between the portlet container and the portlet addressing the areas of personalization, presentation and security.

6.3.2.2. Data Services

Data services are used to expose data classes from the VA ELDM¹⁷ to the rest of the enterprise. That the data class (entities) is in the ELDM implies that the data elements (attributes) in the class are of interest across the enterprise, and thus, must be exposed across the enterprise. Data that is strictly of interest within an application would not be included in the ELDM. Therefore, data access services that expose data described in the ELDM will need to be published at the VA level. The Technical steward for the “authoritative instance” of data (see Section 7 Data Architecture) shall provide specified standard services (e.g., read, update, create, delete, subscribe) as appropriate, which will be published at the VA level.

Data services exposing sets of data elements that are not a class in the ELDM should not be exposed at the VA level because, by definition, these are not data classes of interest outside of the application. The data services will need to be developed by the steward for the data class and be hosted close to the data store.

6.3.2.3. Functional Services

Functional services are provided by applications and / or application developers. These services can either be developed for VA-wide use or for use exclusively in the application. Any service required by one or more applications other than the application that developed that service must be published at the VA level by the application that has functional responsibility for the service. There are four instances in which these services are published at the VA level.

¹⁷ It is recognized that at the current time the VA ELDM is not sufficiently mature or complete to support the usage described in this section. The intent is first to ensure that the stewards of enterprise data make that data available across the enterprise, but second, that the steward of that data not need to support multiple requests for similar sets of data. The notion that data is shared at the class level is intended to allow sharing of all data, but to provide it in standard “chunks” so that there were not a multitude of services providing only slightly overlapping sets of data.

- **VA Mandated Services** – VA may identify a service and assign it to an application for development to be exposed at the VA level. One special class of VA mandated services will be the Core Common Business Services (CCBS).
 - **Core Common Business Services (CCBS)** – These services are rather coarse grained business services that perform major business functions that may be required across VA. Examples of the CCBS include:
 - Reporting.
 - Case Management.
 - Workflow Management.
 - Correspondence.
 - Adjudication.
 - Finance.

While many of these functions exist in multiple legacy systems, modernized, service-based implementations of these functions will be performed only once with the services that are developed being made available across VA. Each CCBS will be assigned to a single project to implement for the enterprise.

- **Application Requested Services** – An application may request a service that is in the functional purview of another application and therefore, will need to be developed by that other application. Since that service will be required by an application other than the application that developed it, that service must be published at the VA level for all applications to use.
- **External Interface Service** – Where an application has established an interface with an entity outside of VA then the service must be exposed as a VA-wide service, published at the VA level.
- **Application Published Services** – Any application may publish at the VA level any other services that it develops that are within its functional purview. It is wholly within the application's discretion to submit an EA change request to the EA to publish such services at the VA level and have them incorporated into the OneVA EA. However, once published at the VA level, the application may not then restrict access to the service except by going through the EA change process to delete this service from the OneVA EA. An application may not develop or publish any service that is not in its functional purview, but must request that the application with the functional responsibility for the service to develop the service.

6.3.2.4. Infrastructure Services

In this case infrastructure includes not just the typical infrastructure services that are required to support the operation of systems and services, but functional infrastructure – those services that would underlie the CCBS as well. These services are of two types. The first are those identified as being required by VA and which is either developed centrally and made available for use throughout VA or for which a VA element or project is assigned responsibility for developing the capability for VA-wide use. These are the Core Common Integration Services (CCIS). The other type is a

capability that is developed by a project for its own use, but that is of more general use throughout VA and is made available for more general VA use.

- **Core Common Integration Services (CCIS)** – A number of services (e.g., ID Management) that perform infrastructure functions not business functions may be identified. Once these services are identified, they may be assigned specific project, MI, or business area for stewardship and / or the development of specific services to be published at the VA level or assign a project, MI, or business area as the “VA Lead” for the development of a capability.
- **Project Developed Infrastructure and Integration** –In addition, a project may be the first to develop an infrastructure, integration, interface service or capability that will be required across the enterprise. In these instances the service is made available at the VA level. In general, where feasible, each infrastructure, integration and interface service, should be developed once and used by all applications.

6.4. Systems vs. Services

The discussion to this point has discussed SOA services in terms of the development of VA applications. The development of services is well suited to the Scrum / Sprint–based Agile development approach being used by VA. However, these approaches are oriented towards the development of functionality as part of a larger application system. Services developed – even within a sprint, will typically be developed within the boundaries of an application system, but not all.

6.4.1. System Based Services

It is expected that virtually all services will be developed in association with an existing Federal Information Security Management Act of 2002 (FISMA) system or a new system that will be subject to FISMA requirements and the services will be included within the C&A boundaries and the C&A of that system. For the most part, services are considered much the same as new interfaces or other increases in the functionality of an existing system or maintenance changes to that system. This will be true even in the case that an independent project is chartered to implement the service. The documentation required for a new service will be the standard life cycle documentation that would be required to a comparable non-service based change to the system.

6.4.2. Non-System Services

While most services will be developed as part of an application or a project, some may not be. Services can be developed and exposed that are merely orchestrations of other existing services, without adding any of their own organic functionality. The orchestration may, in effect, integrate services from each of several independent FISMA applications. Since developing these services will be much simpler than developing fully functional services, these services can be developed outside the bounds of a full project or outside the bounds of an application. These “singleton” services have a number of implications for the SDLC, for security, for privacy, and for services management. Since these services are being built outside of an application, they are not part of a system design that goes through gate reviews and are likely to be able to escape many of the management controls that services developed as part of a project would be subject to.

6.4.2.1. Security Considerations

The “singleton” services also pose a security issue. Individual services that are part of an application do not require their own C&A, as they could be viewed as a small modification to systems that already had a C&A and they were running on infrastructure that itself had be C&A’ed. These non-

application, singleton services, while they will be running on C&A'ed infrastructure will not be included within the boundaries of a system that had been C&A'ed. It will be a determination of the Security Policy group the extent to which any of these systems can be included in an existing C&A or the extent to which a new C&A is required.

6.4.2.2. Privacy Considerations

The privacy considerations for these non-application, singleton services are more complex than the security considerations. While most VA systems will have developed the required privacy documents so that services developed as part of those applications do not need to individually develop such documents, those services that are not part of an application will need to produce their own privacy documents. These privacy documents are required anytime a "system" creates a "new collection of information." SOA services clearly qualify as "systems" under the definitions of the in the privacy legislation. A service that combines information from multiple existing systems would almost certainly create a new collection of information unless perchance that collection of information had been assembled previously.

- **Privacy Threshold Assessment (PTA)** – The Privacy Threshold Assessment (PTA) is an assessment that is used to determine whether further privacy assessment or analyses are required. The PTA consists of five (5) essentially yes / no questions that are used to determine whether a system contains any personally identifiable information. All services are "systems" within the context of the definition used in the PTA, and since most VA systems contain information about people, most VA systems would meet the threshold.
- **Privacy Impact Analysis (PIA)** – A PIA is an analysis of how personally identifiable information is collected, stored, protected, shared, and managed. The purpose of a PIA is to demonstrate that system owners and developers have consciously incorporated privacy protections throughout the entire life cycle of a system. The PIA for each service should include a description of the data that would be used by that service and enough of a description of the other service that use the data to enable tracking of all services that make use of that same target data.
- **Systems of Records Notice (SORN)** – The System of Records Notice requirement is driven by the Privacy Act of 1974 and is generally required when any collection of personally identifiable information exists and is accessed through a personal identifier. There are more detailed requirements and specific exceptions to this requirement. In general, if a service uses personally identifiable information in a way that retrieves that information by personal identifier, a SORN may be required.

6.5. The Effect of the SOA on Cross-Project Dependencies

It has always been a goal of the OneVA EA to foster greater reuse of existing services that reduce cost and maximize application efficiencies. However, today's code and / or system functionality are rarely reused, and each stovepipe application develops all of its own functionality with interfaces to other applications only when absolutely necessary. Functionality is often duplicated and not reused. The direction that code be reused becomes much more explicit in an SOA where a service is only developed once and all users requiring that functionality consume that service rather than developing their own functionality. Applications are not given the freedom to develop their own copies of a service, but rather are directed to use the common instance of the service.

While promoting reuse of services can reduce design, development, test, deployment and ongoing support costs in the long term, these benefits come at a cost. Leveraging functionality across projects creates dependencies among those projects. The development schedule of a project now depends upon the services that it expects to have been developed by other projects. Problems or delays in one project may manifest themselves as delays in providing services to other. Proper controls and processes at the VA level, beginning during the governance process, will mitigate the negative effects of these dependencies.

6.6. Project Responsibilities

High-level services will either be assigned to business areas, MIs or projects or identified by business areas, MIs or projects for implementation. High-level services and service areas assigned to projects will conform to the areas for which the projects have functional responsibility. Projects will need to treat the requirement to provide service interfaces or service facades for its existing systems as requirements and must plan for funding as well as implementation of those services. However, since services providing new functionality are likely to be a much higher priority for the application that will be consuming the service than for the one implementing the service, service consuming projects might need to either justify funding or provide development resources for development of the service. It is assumed that as part of the service governance process there is a process to ensure that applications build the services that they are required to expose in a timely manner. The schedules for building these services shall be adjudicated at the VA level.

Another issue arises in which a project is to develop a service for another project, but that the service is much more important to the service consumer than to the producer. If the producing project develops cost or schedule issues, the tendency will be for the project to delay items that are of lowest priority or importance to them. Governance mechanisms will need to assure that the producing product does not jettison services due to the lack of importance that it attaches to that service.

Projects will be assigned high-level service areas for which they will be responsible for the definition and development of services (e.g., workflow management) and will decompose these high-level services to a level that can be used to support business processes. The decomposition will need to be at a level that the decomposed service can be implemented within a single release – even the relatively small releases that are developed as part of the Agile development process that VA is adopting.

Applications may request changes (add new services, modify existing services, delete existing) to the list of services published at the VA level by requesting changes to the OneVA EA. Changes to the OneVA EA will be requested changes using the EA change management process the same as for any other EA change.

In the legacy environment, the organization that owns an application is responsible for all aspects of the application and is responsible for any required enhancements to the applications. The owning organization is also responsible for obtaining funding for application changes required by other organizations. There will continue to be a concept of a project owning a service in that it has responsibility for the functional area in which the process operates and is the technical steward for the data that the process requires to operate (see Section 7 Data Architecture). However, the responsibilities of ownership will be divided into a number of functions that may be performed by more than one organization. The following roles relate to the development of new services or modification to existing services:

- **Funding Provider** – Provides the funds for the implementation of the service. In the case of services to be published at the VA level, this may either be the project that will be building the service, it may be the project that requires the service, or it may be the project that requires the service providing the justifications necessary to ensure that the project building and maintaining the service has the required resources.
- **Steward / Owner** – Is functionally responsible for the service. In the case of a service providing an interface to external systems, this is the owner of the interface. It is the owner's authentication / authorization and logging / auditing requirements that are used for the service.
- **Builder** – Organization that actually builds the service. The choice of the builder may be based on the availability of development resources between the various organizations. The builder may be an application development project, a central development team, an infrastructure development team, or another VA organization
- **Maintainer** – Organization that provides support for the service once it has been developed.
- **Operator** – Performs the physical operation of the service (i.e., the data center and system administration)

6.7. Sources for Services

One of the significant shifts in thinking about SOAs is that they introduce the notion that business logic is not engrained in programming code, but rather in the declarative metadata that describes a service and how it interacts with other services. In essence, SOA advocates a movement away from code-centric development to configuration-centric composition. Business logic is contained in the orchestration of processes rather than in the control logic of applications. This can only occur if current business functionality is exposed as services that can then be repurposed or refactored.

This shift to metadata-driven development introduces new challenges to how organizations develop or repurpose services. Rather than dealing with traditional development lifecycles, the organization must now understand how to continually iterate the development of their services and therefore, their service metadata as well.

There are a number of sources for new services that can be used in the enterprise. These include:

- Development of new services from scratch.
- Development of service wrappers for existing legacy systems.
- Repurposing of existing services.

6.7.1. Developing New Services

There are two different types of assets developers must deal with when building services: the application logic that underlies a service interface and the service metadata – information regarding the flow of control through the service. Therefore, building business services “from scratch” really means developing the policies and other metadata that govern how the services work as well as the actual functional code that does the work.

One major difference between developing traditional applications and developing services is that no business process or workflow logic should be coded into the application, but rather this should be left for the metadata to handle. In an SOA implementation, security, audit logging, reliability, and

transport binding considerations should likewise be relegated to the service interface and policy contracts that surround a service, rather than to its underlying code.

6.7.2. Leveraging Existing Applications

Some services in the OneVA EA will be provided (in whole or in part) by the existing legacy applications. These services will be provided primarily (at least for the short term) through the development of adapters for legacy applications rather than through modernization of legacy applications. However, as legacy applications are modernized, they must be redesigned based on the development of services. This specifically applies to the development of new interfaces to existing applications, which must be built using the techniques outlined in the SOA.

New requirements for services can be met by creating new services and associated metadata, repurposing them from existing legacy systems, or refactoring legacy systems to create new services. The determination as to whether new requirements should be met through new development or the enhancement of legacy systems will need to be determined based on several factors. The decision as to the implementation approach shall be guided by long-term value as well as short-term cost and schedule. The implementation approach must be mindful of long-term plans for replacing or retiring existing systems.

Putting service interfaces in front of existing application logic is not all that challenging. Since adding service interfaces to legacy systems is straightforward, the real development work comes from defining service contracts and creating the process, security, management and other metadata that sit on top of the services themselves. Furthermore, this approach to service development is only applicable for fine-grained services where it's easy to control the functionality and scope of change for those services. Simply wrapping legacy systems with service interfaces is not sufficient for coarse-grained business services.

The OneVA EA will include design patterns for the development of service-based systems. These design patterns will also need to include patterns applicable to integrating legacy applications into the SOA as well as providing service wrappers to legacy applications. The ability to provide wrappers to allow legacy systems interoperability in the SOA should not be used as an excuse to avoid the modernization of legacy systems that are in need of major renewal (e.g., not just the development of a "new facade").

6.7.3. Repurposing Services

One of the major tenets of SOA is that services should be reused as broadly as possible to support the ever growing and changing set of business requirements. As such, over time investments in services should gradually shift from service exposure and fine-grained development to composition, focused on service reuse. The focus should be on identifying existing services in the enterprise, composing them into new processes and refactoring them as necessary to support new capabilities. Repurposing services may have significant security and privacy impacts. The privacy impacts may be especially severe as repurposing services may result in privacy related data being used for purposes other than which it was originally collected.

Thus, the significant question becomes which services should be built from scratch and which can be repurposed to meet the changing needs of the organization. The answer to this question depends upon the particular circumstances of the organization and the scope of their existing services. Many of the fine-grained services that will have been developed by early SOA projects will not be amenable to much refactoring, since they may have come from simple extensions of their legacy APIs.

By defining services in a top-down fashion by starting with an overall architectural plan, rather than building the services from the bottom up (e.g., by building fine-grained services as legacy wrappers) will better enable repurposing and refactoring of the services for new capabilities. Therefore, developing process-driven, coarse-grained, composite services will provide better flexibility and ability to extract continuing value from service investments.

Service metadata that defines security, policy, reliability and declarative business logic for interacting with services should be amenable to repurposing. Service metadata should be created once and repurposed as many times as possible to meet ever-changing business requirements. As such, the real development effort here is not one of creating services from scratch, but rather developing from the start the most agile, flexible services that can be repurposed to meet new needs of the organization.

6.8. Service Contracts

One major aspect of an SOA is the concept of a service contract or Service Level Agreement (SLA). A service contract may be thought of as fulfilling the role of the Interconnection Control Document (ICD) and Memorandum of Understanding (MOU) in traditional systems, as it specifies the connection between the service provider and the level of service that the consumer will receive. However, the service contract goes much further than the ICD, as it specifies all aspects of the manner in which the service will be provided and consumed, not just the flows of information across the interface.

A service is like a black box that is specified in terms of the inputs to the service, the outputs from the service and a transformation i.e., the functional relationship between the service inputs and the outputs. The service contract is a functional description of what the service does, but not how the service performs the transformation or provides the functionality. The user is not aware if the service provides the functionality by means of a single atomic service or a set of composite service, or whether that changes over the life of the system.

The transport used for the service will not necessarily be a part of the service contract, because the Enterprise Services Bus (ESB) will perform transport conversions (see Section 3.7 Enterprise Service Bus). The service contract is a specification of the functionality of the service, but should not specify the manner in which the service provider will provide the service. In fact, the service provider should be free to change any aspect of the way in which the service is implemented as long as the service contract continues to be honored. The details of the implementation of a service should be hidden from the consumer of the service and the consumer of the service should not make any assumptions nor have any dependencies on the precise manner in which the service is implemented.

However, in the VA environment the manner in which a service is provided may be very important in terms of who may access the data and the manner in which it may be used or distributed (e.g., does the service access or present PHI). Therefore, the service contracts will need to specify those aspects of how the service is provided that may impact the functionality of the service as seen by the consumer or which may place restrictions on what the user may do with the output of the service.

Web Services Definition Language (WSDL) is a standard form of a machine-readable service contract that can be entered into a directory and be discovered by service consumers¹⁸. Services contracts as used in this document go beyond the bare bones service contracts specified by the WSDL standard contracts which are designed to be machine-readable and serve as a mechanism for systems to locate and access services without human intervention. The service contracts as specified in the SOA will have a human-readable component for inclusion in the OneVA EA. Aspects of the service contract such as availability, performance, responsiveness and latency will be outside of the WSDL service description, if any. These aspects of the service definition will need to be included in the description of the service in the EA.

6.9. Service Level Agreements

Services are defined, approved, and developed based on a set of requirements. Those requirements include items such as the:

- Amount of the service that is required.
- Performance (response time / throughput).
- Times at which a service must be available.
- Frequency with which the service can fail.
- How quickly a service must be recovered / made operational after a failure

These items are specified in the Service Level Agreement (SLA), which specifies the terms of service under which service is provided and under which it can be consumed. These typically include the measures such as the expected workload and the reliability and availability for the service. This specifically includes aspects of the service included in SLAs including such measures as:

- **Input Volumes** – Identifies the number of service requests per unit time that will be made. Further, if there could be significant differences in the workload between service requests, then the service requests should be categorized with input volume estimates for each of the service request categories.
- **Performance Measures** – Specifies the promised responsiveness to the system is response to the service requests, either:
 - **Response Time** – For interactive requests where a user at an end-user device is the source of the request. Response time is the time from when the user hits enter or a service request is initiated.
 - **Turnaround Time** – Where a service is requested to perform a batch of actions as part of the service request. Turnaround is a measure of the amount of time taken from

¹⁸ WSDL is one of the many Web services standards. As noted, this document allows services that are not compliant with Web services standards. Even services that are not themselves Web services compliant can have a WSDL description that will be entered in a registry / repository. Virtually all registry / repository products allow the inclusion of WSDL descriptions for non-Web services compliant services.

the start of the first request in a batch to the completion of the last service request in the batch.

- **Availability** – Specifies that the likelihood that the service will be available during non-scheduled downtime periods expressed as percentage calculated as:

$$([\text{total time service is available during the period}]/[\text{non-scheduled downtime during the period}]) * 100$$

The VA IT Principles specify three levels of system *availability*:

- Mission critical (99.999% available),
 - Mission important (99% available), and
 - *Generally* important (95% available).
- **Reliability** – Is specified by two quantities:
 - Mean Time Between Failure (MTBF), and
 - Mean Time To Repair (MTTR).

MTBF describes how often a system or service fails, while MTTR describes how long it takes for a system to recover once it has failed. While reliability and availability are certainly related, a highly available system might not be especially reliable if the MTTR is low enough. A system that failed once per hour (not very reliable) but recovered in one (1) second (very good MTTR) would have a measured availability (assuming no scheduled downtime) of 99.97%.

The SLAs for composite services will be based on a combination of the SLAs of the first level services of which the composite services is composed. Each of these measures is critical to the ability of the service provider to provide the service as requested and for the service consumer to receive the service in an acceptable manner.

6.9.1. Setting SLAs for Services

Services will be of two types. First, there will be newly defined and developed services that are exposed at the VA level. The SLAs for these services should be based on the most stringent documented SLA requirements of any of the likely consumers. The definition of the requirements for the service should include the mandated SLAs, and resource estimates should include the resources required to meet the SLAs. Since these are approved requirements, there is a presumption that the VA funding process will ensure that required resources are made available in a timely manner.

Second, MIs or business areas may be directed or choose to expose existing application functionality as services at the VA level. Most applications operate under a set of, at least implicit, SLAs that were set as appropriate for their operational environment. However, when a system is providing one or more services to other systems, the concept of SLAs and setting the appropriate SLA becomes a much more complex issue. Production availability of one system may depend on services provided by other systems, and further, the service consumer may require more stringent SLAs than the service provider.

For example, VBA generally works office hours and it may only require systems to be available during office hours. If a VBA project were to develop a service for VA-wide use, VHA systems might require that service – if it were critical to providing care in a VAMC, to be available 24x7.

Unfortunately, there may be a cost impact to the system providing the service if it now must meet a much more stringent SLA than that for which it was designed. There may be issues related to the additional capacity, availability, and functional requirements imposed on the service provider.

When the service provider develops and publishes a services contract for a given service, the service provider will document the SLAs under which the service will be provided. When a service consumer requires services subject to a more stringent set of SLAs, the upgrade of the SLAs will be negotiated between the service provider and the potential service consumer. The result of that negotiation may be a requirement for funding of the service provider based on requirements and priorities of the service consumer. In such instances, the consumer will need to ensure that requirements are identified and justified at the VA level and that VA funding and capital investment processes ensure that the provider receives adequate funding

6.9.2. Multiple SLAs for a Single Service

There may be multiple SLAs for the same service. However, the cost of providing a service will depend on both the workload estimates for that service as well as the service level for the service. The multiple SLAs may be used to establish multiple service levels and to allow pricing based on quality of service. For example a premium price may be charged for a service with a “guaranteed” one (1.0) second response time requirement, and a lower rate charged for services that are performed on a “best efforts” basis i.e., with no response time limit. The only limitation is that all users who are similarly situated, requesting the same level of service should be subject to the same SLA and pay the same fee. Since there is a cost and level of effort associated with establishing and enforcing SLAs, there is a desire to limit the number of SLAs that are written and enforced. This can be done by allowing multiple users to sign a single SLA and only developing new SLA agreements when they are needed.

6.10. Managing SLAs

One of the major goals of the SOA will be to foster reuse of services developed by others and to support the operation of the services on common infrastructure. This, by its very nature creates inter project dependencies – where the ability of one project to deliver on its commitments is dependent upon the service to be delivered by other projects. The SOA infrastructure will be managed to ensure that SLAs promised to each service consumer are met and that one consumer’s workload does not impact the level of service provided to other consumers. However, there will be occasions where resources must be dedicated to specific functions even where that would result in SLAs for some consumers not being met. VA governance structures are currently in place that can lead to the prioritization of some work over all other.

6.11. Repositories, Directories, and Registries

Services descriptions will be maintained in both human and machine-readable formats. All services (other than those only accessible from within an application) will be held in a central registry and repository or repositories (e.g., one for machine-readable and one for human-readable). Access to the registry will depend on whether the user is in VA, DoD MHS, US Government, or is external to the US Government. Access to the registry will depend upon the user role (e.g., development versus production).

A “registry” is a place where official records are kept. It is an authoritative store of information that relates to a particular task at hand. Registries store metadata that relate to a particular asset without actually containing those assets. The store that actually contains those assets is the repository. So,

while a registry simply records official information that relates to an asset, the repository stores the assets themselves. While a registry might be the right place to store a Service definition and associated policy metadata, the repository is the right place to store service artifacts such as models, maps, shared keys, and transformational schemas.

Design time registries help developers locate assets, make decisions about which ones are best to use among many that might be similarly appropriate or adequate, and understand the various costs involved in their consumption. Runtime registries help systems make automated, policy-based decisions about service selection. However, there is no such design time vs. runtime delineation for repositories. A repository is simply a repository – to be used at both design time and runtime.

It is important to distinguish between two basically different concepts for the discovery of Web services. The first is essentially a manual, design time activity, where developers (or other people who want to find a particular service) search for information with some kind of service discovery tool. The second is basically an automatic, runtime activity, where a service consumer (which is a piece of software) has an indirect reference to the service it requires and looks up that reference in a runtime registry to find the service contract (e.g., WSDL file) that tells the consumer where to find the service and how to bind to it, in a manner similar to how the internet Domain Name Service (DNS) binds names of Internet addresses to the physical IP address of a requested server.

This manual, design time sense of service discovery is best handled by an asset management or metadata management tool. Such tools may support UDDI as part of how they provide for the discovery of services, but the information that might be contained in an UDDI-enabled registry is only part of the metadata that asset management tools must deal with. In other words, asset management tools are metadata management tools that may or may not support the UDDI standard, but in any case, must offer users far more information than what would normally be included in a UDDI registry. This includes metadata about Service levels, support for Service consumers, security and policy requirements and information about what the service definitions are, as well as how to use them. Asset management is essential to how people use an SOA to build loosely-coupled, coarse-grained systems composed of reusable service assets, but is only indirectly related to the runtime operation of the architecture. As part of this SOA that there will be a central VA directory containing information on all services available to VA systems.

In contrast, the automatic, runtime sense of service discovery is essential to the proper operation of an SOA. To understand why runtime discovery is so important, consider that a service consumer should query the registry every time a particular service is requested. The registry might conceivably return a different service contract from one request to the next. For example, a Service-Oriented (SO) Management application that incorporates an UDDI-based service registry might route a request to a different server to provide better scalability or availability, support different service versions or service levels, it may handle protocol translation, or may even convert a synchronous request to an asynchronous one.

In addition, service consumers may not be aware of security or policy changes that may require a change in the way that they access a given service. In these cases, there may be a service negotiation where service consumers communicate first with a registry to determine how best to access particular functionality. Regardless, all of this management activity takes place behind the scenes, in that the user of the service need be none the wiser about the details of the service contract or that the underlying service provider is different from one request to the next.

6.12. Service Discovery and Service Registries

In the Web services world, there is a concept of “service discovery” in which a user can go to a central location to identify which service providers are providing a service that meets specified requirements. This is done so that the Web service user can choose which service to access. Because of the sensitivity of VA data and the nature of its operations, VA’s approach to publishing services will be markedly different. First, VA will not allow unknown entities to connect to its systems or to browse for or access its services without specific authorization. This authorization should include ensuring that all security and privacy requirements related to service and the data that it exposes continue to be satisfied. All services provided to entities external to VA will be governed by an ISA and MOU with that entity. Second, only a single instance of specific services will be published so as to attain VA service reuse goals.

Any changes to services visible to external users would need to be publicized well in advance and only be implemented after due notice has been provided. It is unlikely that all users will ever be able to upgrade to new versions of existing services or to new services on the same date. Therefore, it will be likely that multiple versions of a service will need to be supported for some period of time. This implies that there would be a need for UDDI-based registry services to support external services even though service discovery would not be allowed in the traditional sense.

Just as in the case of the external users, a UDDI-based registry will be needed to support internal users as well. Although, a major feature of the SOA is that services are developed once and are reused throughout the enterprise, there will still be a need for a discoverable registry. While users requiring a specific service will find only one provider of that service, multiple versions of the service may exist. An UDDI-based directory will allow for the location (i.e., URL) or other characteristics of a service to change and for these changes to be hidden from the applications that access that service.

6.13. Service Versions

One of the precepts of an SOA is that the external definition of the service is defined and published and there is no visibility into the manner in which the service is implemented. While this is the ideal, it is not always the case that changes in the implementation can be hidden from the consumers of the service. For example:

- Users may have different levels of access to different data sources and thus the source from which the data is provided may impact what a specific user may see.
- As systems are being modernized what information is available and how it is processed may change. As these changes are made, the information that a service can provide and / or the processing that it performs may need to change which will require changes to the service.
- New requirements e.g., new laws or regulations may change the data that is available or the way in which it can be disseminated.

These are a few of the reasons why a new, incompatible version of a service may need to be produced and why there will need to be governance processes related to . However, it is not realistic to assume that all consumers of a service will be able to migrate to the new version of the service at the same time. Therefore, there will always be a requirement to support multiple versions of a service, but some questions remain related to providing those multiple versions of service. The issues related providing multiple versions of a service that must be examined include, but are not limited to:

- How many back versions are needed / must be supported

- For how long a period of time must the old version be supported
- How can service consumers be forced to migrate to the new service
- How much notice must be given prior to being able to make the change to the service without the consumers' agreement i.e., when can the service provider stop supporting the old one.

The answer to the above questions will likely depend upon the specific service, service consumer, and service providers.

6.14. **Publishing Services**

Services will be developed by application projects. These services may have been developed for the exclusive use of that application, or they may have been developed for use by:

- Other applications within the MI.
- Other applications within the business area.
- Other applications within VA.
- Applications in the MHS.
- Applications or users in other Federal agencies.
- Users external to the Federal Government.

A service is made available to applications by publishing that service in a VA level directory. Allowing access to the service by other Federal agencies or non-government users or systems is accomplished by publishing the service in an externally accessible directory. By not publishing a service outside of the application, business area or MI, access to the service will be restricted that application or to applications in that business area or MI. If the service is exposed to other business areas or MIs, it would be included in the OneVA EA and be governed by the OneVA EA change processes. If it is exposed to applications in a business area or MI, it would be included in the segment architecture and governed by its change process. And if the service is only exposed within the application, then it would not appear in any EA and would be governed by the application's internal change control procedures.

Services published at the VA level will include both those services designated in the EA as VA-level enterprise services for use by all applications and assigned to a single application for implementation. In addition, an individual application may publish additional services – assigned to it by the EA – at the VA level. The key will be to ensure that there is no redundancy between the services various applications publish at the VA level.

6.15. **Quality Control for Services Published at the Enterprise Level**

The OneVA EA will identify services to be made available to all applications and will direct a specific application to implement them. There will need to be an organization that ensures the functional correctness of the services. There will also need to be assurances that all required security and privacy reviews have been completed. The verification that the security and privacy requirements have been met and that all security and privacy reviews have been performed will be specified as part of the service development life cycle. This Independent Verification and Validation (IV&V) organization will ensure that all services published at the VA enterprise level meet their assigned requirements and SLAs.

7. Data Architecture

This SOA is based on the existence of a set of standard data element definitions that are recognized across VA. The VA ELDM will, once complete, identify all data classes containing VA enterprise data and include the definitions and metadata for each data element in the data class. The ELDM is part of the DRM and the Data Architecture. The ELDM will be the basis for VA standard messages and for the enterprise-wide harmonization of data across VA. This section summarizes some of the information contained in the OIT Enterprise Data Management Principles, the Data Architecture, and the DRM and then provides information related to the use and transfer of data as part of this SOA. This discussion is not meant to replace or supersede the information in any of those sources, only to collect the particularly relevant information in one place and to relate it to the SOA described in this document.

7.1. Data Ownership and Stewardship

For each data class in the ELDM there shall be one data class functional steward and one data class technical steward who will ensure the currency and quality of the data belonging to that class and support the sharing of that information.

- **Functional Data Steward** – The Functional Steward is the “owner” of the enterprise data represented by the entity in the ELDM. The “owner” of the entity is the business organization responsible for collecting the data and ensuring its accuracy, timeliness, and consistency. In addition, the functional steward is responsible for identifying the applicable security, privacy, records management, and records retention requirements – with records retention requirements subject to approval by National Archive and Records Administration (NARA). The steward is also responsible for designating the domain management rules which govern processes for creating and approving new, valid data values.
- **Technical Data Steward** – The Technical Steward is the owner of the database(s) or data stores used to store the data and the systems and services used to access the data. The Technical Steward decides on the physical storage of all VA data (e.g., if it is stored in a single or multiple databases) and ensures the enforcement of consistent data use by all application developers as designated by the functional stewards and that it is consistent and in compliance with applicable security, privacy, records management, and retention requirements – with records retention requirements subject to approval by NARA.

The following are the primary principles on which the data architecture is based and which permeate the SOA:

- **Data should be shared at the Data Class (Entity) Level** – The technical data steward for each data class in the ELDM shall provide a service that returns all data elements in that data class. Additional services may be defined, subject to the data class steward’s approval, that return only specified portions of the data class. ELDM data classes may be split into sub classes to allow for the more efficient transfer of information when users of the data do not require all data elements in the data class. Sub data classes will be defined in the VA ELDM where different components are the natural owners and stewards for data elements in a single ELDM class. Data elements that are exchanged between applications shall be as defined in the ELDM.

- **There will be a single authoritative instance of all data** – A single instance of each data class will be designated as “Authoritative” and will serve as a unique and unambiguous source of data to be shared operationally across all OLTP systems in the VA enterprise. All systems should access the authoritative copy of data wherever feasible. While copies of authoritative data are allowed for performance reasons, no changes made to such copies of authoritative data will be considered authoritative until such time as the authoritative instance has been updated.
 - Because they are derivatives of Online Transaction Processing (OLTP) data, neither Online Analytical Processing (OLAP) nor Operational Data Management System (ODMS) data shall be designated authoritative.
 - Because of its very low update frequency and high access rates, “reference” data may be replicated across the enterprise provided that there is a publish / subscribe mechanism to publish changes across the enterprise.
- **There will be a separation of business logic and data logic** – All data shall be accessed through a Data Access Service (DAS). Applications will consist only of business logic and will be separated from data logic so that each can be developed by those best able to do so and each can be optimized independent of the other. The SOA Service Layer contains data services that make the data available to the business logic services and will communicate with a series of Data Layer services that support the physical access to the data. The SOA Layer data services have no visibility into the physical structure of the data and can contain no SQL allowing changes to be made to the physical data structure transparently to the services at this level.

OLTP, OLAP (data warehouse / data mart data) and ODMS data will be kept separate, and not intermingled. Neither OLAP nor ODMS data can ever be authoritative as both are derived from OLTP data. Each type of data store is designed to support data access by a single type of application. Data stores shall only be accessed by its designated type of application.

7.2. Data Services

Previous sections described the four levels of services that would need to be supported by the SOA.

- **Presentation Services** – Provides external systems and end users access to VA data, services, and applications as appropriate. Presentation services may be implemented as JSR168 portlets in a portal.
- **Functional Services** – Are the services published by applications that perform functional services for other applications and are the traditional services described by an SOA.
- **Infrastructure Services** – Are supporting services (e.g., messaging and user authentication) that are provided by the infrastructure to all applications. Some of these would usually be provided as part of the ESB and others as core infrastructure. Infrastructure services may either be provided by VA for use throughout the enterprise, or where there is no VA provided services by each business area or MI for use throughout the business area or MI. These must be lightweight services because of their high usage and because they provide the core ESB capabilities
- **Data Services** – Provide applications access to data and are used to support the core principle that applications business / functional logic must be separated from data logic. These services

must very “lightweight” as they are used for all data access. Use of a data service to mediate data access for applications cannot use the ESB due to performance requirements.

7.2.1. Accessing Data Using Data Services

Data services ensure that there is a single authoritative source for all data in the enterprise and that all services can access that data as if it were local, regardless of where the data is located and who owns the data. Data services hide all aspects of the physical storage of the data from the applications allowing physical databases to be combined, partitioned, rehosted or redesigned or changed in other ways without impact to the application.

Data services specify the mechanisms by which services are allowed to access data. Two services are defined. These are Data Access Services (DAS) and Data Interchange Services (DIS). DAS are the services through which a functional service accesses all data. DAS provides direct access to data in data stores owned by the application of which the service is a part. For data housed in data stores not owned by the application, DAS will request services from DIS. DIS formats the data request as a message, passes the message to the ESB services, receives that data from ESB and provides the requested remote data to DAS for delivery to the user.

Conceptually, applications exchanging data through DIS and DAS would appear as in Figure 10 above. A distinction is drawn between the exchange of data through DIS and DAS and the request for processing services as requested through the Service Adapter.

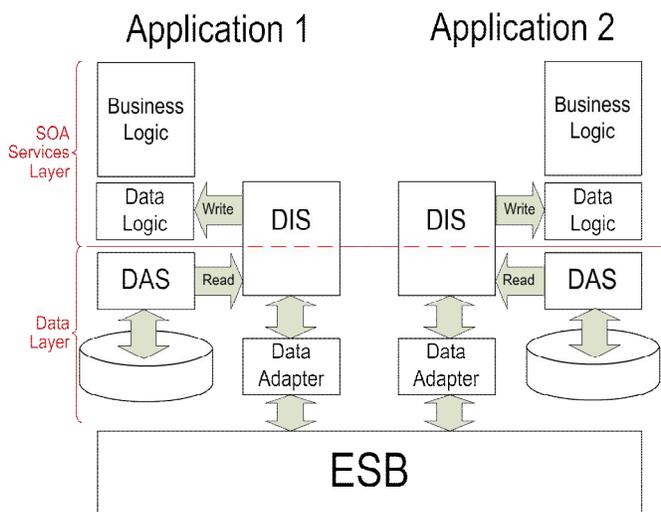


Figure 10 - Data Access and Data Interchange Services

- **Data Access Services** – Are used to isolate the physical database calls and data structures from the business logic and data logic in the SOA Services Layer to allow changes to the underlying data structures without changes to the applications. The DAS is used to populate objects with data and therefore are used to transfer quantities that are meaningful in a business sense. A single DAS request to read or write data may require access to multiple records in a database table or multiple tables in a database. One of the major reasons for the use of DAS is to allow the functional services to make a single request for data that may require an arbitrary number of accesses. The DAS also isolates services in the SOA Services Layer from changes to database layouts.

- **Data Interchange Services** – Are used by the DAS to access data for which the authoritative source is another application. This virtualization allows the business logic to be written as if all data were local. All DIS requests to update remote date are forced to pass through the remote service’s business logic to validate the data against the data owner’s business logic (e.g., perform data validation and editing) prior to updating the databases. The DIS will pass information through an ESB, which can perform the required logging. Note that DIS reads can access data through the DAS, but writes are routed through the business logic. The need

for all writes to pass through the business logic is to ensure that all appropriate business rules and data validations are performed prior to the database being updated.

To ensure application performance, these services will need to be very lightweight. The service call may require the ID of the user on whose behalf the request is being made so that requests can be authorized and logged, but no additional user authentication is provided. This logging can be done within the service so; it is not the responsibility of the application.

7.2.2. Exchange of Data Using Functional Services

Two major issues with legacy applications providing data through services interfaces to modernized applications are 1) a common understanding of the meaning of the data and 2) that all required data is available and can be transmitted. One common way of doing this is to implement a data transformation capability to ensure that all data transfers conform to the ELDM class and data element structures even though these are not native to the legacy applications, i.e., to syntactically and semantically harmonize the data. Because the ELDM data classes and data element definitions may be quite different than the data definitions used by the legacy application there may be a significant amount of work structuring the data from legacy data stores to the new data model. This would commonly occur when multiple applications have conflicting data definitions.

Translating legacy application interfaces and APIs into a standard form or into a form understandable by another application and ensuring a common understanding of the data to be transferred is difficult. This is basically the “heavy lifting” that is performed in Enterprise Application Integration (EAI). EAI hubs often do pair-wise data extractions, transformations, and reformatting or may transform all data to a standard form before transforming it to a form understood by the requesting application.

SOAs tend not only to need to do the data transformation to a set of common semantics, but they tend to specify the transfer information in a common syntax with a neutral format. This requires the transformation of the data from the data owning application’s native semantics format to the standard semantics and then to the requesting application’s native format and semantics. Because of the severe performance impacts of XML, many such transfers may use formats other than XML.

In order for legacy applications to participate in an SOA implementation a set of adapters must be developed, which allows the legacy applications to talk using the standard service protocols and data definitions. This will require some set of custom adapters, one for each such application. These custom adapters will need to be updated as the application changes over time.

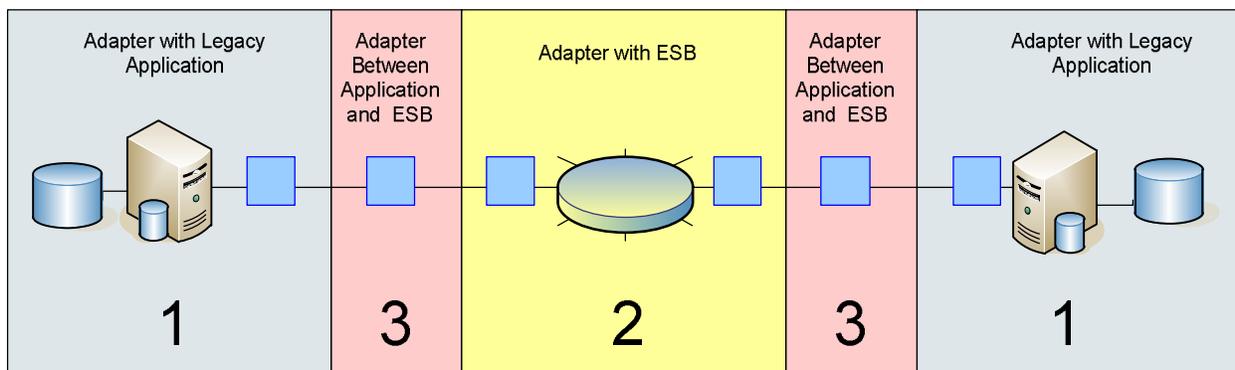


Figure 11 - Possible Locations for Data Transformation Adapters

Issues arise as to where in the architecture these adapters reside and who is responsible for them. The adapters can either be considered to be part of the applications (the region labeled “1” in Figure 11) and the responsibility of those applications to update. They can be part of the ESB or hub (the region labeled “2” in Figure 11) and their development and maintenance is the responsibility of the enterprise service infrastructure group, or, at least in theory, they could be in the area in between with responsibility being assigned on an ad hoc basis. In the first instance, it would be the responsibility of the application to build the adapters or interfaces needed to transform its data into the standard data definitions. This greatly simplifies the transformation work that must be done by the ESB and places responsibility for the most complex data transformation where the most knowledgeable people are – in the applications. The second approach makes virtually all of the complex transformation work the responsibility of the enterprise infrastructure group supporting the ESB and leads to the myriad of complexities and difficulties that have plagued EAI for years. Prior to the advent of services and SOAs, both EAI and MOM were touted as solutions to this enterprise integration problem, and neither was terribly successful at solving this problem. Burying this problem in the ESB is not likely to solve the problem either. Unless and until applications take the responsibility for distributing standard data elements, with standard definitions, in a standard protocol, these efforts will continue.

The SOA solution will be for the application owning the data to perform the syntactic and semantic transformations – transforming the data to the standard data definitions until such time as the applications modernize and natively store data in the standard format.

7.2.3. Use of XML in Data Services

One major advantage of XML is its extensibility and self-describing nature – which allows additional data fields to be added to existing messages while allowing existing consumers of those messages that do not need the additional data to ignore the additional fields. However, XML can be quite expensive to use both in terms of increased message sizes and the processing required to convert data to and from text strings. The problem with large numbers of smaller messages is exacerbated by the likelihood that identifying information would need to be sent with each message, further increasing the amount of work to be done. These factors would push towards a relatively coarse-grained definition for functional services to limit the number of calls and the amount of overhead work to be performed.

While XML may be used as the basis for functional services, it should not be used as the basis for enterprise “technical” services. Enterprise services tend to be much more finely grained than functional services and are called much more frequently, making the issue of very high XML processing overhead much more serious. XML should not be used for enterprise services and should not be used for service calls within an application. Where feasible, XML can be used, but for those functional services with high volume, high performance, or near real-time requirements, more efficient XML-like constructs should be used.

7.3. Data Sharing

VA data will be shared between many types of entities in a number of ways. In the diagrams that follow, security describes the number of security mechanisms available to protect data, authenticate users, or to authorize access. Level of trust describes how much trust there is between entities sharing the data.

There are of course privacy concerns related to the sharing of any data that may potentially be used to identify an individual or is PII or PHI. The privacy concerns include the exposure of data, the

aggregation of data, and the use of data for purposes other than those for which it was collected and unless all privacy compliance concerns are satisfied and requirements are met, any data sharing or data transfer described in this section is prohibited.

7.3.1. Types of Access Provided

Services will need to share information with other services within their application, with services in other applications within a business area or MI, with services throughout VA, and with services and systems external to VA. A service can access information in a number of ways. Some require a high degree of trust and thus, can only be used between services that have a high degree of trust between them while others are much more general and can be used with virtually any service. This section describes the basic mechanisms by which services can share information. Later sections will discuss the circumstances in which each of the mechanisms can be used.

Mechanism	Description
Federated Data Stores	Data Federation allows multiple systems to access a single DB giving multiple systems direct access to the data without requiring additional physical copies to be made or transmitted. Federated data stores allow multiple applications direct read or read / write SQL access to a common database. This is a highly efficient form of data sharing but can only be used for co-located systems and require intimate knowledge of the DBMS, DB design and schemas etc. All applications using the federated data store must be on systems with direct physical access to the federated data base.
Data Base Access through ODBC or JDBC	Open Data Base Connectivity (ODBC) or Java Data Base Connectivity (JDBC) provides a minimal layer of abstraction over the federated data model in that there is a minimal virtualization layer. This shields the user from requiring knowledge about the specific DBMS, but would still require the user to have direct knowledge of the data base schemas and makes the application sensitive to changes in those schemas.
Application Program Interface (API) Based	API-based data sharing is perhaps the most common between closely related applications today. Using this approach the application that owns the data provides a programming language-based API to the requesting application to allow the requesting application to request the data on-demand. The APIs are normally custom developed to meet the specific needs of the requesting application, require the development of point-to-point communications, and create dependencies between applications. It does, however, relieve the requesting application from requiring detailed knowledge of the DB schemas used.
Message Service - Native Application Format	The Native Application Format is the current format that the legacy applications use for communicating with other applications. These formats tend to be custom and unique to the specific pair of applications. In most cases these interfaces are over a standard COTS based or locally developed message queuing software. Queue-based message services rely on a COTS-based messaging system to provide all of the queuing and message delivery capabilities. Using such an approach, application requests for data are passed as a message or series of messages based on message format standards agreed to by the sending and receiving application. The characteristics of the queue-based messaging services are similar to the data

Mechanism	Description
	access / data interchange services discussed above in terms of being able to require end user identity as part of the request and the ability to implement logging as part of the service. Although the queue-based messaging approach requires more overhead than the data access / data interchange service discussed above, features such as additional security, guaranteed delivery, and publish / subscribe capabilities make it more suitable for data sharing between entities with a lower degree of trust.
Extract, Transform, and Load (ETL)	ETL access of data stores is used to extract data from potentially multiple data stores with overlapping information, transforming the data into some standard set of data items and loading the data into some more permanent data store to support OLAP processing. Because of the large data volumes the ETL process must be highly efficient, but because of the complexities of the data the transform process may be quite complex.
Data Interchange Service (DIS)	DIS comprise a set of standard data-interchange routines that give applications create, read, update and delete access to data owned by remote applications. DIS is part of the Enterprise Services Layer that is comprised of a set of common capabilities that are deployed across the Enterprise. These services are part of the shared IT infrastructure components that mediate relationships between applications throughout the Enterprise and the External Community.
Data Access Service (DAS)	DAS comprise a set of standard data-access routines that give applications the ability create, read, update and delete access to persistent, online data stores they own. DAS provides data abstraction to application developers and isolates them from the underlying details of database structures and the data manipulation languages. Data management and data optimization activities are contained within the DAS layer, allowing database specialists to independently construct and maintain these services without impact to the applications that use them. These services also include wrapper services such as security and privacy control, logging of data access requests, and error checking.
VA Standard Messages	VA Standard Messages are syntactically and semantically harmonized and have consistent metadata so that to data elements have the same meanings wherever in VA that they are used and that any two data elements with different names have different syntax and / or semantics. VA Standard Messages can be formatted as either XML, binary or other formats, but the syntax and semantics must be fully harmonized. Harmonization is achieved through the ELDM which identifies of the data elements that are of enterprise interest and provides for their unique definition.
Industry Standard Messages	Industry standard messages are messages that are transmitted based on some externally specified data protocol. HL7 or Electronic Data Interchange For Administration, Commerce and Transport (EDIFACT) are examples of externally defined, industry defined message formats. The Industry Standard Message protocol will not be semantically or syntactically harmonized with VA standard messages. There will need to be a conversion between the Industry Standard and

Mechanism	Description
	VA Standard messages.
Full Web services	Full Web services are the most general form for requesting services from an unknown party. Web services were specifically designed to allow commerce between unknown parties and were designed, in part, to protect each party from the other. However, much of the promise of Web services is unfulfilled due to the fact that standards in many areas, such as security, are not yet finalized and thus not generally implemented. Furthermore, the overhead associated with the multiple levels of the Web services protocol stack put very severe limits on the performance obtainable through Web services, thus, strictly limiting the transaction volume that can be sustained through Web services. Thus, Web services are most useful for entities with whom one has the least trust and with whom only limited number of transactions must be passed.
Custom Strategies	Custom Strategies could in theory include any type of messaging not included in the categories described above. However, they are commonly based on the point-to-point transfer of data through the use of data formats and interfaces that have been defined on a case-by-case basis between pairs of applications based solely on their requirements. This would include native format or XML messages and may use a standard message transport. Over time, other applications may receive a copy of the data feed transforming the custom strategy into a quasi-standard that will need to be supported over long periods of time.

Table 1. Data Transfer Mechanism

7.3.2. Characteristics of Organizations with which VA Components Share Data

VA systems transfer data with a large number of different types of organizations and entities. Some relationships are very tightly coupled (e.g., between application subsystems) while others are much looser (e.g., external entities). In general, data sharing relationships may be characterized by a number of attributes, each of which may impact the technologies and mechanisms that are used for data transmission. These attributes include:

- **Level of Trust** – The degree which there is a trust relationship between the data requestor and the data provider which may limit the level of access that the data requestor is provided to the data provider’s data.
- **Data Volatility** – Describes the amount of data that must be transmitted. Data volatility is based on the number of transactions per unit time and the size of each transaction. Data volatility will determine the feasibility of keeping multiple copies of databases synchronized.
- **Physical Location** – Limits some of the types of access that may be provided and the number of firewalls through which the data must pass.
- **Data Request** – The type of request that is being made. Typical types of requests can include:
 - Data Query.
 - Data Update.

- DB Synchronization.
- Data Download.

VA systems have a requirement to provide subsets of data to external parties. This data is extracted directly from the data warehouses and formatted based on the extract requirements. Based on business requirements for cargo, passenger or border crossing data to be included in these extracts, data may be extracted from OLTP sources.

Organization	Data Sharing Relationship
Application Subsystems	Application subsystems are the most closely linked entities and are the entities within which there is the highest level of trust – usually complete trust. The subsystems are all segments of a single integrated application and are likely to have common access to the application’s data stores. There is not usually any security mechanisms applied to information flows between application subsystems. Because of their high level of interactions there tends to be the highest levels of transactions flows between entities at this level and the greatest performance requirements. However, even at this level of trust, direct database access is not recommended; rather, the recommendation is that Data Access Services be used.
Application Groups	Application systems are not monolithic, but have evolved into a series of more or less loosely coupled applications. In some instances these associated applications are considered to be independent applications and in some instances they are considered to be subsystems. Sometimes a specific system is considered to be an independent system and at other times a subsystem of a larger application. For example, some system may be considered to be an independent system but are within the C&A and FISMA identification of the larger system. There is usually a very high degree of interoperability between applications in a group and considerable information interchange. Thus, performance is almost as much a criterion as for communications between application subsystems. The applications are more loosely coupled than actual application subsystems and it should be possible to include some additional security mechanisms, but performance and ease of use requirements (e.g., single sign-on) limit the additional security mechanisms that are typically employed. Because all applications in a group are typically owned by a single business area or MI and are usually hosted in the same data center there is still a very high level of trust. Queue-based message services should be employed if performance requirements are not so severe as to preclude their use. If queue-based message services cannot be used for performance reasons then the data access service or data interchange services can reasonably be employed.
Business Area / MI	Applications within a single business area or MI are much more closely related than systems in other parts of VA, but not as closely related to systems in other parts of VA. Because they are being developed by members of the same organization for many of the same customers. There will be a high level of trust between the systems in a business area or MI and likely a relatively high degree of communications between them. Because

Organization	Data Sharing Relationship
	they are not part of the same larger system, communication between systems in a single business area or MI, communications between these systems should be based on a formal mechanism.
VA	There is a requirement for communication between applications owned by different business areas or MIs. While there is still a high level of trust between applications at this level, they may present very different risk exposures because the applications may be hosted in different data centers and because of differences in their user communities. For example VBA systems usually only process PII information, while VHA applications process both PII and PHI information. Therefore, in general, there cannot be the level of trust that exists within an application group. Further, the amount of traffic between applications should be substantially lower than traffic within an application group.
DoD Military Health Services	While there is a strong difference in the level of trust and volume of communications between pairs of systems that are internal to VA and between pairs of systems that are external to VA. There is one exception to this, and that is MHS. MHS and VA have a special relationship and are in the process of developing joint systems that will share data. Therefore, from a view point of system trust and communications volumes the MHS systems would more closely resemble VHA systems than other systems external to VA. While there will be a close relationship between the MHS and VHA systems, they will typically be run on separate networks in separate data centers, so therefore there will be a limited to how closely connected the systems may be.
US Government Agencies	US Government agencies cover a wide range of agencies (e.g., Social Security Administration or Centers, the US Treasury, or the Centers for Disease Control), and as a result, the data to be shared will be at varying levels of sensitivity. Each such interface will be governed by one or more ISAs and MOUs, which will specify the level of sensitivity of the data being shared. The goal would be to replace the myriad of special interfaces that exist with a standard set of the services and message types. However, the speed with which this can be accomplished will depend on the speed with which the partner agencies can migrate their systems. Web services will be the preferred approach to satisfying the information requirements of the other US Government agencies.
External Entities	It will be necessary for some systems to communicate and provide services to external entities. While systems may request (require) external users to register, it can never be absolutely sure of the identity of the party requesting data. Therefore, there may only be minimal trust between applications and external entities. Communications with these external entities must be tightly controlled and should use Web services to provide the level of protection that is possible.

Table 2. Data Sharing Relationships

7.3.3. Data Transfer

Each data transfer can be described in terms of the three attributes that were described above:

- Data type (e.g., OLTP, Data Warehouse).
- Data transfer mechanism (e.g., Web service, MQ message).
- Organization to which the data is being transferred. Conceptually all possible combinations of these characteristics are described as a cube with each cell of the cube representing one specific data reference.

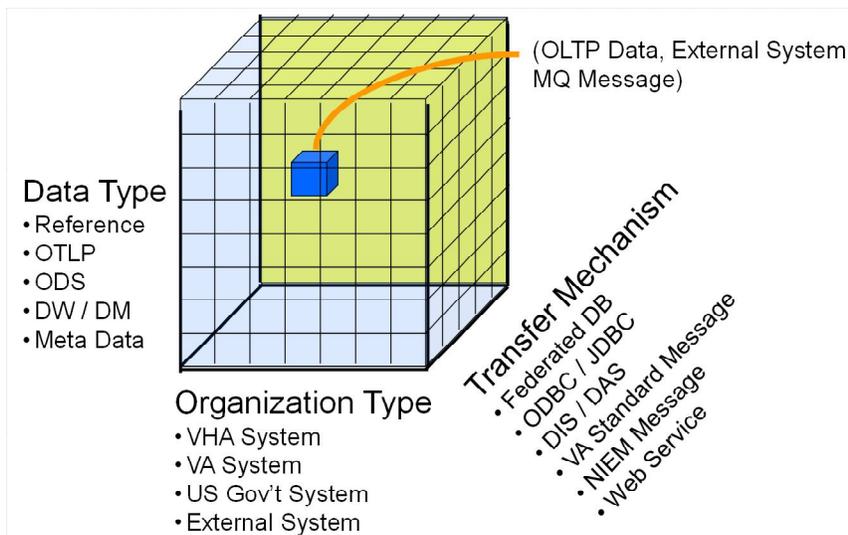


Figure 12 - Data Reference Characteristics

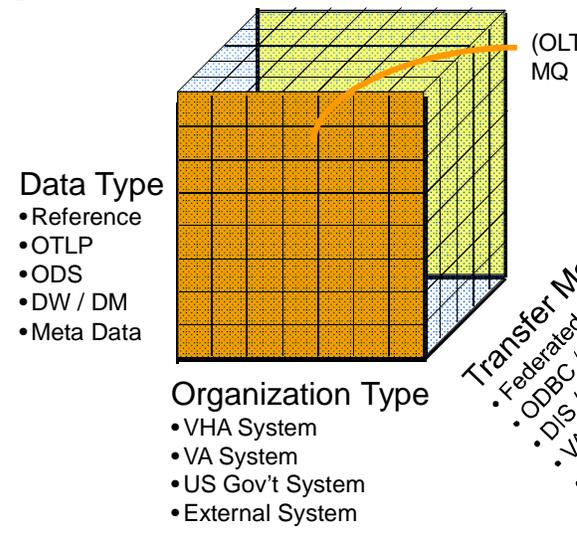


Figure 12 illustrates the three attributes and how they combine to define all possible data reference types.

reference types.

Each cell may also be described in terms of any number of attributes including:

- Level of Trust between the data requester and data provider.
- Data Volatility.
- Data Request Type.

These characteristics will then combine to provide a value for a cell, which specifies the values of the chosen attribute as it applies to that particular data transfer (transfer of that type of data, to that type of organization, using that transfer mechanism). Since the value for that cell is based on the attributes of the cell, and a single cell may represent a large set of accesses (e.g., transfers to all US Government Systems are represented in one cell) there can be some level of variability allowed. To illustrate the types of transfers that are recommended, several slices of the cube are provided below. The slices that are provided as illustrations are:

- Data Transfer Mechanisms vs. Organization Type.
- Data Transfer Mechanisms vs. Data Type.

Figure 13 identifies the recommended and allowed mechanisms for transferring data between a module in an application and systems in other organizations. The color codes used in Figure 13 are explained in Table 3. The values in Figure 13 are to be construed as minimums. Business Areas and MIs may restrict access further than indicated in Figure 13, but may not lessen the level of restriction in that figure. For example, where Figure 13 may identify an access mechanism as being a “secondary recommendation,” a Business Area or MIs may list the access mechanism as “allowed with waiver.”

Value	Description
Red – Not Technically Feasible	This approach is not allowed. The primary reason for an approach to be absolutely prohibited for a specific use is that the approach is not technically feasible for that intended usage. Therefore, there is no waiver process provided in this instance. Over time, some technically infeasible approaches may become feasible and migrate from red to orange, then yellow, and then green.
Orange – Not Allowed except under most extreme circumstances	<p>While technically feasible, this approach has been prohibited. It may have been prohibited for any of a number of reasons including:</p> <p>Security – The approach does not provide a sufficient level of security for the low level of trust between the organizations.</p> <p>Performance – The approach would incur a level of overhead that could not be sustained for the volume of transactions that would be expected.</p> <p>It is recognized that there may be very specific and highly unusual instances in which a prohibited technology may need to be used. Requiring a waiver for the deviation from the standard ensures that VA management has the opportunity to review any non-standard usage to determine that the non-standard approach is required and to negotiate limitations on the use of that non-standard approach.</p>
Yellow – Allowed, but Not Recommended	<p>This approach is not recommended, but is allowed. This approach is not considered to be best practice for this type of data transfer. Reasons that an approach may not be recommended include, but are not limited to:</p> <ul style="list-style-type: none"> • Performance – The overhead associated with the approach would create an unwarranted performance penalty for high volume applications. • Dependence – The strong desire is to promote the highest degree of independence, subject to performance considerations, between parties sharing data so that changes in the data or process structures do not impact the partners with whom the data is being shared. <p>The Yellow – Allowed but Not Recommended category was included to recognize the differences in levels of trust, transaction volume, and performance requirements of some of the systems and organizations within a category. Thus, a broader range of approaches was allowed. Further, it was included to allow solutions that are required because of laws, international agreements, specific standards or longstanding practices.</p>
Greenish Yellow – Secondary	This approach is not the primary recommendation, but may be recommended under circumstances where the primary recommendation would not be

Value	Description
Recommendation	appropriate. In most cases the secondary recommendation was included due to the variability of the systems and organizations within a category. Further, the secondary recommendations are designed to deal with the practical aspect that laws or longstanding practices mandate some interfaces and interface types.
Green – Recommended	This approach is the recommended approach for sharing data with organizations of the specified type. It provides the best balance of security, performance, maturity etc. and should be used in most cases. Some columns have multiple recommended approaches because of the variety of systems and organizations in the given category.

Table 3. Data Reference Recommendations

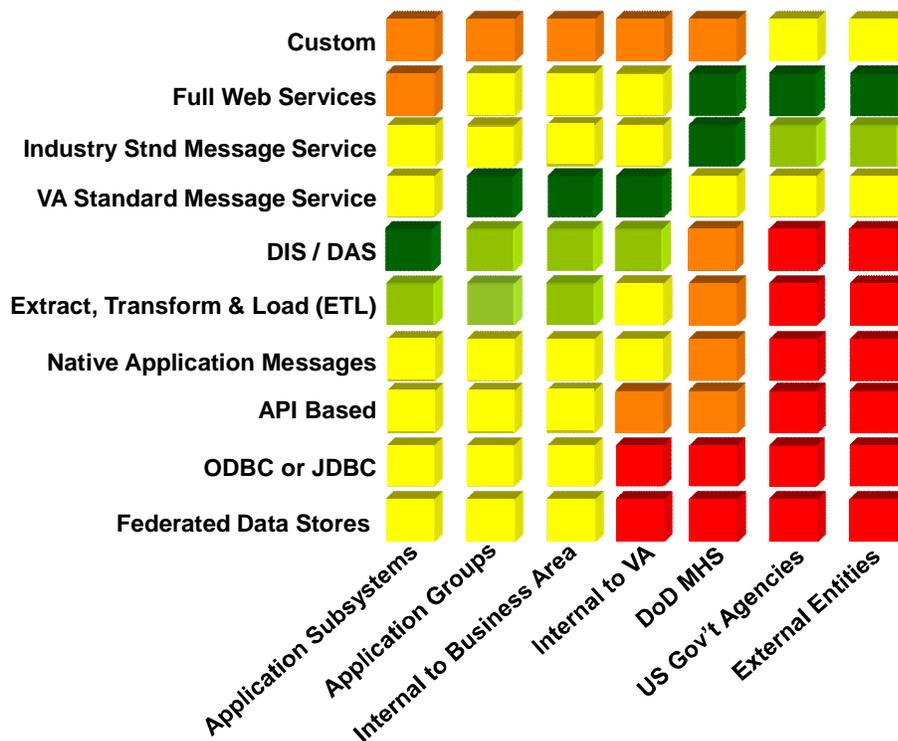


Figure 13 - Recommended and Allowed Data Transfer Mechanisms vs. Organizations to which the Data is Transferred

7.3.4. Detailed Data Transfer Recommendations

In addition to the system-to-system data transfer approaches, there is also a recommended approach for human interaction with systems. Human interactions should be via a browser-based portal that exists as a presentation service in the presentation layer or through specialized apps for PEDs. The portal should provide a tailorable, custom interface designed for users from each type of organization – i.e., users within VA would see different information than external users. The portal experience should be such that there is uniformity across the applications in terms of appearance, navigation, information presentation, etc.

8. SOA Security

This section describes the security aspects of the SOA. VA has a security architecture in place, has implemented a number of security mechanisms to instantiate that architecture, and has promulgated a number of policies enforcing use of those mechanisms. Nothing put forth in this security section is intended to in any way violate, or even relax, any of the currently implemented security architectures, policies, guidelines or mechanisms. This section only describes how security is to be applied to services and some special aspects of security relevant to the SOA and the mechanisms needed to secure the ESB.

This section is included because the implementation of an SOA creates new vulnerabilities and opens new avenues for attack that are not present in the development of monolithic, or even distributed, applications. It is the intention of this section to discuss architectures, mechanisms, and approaches to closing the holes and lessening the vulnerabilities that the implementation of services under this SOA will create. The security approaches described in this section are intended to be implemented over and above any security approaches and mechanisms that are currently implemented or mandated.

SOAs are based on the definition of a set of common services that are used by all applications and which typically do not differentiate between calling applications. Since services are called and processing is done outside of the calling application, it is quite possible that the service will be provided from a process in a domain other than the one in which the service consumer resides. Because of the nature of services and SOAs, an application may be unaware of the security domain from which a service it consumes is provided, or even if, at some point, the service is moved from one security domain to another. Any flow of information between security domains introduces potential vulnerabilities. While multiple mechanisms and protections can be included on top of the services, this introduces a layer of cost, processing and inefficiency.

Because of the privacy sensitivity of the information processed by VA systems, it will be critical for the SOA to be developed in a manner consistent with established VA Security Architecture so as to ensure that vulnerabilities are not introduced by the introduction of the SOA and to keep additional complexity and processing to a minimum. A key to ensuring the required level of security of the VA and enterprises will be the very careful integration of the SOA with the enterprise security architectures.

Authentication of user identity and control of access to services will be critical to the success of this SOA. This SOA is based on VA employing the “defense in depth”¹⁹ security concept. In this approach multiple levels of defense and multiple mechanisms are used to ensure that even if one security mechanism is breached, the security of the overall system remains intact. *Defense-in-depth* is one of the guiding principles of this SOA.²⁰ Multiple layers of protection will be employed against

¹⁹ Defense in depth is a security concept in which there are multiple security mechanisms and multiple layers or tiers at which security mechanism are applied so that were one mechanism compromised or one layer or tier breached that the enterprise data and resources would still be protected and secure.

²⁰ See NIST SP 800-27, Engineering Principles for Information Technology Security (A Baseline for Achieving Security). It briefly defines defense-in-depth, provides further references, and outlines engineering principles such as layered security. URL: <http://csrc.nist.gov/publications/nistpubs/800-27/sp800-27.pdf>

threats to people, processes, procedures, facilities, systems, data and technology, beginning with adherence to all applicable Federal and Department security policies and procedures.

Defense in depth mandates that authentication and authorization be distinct operations. Authentication ensures that the identity and role of a person is that which is claimed. Authorization determines what a person with a known identity and role is allowed to do. For example, when logging onto Veterans Health Information Systems and Technology Architecture (VistA), a person might be authenticated with an identity of "John Doe" and role of "Pharmacist." This authentication in and of itself would not allow access to any VistA data. VistA would then need to use its own authorization mechanism to determine whether "John Doe" as a "Pharmacist" would be authorized to see any specific data. Authentication can, and in the future, is expected to be performed on a VA-wide basis by a single service – a VA Enterprise Identity Management Service. Authorization will need to be performed on a system-by-system basis and possibly even within specific subsystems or related to specific data classes or data records.

Since the SOA will be an integral part of the VA enterprise, the security mechanisms used to support the SOA will need to be integrated with the security mechanisms for the rest of the enterprise. Because the SOA provides new access paths to systems and data it follows that these paths must be protected to the level that the rest of the enterprise is protected, lest the SOA provide new less protected paths to current systems. One side effect of the use of an SOA is to remove applications boundaries and to allow access to code that once resided inside of an application and was thus, inaccessible. That this code can now be exposed as a service and made accessible introduces vulnerabilities that require protection. The SOA also provides increased mechanisms for external access (e.g., internet access) to VA functionality. This too introduces vulnerabilities that must be defended against.

New applications will be built based on a set of services built in accordance with the SOA. Service-based interfaces will be built for legacy systems to allow service requests to be used as the primary interface to those systems. This will inextricably tie the security and access control approaches used to support the SOA with those that support the legacy and modernized systems.

While services, as used in an SOA, do not imply "Web services," there will certainly be a requirement to expose a series of services as of Web services. These Web services will be built using a number of Web services standards, including a number of Web services standards specifically dealing with security, identification, authentication and other related topics. They are designed to provide a level of trust between Web services consumers and Web services providers. However, not all of these standards are complete or approved, much less implemented. Further, these standards are limited in the fact that they pertain only to the interaction between the service consumer and the service provider and do not deal with the vulnerabilities of the design of the service being exposed or the vulnerabilities of the environment in which they are being implemented.

The WS-Security specification²¹ and the others will allow the attachment of certificates to messages, the encryption of messages and some other capabilities.

One issue with the use of Web services approaches to security is the number of service requests that may be made, particularly where the SOA is being used to support the integration of internal systems. If a process orchestration contains five service requests, and each of those five services calls five lower level services and each of those decomposes into five more services – there will be 125 service calls for each of instance of the orchestration. Were there to be a requirement to perform this business process 100 times per second, the number of service calls becomes very large. Were each service call to require authentication, authentication would be required 12,500 times per second. Since each Web services authentication requires multiple messages and data accesses, performing this number of authentications would be infeasible when services are used to implement relatively fine-grained services. Therefore, a means is required that provides strong authentication, but then does not require authentication on each subsequent service request. If the subsequent services are not required to perform their own authentication, they must be ensured that they can only be reached by service requests that have been reliably authenticated.

Further, while adherence to the Web services standards can ensure that the consumer of a service is who “it” claims to be and provides security for the interaction between service consumers and providers, there are many more design considerations related to security that they do not address. While the WS-Security family of the standards will necessarily form the basis for any Web services that are developed, they do not address any of the traditional issues related to the design and development of applications that are secure, trustworthy, and properly handle design and implementation issues such as authorization and role-based security. Each of the design considerations that have always been required of VA applications, including logging, authentication,

²¹ WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

WS-Security also provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. It is designed to be extensible (e.g. support multiple security token formats). For example, a client might provide proof of identity and proof that they have a particular business certification.

Additionally, WS-Security describes how to encode binary security tokens. Specifically, the specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. It also includes extensibility mechanisms that can be used to further describe the characteristics of the credentials that are included with a message.

By using the SOAP extensibility model, SOAP-based specifications are designed to be composed with each other to provide a rich messaging environment. By itself, WS-Security does not ensure security nor does it provide a complete security solution. WS-Security is a building block that is used in conjunction with other Web service and application-specific protocols to accommodate a wide variety of security models and encryption technologies. Implementing WS-Security does not mean that an application cannot be attacked or that the security cannot be compromised. (Specification: Web Services Security (WS-Security) Version 1.0 05 April 2002).

access control, authorization, etc. will still need to be included in the design of any services under this SOA – even though they may adhere to each of the WS-Security standards.

There are two basic approaches to access control, which are often combined to provide an even more secure environment. These are “security domains” or “domains of trust” and “credential” or “token” based approaches. In the security domain approach there is a high level of authentication and security applied prior to entering the domain. Once in the domain, there can be a high level of trust between systems and relatively little need for further authentication – but there is a continuing need for authorization. In the token or credential approach, users are authenticated either based on standard USERIDs and Passwords, tokens, or credentials. Systems are authenticated, one to another, using certificates²².

Both approaches can be used in conjunction with one another to provide even stronger security. Standards such as WS-Security can be used to authenticate external users to VA systems, but once authenticated, VA systems and services can use internal authorization mechanisms. The defense in depth strategy includes the use of both of these approaches.

8.1. Domain Model

NIST Special Publication (SP) 800-33 defines a security domain as “a set of active entities (persons, processes, or devices), their data objects, and a common security policy.” More concretely, a security domain is a set of networked hosts protected by one or more security perimeter devices from unconstrained access by other network components. Security domains, with their accompanying perimeters, provide the basis for placement, and configuration of firewalls, VPNs, and remote access protection devices.

A security domain may consist of a single host, one or more LANs at a site, or a group of networks connected via a network provider or backbone. It is distinguished by the fact that security devices at one or more access chokepoints mediate access to hosts or network components within the domain from any host or network component outside the domain. The security perimeter is the set of all access points, such as routers or modems, through which the hosts and network components within the domain are connected to networks or other components outside. Forcing all accesses into the domain from entities outside, and vice versa, to pass through security devices at the access points protects the domain and enforces a policy at the security perimeter. Typically, there will also be physical protection at a security perimeter.

²² FIPS PUB 201 “Personal Identity Verification (PIV) of Federal Employees and Contractors” specifies the architecture and technical requirements for a common identification standard for Federal employees and contractors. The overall goal is to achieve appropriate security assurance for multiple applications by efficiently verifying the claimed identity of individuals seeking physical access to Federally controlled government facilities and electronic access to government information systems. The standard contains two major sections. Part one describes the minimum requirements for a Federal personal identity verification system that meets the control and security objectives of Homeland Security Presidential Directive 12, including personal identity proofing, registration, and issuance. Part two provides detailed specifications that will support technical interoperability among PIV systems of Federal departments and agencies. It describes the card elements, system interfaces, and security controls required to securely store, process, and retrieve identity credentials from the card.

Since authentication is performed each time that a service request passes through a gateway, there is some overhead associated with passing through a gateway. Because of this, the number of levels in the hierarchy should be kept to the minimum needed and that business process orchestrations stay within a single domain to the extent feasible. Any authorization to access a specific service or specific data would need to be done by the services within a domain since authentication at the gateway does not provide any authorization to see data or perform processing within the fortress.

There is a distinction that should be drawn between where a service resides and the level at which it is published – i.e., made available to users. A service may actually reside at a low level of the hierarchy because of a need to access rather sensitive data, but be published at a higher level. Access to this low level service can be provided through a “proxy service” that is published at the higher level. This proxy would exist and be accessible at the higher level and would call the lower level service.

The domain model assumes a high degree of trust and sharing amongst those within the domain and very tightly controlled access to those outside the domain. This model can be implemented with very little credential checking within the domain because of the level credential checking and authentication that occurred on entry to the domain and because the only way to get to the resources within the domain is through the heavily guarded gate to the domain. Note that the credential checking at the gateway provides authentication – verifies that the user is who he or she says that they are, it does not provide authorization to do anything in the fortress. Services in the domain will still implement role-based security as they require. The credential checking at the gateway ensures the identity of the service requestor and verifies their role, so that the services inside of the domain need only concern themselves with the access that the specified role is allowed and not need to be concerned with whether the user deserves that role.

However, just because credential checking is not required within the domain does not mean that such checking cannot be used to further increase security within the enterprise. This model does not specify the security model used to authenticate the user at the incoming drawbridge or gateway. The authentication could be based on USERID and PASSWORD or other stronger approaches such as PKI. In the short term applications may continue to use USERID and password, but once certificates are issued, PKI techniques should be used. The domain or trusted computing base approach also works well with a “role-based security” approach because the role is assigned on entry to the fortress and that role can be used to authorize access at each process within the fortress.

In essence the domain or fortress model of security – if applied at a granular level – only provides the level of security inherent in applications today. It is only possible to access applications through the “front door” (e.g., by logging into the application or entering through an approved access point) and no concept of allowing external users to access Dynamic Link Libraries (DLLs) or modules within an application. The SOA, by exposing application functionality as services – whether they are intended for external access or only internal access, allows this level of access. Providing perimeter security around the newly developed services provides the security that the application boundaries provided previously. While Web services standards such as WS-Security and portals may help with services exposed externally, they are of exceedingly limited assistance for securing services intended to be exposed only internal to an application or an application domain or application group. The fortress model provides a hard boundary that corresponds to what would have been the application boundary.

Each domain will need to be on an isolated network segment – either a physically isolated segment or a logically isolated segment – to ensure that there can be no access to the fortress other than through

the gateway. There will be a requirement for network engineering to ensure that the services inside of the domain can only be accessed through the gateway.

8.2. Token or Credential Based Security

The token or credential-based model is based on the application being provided a token or credential (e.g., a PKI certificate) that can be used by applications to enforce predefined security policies possibly within the context of a role based security system. This has the further advantage of being able to provide a “single sign-on” mechanism that allows the user to sign-on once and be allowed role-based access to each of the applications for which he is authorized (i.e., his token or certificate is passed from system to system). The certificate or token is used to identify the user to the systems that he accesses and the system determines through its own mechanisms that level of access which the user is supplied. Even with certificate-based single sign-on, there will probably be cases where some services will require “re-authentication” prior to granting access.

One issue with token or credential-based security systems when applied to services in an SOA is time scales. In most cases the tokens or credentials that are supplied are only valid for a predetermined period of time. However, with the orchestration of services common in an SOA where a single orchestration may have a long duration because of the number of services that are consumed or the delays between services (e.g., where there is a human interaction) the duration of the orchestration may be longer than the time for which the token or credential is valid. This could result in the need to reauthenticate the user multiple times during the processing of the orchestration if the time limit for the validity of the token or credential is not set to a long enough interval. Even for short orchestrations the credential would need to be checked at each service call to ensure that the certificate had not expired.

8.2.1. VA PKI Implementation

VA Security and Identity Management are based on the use of PKI-based encryption.

8.2.1.1. Public Key Infrastructure

Public Key Infrastructure (PKI) uses public/private key pairs to enable enhanced services for Identification, Authentication, Confidentiality, and non-repudiation. PKI also provides capabilities for digital signing of documents. The use of two-factor identification (something you have – a certificate, in addition to something you know -- a password) improves security. The use of public/private key pairs allows data and communications to be encrypted by anyone, but decrypted only by the designated recipient. It also allows a user to digitally sign an electronic document with that user’s private key and then have that signature verifiable with the use of the public key. This signing capability provides the benefit of non-repudiation, making it impossible for someone who signed a transaction to deny having made the transaction.

Users of PKI must register, adhering to defined processes that validate their identity and permissions, before receiving a certificate. The certificate contains a variety of information about the issuer, the recipient, and the private key for that user.

8.2.1.2. PKI Trust Relationships

The general PKI architecture defines a set of trust relationships. A root trust authority, known as the Root Certificate Authority (Root CA), issues certificates. The Root CA typically only issues certificates to a small number of subsidiary Certificate Authorities (CAs), or Registration Authorities (RAs), which then issue certificates to end users. The process for establishing the root certificates is

documented in the Certificate Practices Statement and the Certificate Policies. These legal documents describe the trust relationships, the levels of trust, and the processes and procedures for granting and revoking certificates.

In operation, the PKI consists of one or more CA servers, one or more RA servers, and a Directory that stores the certificates, and makes their public keys available as required to applications. Depending on vendor implementation, CA servers and directories may be combined in various ways on one or more physical machines.

The Federal Bridge Certificate Authority²³ (FBCA) is the root of the certificate hierarchy for US Government agencies including the VA. The VA root certificate authority that has been signed and certified by the federal bridge at all levels of assurance (basic, medium, and high). The VA root makes use of two "policy" certificate authorities, one internal and one external, that will sign certificates for "issuing" certificate authorities so that they may issue federal bridge certified credentials while maintaining separate policies for those entities internal and external to the VA network and user base. The VA Authentication Federation Infrastructure (VAAFI) has two certificate providers to provide certificates to external users of its systems. As of the date of this document VA has two certificate providers:

- DoD Self-Service Access Center (DS Access) – a credential provider which issues the DS Logon credential.
- Operational Research Consultants (ORC) – a civilian company partnered with the VA.

Provided that the gateway to the presentation layer correctly implements X.509 technologies, the gateway can accept certificates from both internal and external sources. For example, in order to do certificate based authentication using SSL (TLS) the external client and the gateway would simply need to use certificates that were signed by the same authority somewhere in the certificate chain. This will allow the gateway to be able to accept a valid identity from the internal and external CA's as well as an identity from anyone signed by the Federal Bridge or signed by a certificate authority subordinate to the Federal Bridge.

²³ The Federal Bridge Certification Authority (FBCA) is an information system that facilitates acceptance of certifications for transactions. Since its initial conceptualization and operation, the FBCA has evolved into the Federal Public Key Infrastructure Architecture (FPKIA) that encompasses Certification Authorities (CAs) from multiple vendors supporting different FPKI policy and function. The FPKIA enabling policy CAs are the: (1) FBCA, (2) Federal PKI Common Policy Framework (FCPF) CA, and (3) Citizen and Commerce Class Common (C4) CA. The operation also incorporates the E-Governance Certificate Authorities used to issue Secure Sockets Layer/Transport Layer Security protocol certificates supporting assertion-based credentials for Security Assertion Markup Language (SAML) data exchanges. The Federal Bridge Certification Authority (FBCA) supports interoperability among Federal Agency PKI domains in a peer-to-peer fashion. The FBCA will issue a certificate only to those Agency CAs specified by the owning Agency (called "Principal CAs"). The FBCA, or a CA that interoperates with the FBCA, may also issue certificates to individuals who operate the FBCA. The FBCA certificates issued to Agency Principal CAs act as a conduit of trust. The FBCA does not add to and should not subtract from trust relationships existing between the transacting parties. The Federal PKI Policy Authority (FPKIPA) is the governing body over the FBCA that operates under the By-Laws and Operational Procedures/Practices for the FPKIPA (Version 1.2, September, 2011).

The defense in depth approach will combine the token or credential approach with the fortress or trusted computing base. The token or credential is used as the mechanism that allows access to the security domain through the gateway. The user is authenticated using the credential or token approach and is allowed entry into the domain. The PKI certificate, while authenticating that the user is who s/he claims to be, does not provide the user any privileges. The role(s) that the user will have and thus, the privileges that are provided must be determined upon entrance to the domain. At times a user may have a desire to change roles during a session. This may be accomplished by either starting a new session by logging in with the new role or by leaving the fortress and being reauthenticated and then being allowed to re-enter the fortress and to proceed.

8.2.2. Digital Signatures

The authenticity, accuracy, and nonrepudiation of digital messages can be assured by digitally signing the messages.

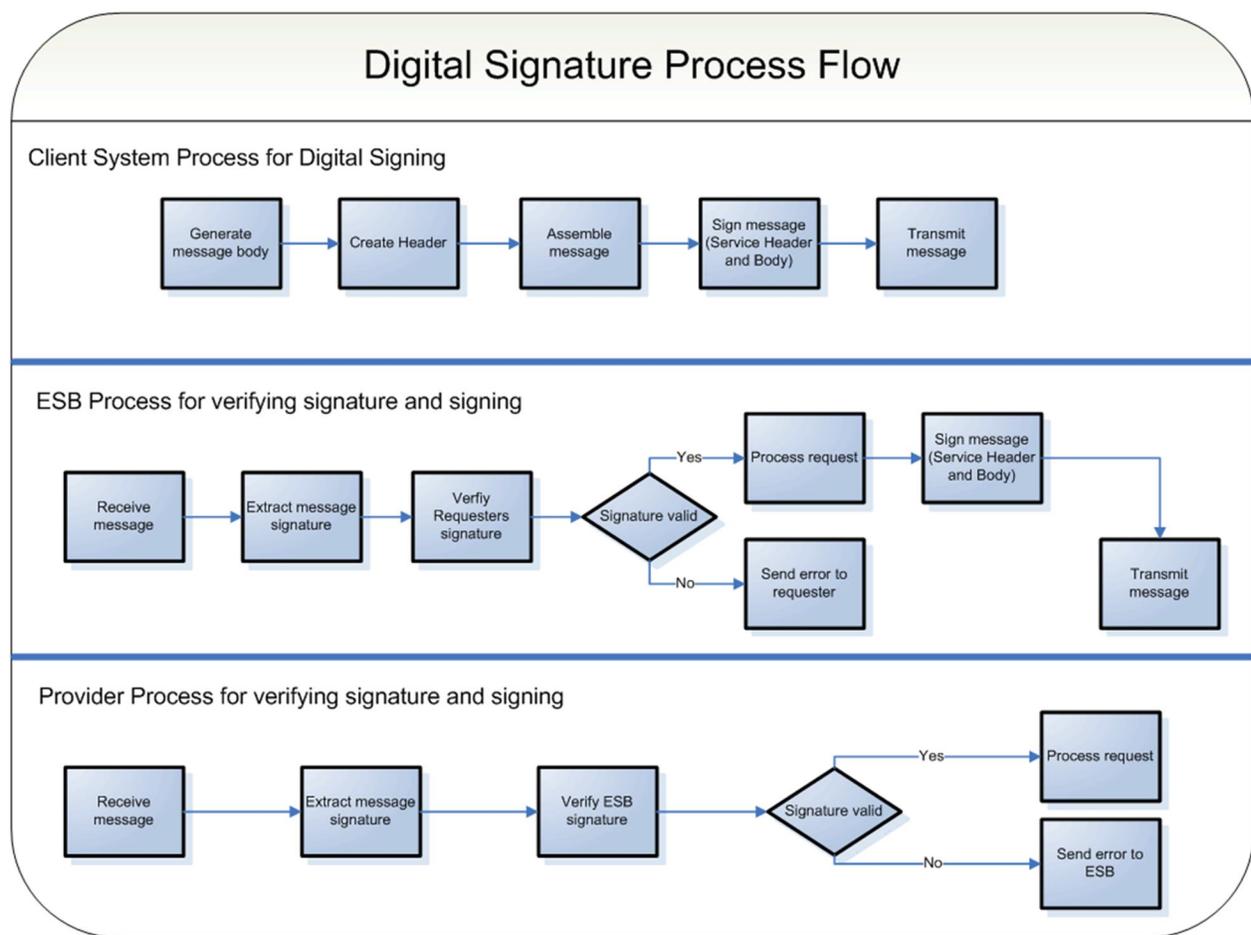


Figure 14 -- Digital Signature Process Flow

Digitally signing a message in a PKI system involves encrypting a hash of a message or a hash of a portion of the message with the sender's private key. Therefore, if the body of the message is transformed or changed the message needs to be signed by the private key of the entity that transformed the message. In the VA SOA the digital signature will be based on the encryption of a hash code that is based on the entire message. Thus, any transformation that changes the content of the message will change the hash code and thus, require the message to be resigned.

If the receiver of the message can decrypt the hash of the message with the purported sender's public key and the value of the hash calculated by the receiver matches the value of the hash calculated by the sender and contained in the message, the receiver can be assured that:

1. The message was sent by the purported sender
2. The message was not changed while in transit
3. The sender cannot claim to have not sent the message (nonrepudiation)

8.2.3. ESB Use of Digital Signatures

The ESB will rely on digital signatures based on certificates issued by a certificate authority subordinated to the Federal Bridge, or cross certified with the VA Root CA. All messages (including responses) sent through the gateway shall be signed by the sender using the sender's private key. Requests that are forwarded unchanged are not resigned--the original sender's signature is forwarded along with the message. When a message is forwarded without transformation and thus without being re-signed, the ESB and gateway services will verify that the sender's key has not been revoked, and will transmit the signature with the original message.

If the message is transformed as it passes through the ESB, it will have been transformed by a proxy service in the ESB, and will be signed by the ESB with the proxy service's private key. In this case, we consider the ESB service to be the sender. In the case that the ESB transforms the message, the ESB will verify the signature of the original message. The outgoing message will then be signed by the ESB transformation service on whose behalf the original message was transformed with that service's private key to verify both the message content and the identity of the original sender. In no case would the ESB sign a message with a user's private key (although it will be able to sign for any of the services that the ESB hosts). When the message is received by its intended recipient, that recipient will verify the signature of the message, and that the signer was either the stated sender or the specified service.

Certificates will be issued to each of the services. These certificates will be requested by the application developing the service. Certificates should be assigned to individual services. However, multiple services within a single security domain may share a certificate; but if they do so, they will each be treated as being identical (i.e., as being the same service) from an authentication and authorization standpoint.

Certificates may be issued to individual users or to VA systems making service requests on behalf of a user. In some cases the application system may be allowed greater access to data than the ultimate end user and it will be the responsibility of the application system to ensure that this additional, or more sensitive, information is not inappropriately exposed.

The PKI certificates issued to the user, or service, ensures that the requestor is who is claimed. However, many applications systems can provide differing amounts of information based on a number of factors other than the identity of the user. Additional information can be included in a service request further specifying end user identity, roles, or access rights and thus, may allow further fine grain authorization and allow access to a greater range or volume of data than would otherwise be provided.

8.3. Security Implementation for Services

While the domain model and the token or credential-based approach can ensure a service provider that a service requestor is who she says that she is, it will be up to the service provider to ensure that

the services are developed consistent with security requirements. Security labels in databases can describe the sensitivity of information that is being accessed and security policies can help ensure that sensitive information is not inadvertently exposed by a service.

For security purposes, it will be necessary to distinguish between services used solely within an application and those that are exposed to users or other systems. There should be no instances in which user-facing, external-facing, or application system-facing services expose data with more than a single level sensitivity. Separate user-facing services should be developed for each level of access that is required. These discrete “*Single Level Security*” (SLS) services may call a common service for actual data access and risk assessment. It is the responsibility of the user-facing, external-facing or application system-facing services to ensure that only a single level of information is exposed through the service interface.

Just as services must be developed that only support a single level of security registries and repositories must be developed to limit access to a single level of information. There may also be a role-based component that only allows specific role-based access to subsets of services.

8.3.1. Overview of the VA SOA Security Model

The VA SOA security model will be based on both the domain and the token based approaches described above. The two approaches are combined because of the nature of the SOA and services being exposed. The VA SOA will encourage the use of Web services for communications between VA applications and outside entities. For these types of services standard Web services approaches to security will be used including WS-Security and SAML. However, Web services were developed to allow relatively coarse grained services to be provided and consumed by organizations which are not known to each other and between which there is no inherent trust.

There are several areas in which the VA SOA differs from the Web services paradigm that impact the security model including:

- The VA SOA allows external services to be other than Web services and thus, security approaches are required that can handle other than Web services.
- The VA SOA encourages the use of fine grained services as the basis for development of new applications.
- The use of fine grained services as the basis for the development of applications will result in the large numbers of service requests. Since each SAML authentication request is very expensive in that it requires several messages and database lookups, it is not suitable for very large number of fine grained service calls.
- When applications are built from large numbers of services, service requests cannot be made to any service, but must be controlled so that they only enter the application at valid entry points.

The domain model is required as it allows users to be authenticated on entry to the domain and all services within the domain can rely on the authentication that was performed on entry to the domain without requiring authentication on every service call.

User authentication will be done using the capabilities of the VA ID management system, once that system is designed and implemented. The VA Identity Management solution will include role based, fine grain access to services based on a uniform set of role definitions across VA. It is assumed that

once such a set of roles is defined they will be instantiated in the VA Identity Management system which the SOA will use for authentication once 1) the VA Identity management solution is implemented, 2) VA-wide roles are defined, and 3) they are instantiated in the VA identity management solution.

Until that solution is in place an interim solution will need to be used in which users will log applications and be authenticated by those applications. The applications will request services on behalf of the user and digitally sign the service request with the application's digital certificate. This provides the VA ESB assurance that the request came from a trusted application. For atomic services the VA ESB will then process the service. For composite services in which the VA ESB issues service requests the services will be digitally signed by the VA ESB using its own digital certificate.

The VA SOA separates authentication from authentication. The SOA will perform authentication at the gateways. Authorization will be performed by the applications based on information provided as part of the service request. The service request may contain information beyond the identity of the requestor, but may include additional user credential information that would allow the application performing the service to provide more sensitive data than it otherwise would.

8.3.2. Security Capabilities

The capabilities in Table 4 will need to be provided to support the SOA. Some, such as authentication should be provided as a service, while others, such as integrity and non-repudiation cannot be implemented by a simple service and are capabilities that must be provided throughout the design. Some of the capabilities will need to be of concern to in the development of services in response to the SOA, while others will not be of direct concern to for the functional service developers.

Much of the access to security and privacy related information will be managed at the database level – only returning data which the requestor is allowed to receive, but this will not always be possible since there will be instances in which service will need to access more highly classified data to generate an answer that will ultimately be published a lower security level.

No	Capability	Definition
1	Identification	A unique identifier, such as a system account user ID, by which a system can recognize a user and associate the user with their system permissions. A system may generate a new USERID for the user when the system must make a service request on behalf of the user and the request must be performed at the system's level of access not the user's level of access. In this case, the system calling the service would need to filter data to ensure data is not inadvertently disclosed.
2	Authentication	Provides proof of identity for communicating principles.
3	Authorization	Defines system objects to which access is granted based on subject identity.
4	Identity management	See Section 5.4.4.1

No	Capability	Definition
5	Confidentiality	Protects data from unauthorized disclosure.
6	Integrity	Protects data from unauthorized modification.
7	Authenticity	Provides assurance of data origin.
8	Non-repudiation	Prevents communicating parties from denying having participated in the communication.
9	Key management	Comprises the technologies, tools, and procedures for management of cryptographic keys including issuance, distribution, query, revocation, expiration and renewal.
10	Access control	Restricts system access based on rules (service, IP address, time, content, etc.) rather than individual identity as with authorization.
11	Log	One or more records of security relevant events. The log itself is distinguished from how the record is handled.
12	Audit	Observation and analysis of log events to identify security anomalies.
13	Privacy	Providing a unique identification that specifies a privacy group to which the service call is subject. The identification may change over time to reflect that companies can buy and sell companies or divisions.

Table 4. Key Security Capabilities

8.4. Identity Management Services

All users of VA information systems and resources, both persons and systems, require electronic user identities (UIDs). These identities map real persons, or real machines, to created identities that are used for controlling access to component resources and data.

Identity Management unifies the management of information related to individuals and their interactions with application systems. It includes managing the user lifecycle (including provisioning) and automating all the business processes required for user management. components must manage UIDs not only for employees and contractors, but for third parties (e.g., external organizations with which VA components communicate), other government agencies, and foreign organizations as well.

Currently, many VA employees may have multiple UIDs. User Identity Management Services provide the means and mechanism for both managing the current proliferation of UIDs and for managing (creating, modifying, and deleting) future IDs, providing the user an integrated identity across the enterprise. VA has committed to the design, development, and implementation of a VA-wide Identity Management System. This identity management system will provide the basic identity management services for this SOA.

A central Identity Management Service application provides core services. Agents reside on target applications, network operating systems, and workstation operating systems providing authentication and authorization services to the various targets, from the primary identity management application. This component structure allows management and administration of user identities to be centralized, while still allowing distribution of the work effort through the lowest appropriate level, out to the end user.

The required Identity Management Service provides an integration layer for users' IDs, and also provides a management interface to that integration layer. The integration layer provides links between the ESB and the individual applications, systems or operating systems that require a unique UID that is specific to that system. Thus, a workstation UID and password can be linked to an email UID and password, associated with the Portal UID and password, and a mainframe application UID and Password. The Identity Management Service is both an identity service and an integration service. The integration service would provide the link between the central identity management application and other ID sources (email, network OS logon, applications, etc.).

8.5. Services Logging

Auditing is the after-the-fact examination of actions on the systems to assure that no one has performed actions that they should not have performed. Auditing requires logging of transaction and database queries so that there is a data store available to be examined. While performance requirements may preclude a detailed real time analysis of all of the transaction or database queries, sufficient information must be logged to allow later analysis.

Although the primary focus of auditing is the satisfaction of security requirements, the logging infrastructure may also be used to support the diagnosis of system failures and performance problems at little additional cost. It should also be noted that audit logging also supports the verification of privacy compliance.

There are a number of sources of requirements for "audit" logging of services and their actions including that "The information system captures sufficient information in audit records to establish what events occurred, the sources of the events, and the outcomes of the events."²⁴

8.5.1. Service Execution Logs

The logging described in this section refers to "audit" logging under the assumption that the log records must support legal action (i.e., investigations or forensics) and fault diagnosis, not data recovery. Data recovery would need to be performed by the DBMSs and / or applications involved. The logging context is that of the complete services request/reply or reply/forward patterns in a Service-Oriented Architecture (SOA) operational environment. Administrative tasks, such as account creation, granting/revoking access, logon/logoff, etc. would not be included. Although such events should be logged, they are not in the scope of SOA operational logging described in this section.

Further, in many applications in which data is separated from processing, it is the application, not the ultimate user that is logged as the user of the DBMS or second tier system. As part of the service

²⁴ NIST Special Publication 800-53, Recommended Security Controls for Federal Information Systems, February 2005

definition, it will be necessary for each service request to include identification of the ultimate user, the subject identity, for whom the work is being performed.²⁵ The following information should be captured as part of every log record:

- Date
- Time²⁶
- Subject identity
- Service name
- Event type
- Event outcome²⁷
- Payload

In addition to the information that is required the following information, based on the ACE logging techniques, should be included:

- **Requesting system** – physical identifier of source and destination, typically an IP address or mainframe terminal identifier
- **Session GUID**²⁸ – Global Unique ID, generated at the external interface layer, used to identify the login session or request
- **Application / Domain ID** – created by the application or domain in which the called service resides.
- **Message ID** – an identifier that describes the kind of request.
- **Service ID** – a unique identifier populated by the service called.
- **Transaction ID** – a unique identifier generated by the external interface layer, to pinpoint the specific request. This ID should be recorded in the logs of all services that are subsequently invoked by the request.

²⁵ For service requests from internal users the identification the ID would be a USERID. For external service requests the user identification might be an external system USERID, and external system ID, or possibly an organization name (e.g., FedEx).

²⁶ Note that effective logging will also require highly accurate timestamps that are synchronized across multiple systems in multiple locations to ensure that the logs can be replayed correctly. Therefore Coordinated Universal Time (UTC) should be used.

²⁷ While NIST 800-53 suggests an event outcome field, it would be beneficial to additionally record the reason for a failure (e.g., authentication failed, user not authorized, certificate expired, etc.).

²⁸ If a System ID or Device ID is incorporated in the GUID then the device or system can independently assign GUIDs and assure that they are unique without requiring access to a central source to assign the IDs.

- **Business ID** – the key of the affected business object (e.g. entry number or passport number), if applicable.
- The following should be logged, perhaps as part of the payload, but wouldn't apply to a generic service request:
 - (URL - Uniform Resource Locator)
 - (HTTP SessionID – derived from the Portal session.)
 - (HTTP SessionID – derived from the Web browser session)

There are two philosophies that can be taken when a log entry is made. The first is to log all user interactions with the system – all of the user inputs and system responses. This captures what the user 'tried' to do and the data that the system provided to the user, but does nothing to provide any information generated internally by the system that is not returned to the user. While this approach might satisfy the legal requirements for logging, it does support the need for post-hoc problem diagnosis and performance reporting.

The second approach is to have logging performed by the services. If each of the services is required to perform its own internal logging, large numbers of duplicative records are written to the log, because the composite nature of services causes large numbers of services to be called. However, if all interactions must go through some common set of points, then, logging may only be required at those points. Since all interaction with a domain, or with external entities, must pass through a single domain gateway for authentication and initial authorization this can also serve as the point where requests must be logged. In this scheme, the portal and external interface would be one of the gateways (the gateway to the component fortress) at which logging would occur, satisfying the need to record user interaction.

In addition to logging the entry and exit of service calls at fortress boundaries, log records should be created to document the cause of service failures. The payload of such log entries would include information returned from the database, operating system, or other service showing the cause of the failure. Further, nothing precludes additional logging that may be needed for an application.

Note that it is not practical to log just at the portal/external interface because this would not appropriately support the additional requirements for error diagnosis as that data would not necessarily be available at this tier. Likewise, it is not helpful to place logging responsibilities at the ESB transport layer since the ESB supports multiple, disparate transports, several of which do not lend to effective logging.

8.5.2. Recording Log Data

Log data generated should be recorded by a single, well-defined log service. This service should be uniformly available enterprise-wide and must offer extremely high performance and availability, lest it become the limiting factor in system performance. The service must also support IA and system operation requirements for retrieving log data.

This does not imply that there need be only one repository for log data, although there should be a sharply limited number of places that logged data is stored. The high volume of requests to the log service may make it impractical to offer just one instance of the service. Regardless of the approach (one or several repositories), there must be one service to obtain complete historical log data.

9. SOA Services Programming

An SOA is an approach for designing, developing and deploying distributed business systems based on encapsulating business functions that can easily be accessed in a loosely coupled fashion. An SOA allows multiple organizations to build service independently, which can then be combined through orchestrations to build process flows that perform major business functions. An SOA is not a:

- Product.
- A specific technology.
- An application.
- A specific standard or set of standards.
- A specific set of rules.

Implementing services as part of an SOA uses established distributed computing and messaging concepts as well as some newer standards based COTS products to implement the series of functional business services. The service and infrastructure that are built scale-up (out) as well as down and can be used to provide functionality at the VA, business area, MI, and application system level as well as allowing services to cross business areas and MIs to create an extended system. The implementation of services within an SOA will rely on infrastructure that manages events and support a number of messaging paradigms (e.g., request / reply, publish / subscribe etc.).

9.1. Services Programming Model

Services are independent functional components. They expose some set of data or perform some piece of business functionality. In many respects programming using services is similar to object oriented programming except for one major area. A key concept of the object oriented approach to programming is the concept of inheritance in which as objects are defined; they inherit their attributes

from their parents. Because of inheritance objects are defined at the higher level and decomposed down.

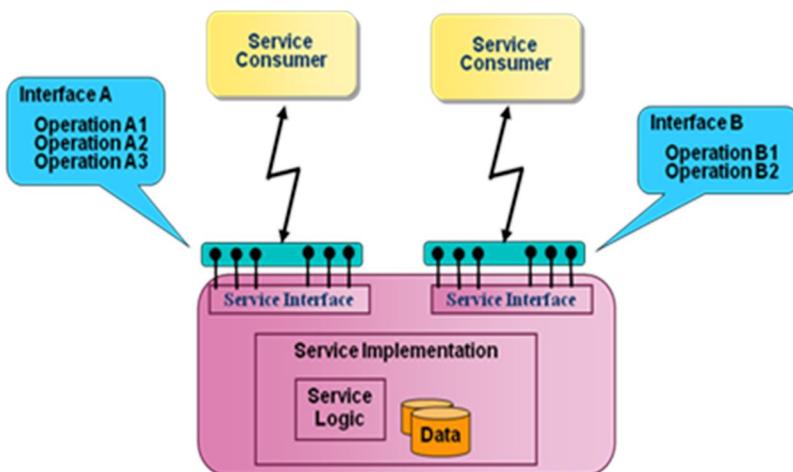


Figure 15 - Service Providing Multiple Functions and Multiple Interfaces

Services are developed based on a compositional model in which smaller services are combined into larger more complex services. Aside from the fact that the object model is based on decomposition and the services model is based on compositions, there is no notion of inheritance in the services model. A composite service is an orchestration of atomic services,

an atomic service(s) and (a) composite service(s) or multiple composite services. The attributes of the composite services are not related to the attributes of the lower level services.

Programming with services is basically much like programming based on messages passing. Basically, services are requested by the service consumer sending a message to the service to the service provider. Based on receipt of the message the service provider executes the service. The service provider then responds with a message to the service consumer. This is much the same as programming in any message passing environment.

9.2. Publishing Services

Services appear to the service consumer as “black boxes” in the sense that the service consumer has no visibility into how the service is performed. The service provider is free to change the manner in which the service is provided as long as there are no changes to the external behavior of the service.

The service provider publishes information about the service in a service directory. The UDDI standard provides a minimal set of information about a service, but most registry / repository services provide capabilities beyond those specified by the UDDI standards. The UDDI registry / repository is used to store copies of all metadata related to the services. The service consumer’s finds the service in the registry. Unlike other SOAs in which multiple services performing the same function could exist, VA only allows a single service to perform a specific function. Therefore, the registry is used to allow services to be relocated.

The Service provider includes business logic and data required to perform the service. The service consumer makes the service request as a function call and talks to the service through the service interface.

A single service can expose different service interfaces (

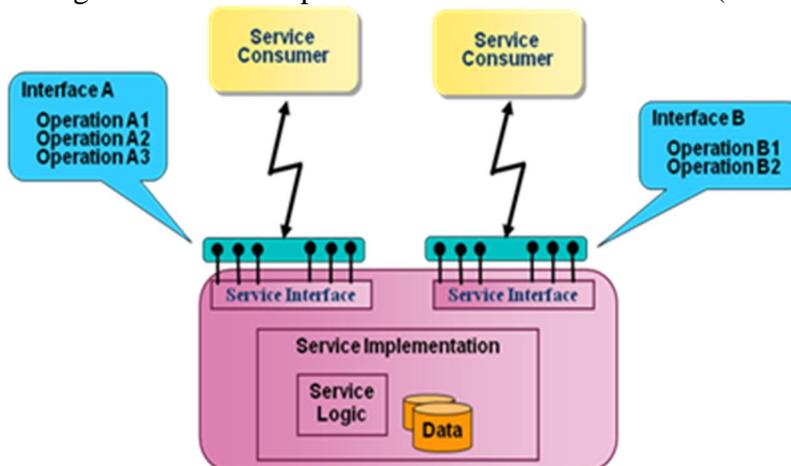


Figure 15) to different users and provide different functionality over those interfaces. The service consumers sees the individual interfaces as separate services and has no knowledge as to whether there multiple services, each performing a single function or whether there is one service exposing multiple functions through multiple interfaces. The manner in which services are provided is at the discretion of the service provider.

9.3. Exposing Web Services

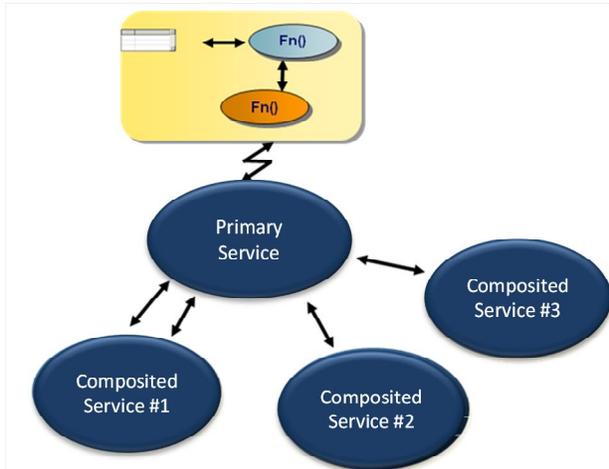


Figure 16 - Orchestration of a Service

Much like a web page, services are stateless. They are called, they are executed and they go away. While the underlying services themselves are stateless, orchestrations of services are stateful and can include control logic. That orchestration engine maintains state for the duration of a service execution. Since the execution of an orchestrated service may include the execution of a large number of constituent services, and even include manual steps, the execution of a single orchestrated service can take a long time, minutes, hours or even days. The orchestration specifies the flow of control and the sequencing of services.

9.3.1. BPEL Processing

One of the primary functions of the VA ESB will be to orchestrate services. Although not required the VA SOA is based on the use of BPEL as the basic orchestration mechanism. BPMN provides a graphical model for business process flow and control and generates BPEL descriptions of the process as output. Products supporting BPMN typically have a “drag and drop” graphical user interface to specify the workflow and the sequencing of service requests. The output of the BPMN product is an XML (BPEL) description of the workflow which can then be executed by a BPEL engine.

BPEL provides a variety of flow control and service sequencing capabilities, obviating the need for this control logic to be included in the services. The BPEL engine produces an XML document specifying the manner and order of execution of those services. Although these services will primarily be XML both the BPEL engines and the VA ESB support non-XML based services. The services whose execution is requested could be arbitrary programs that are referenced from within the BPEL orchestration.

A service consumer accesses the orchestrated service (

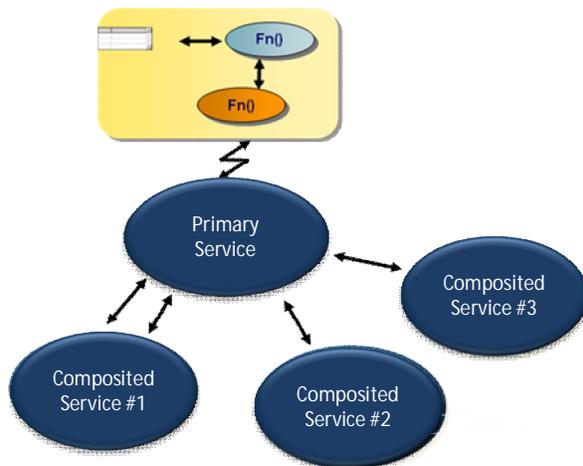


Figure 16) in the same manner as it does an atomic service, and in fact, there is no way that a service consumer would know whether a service were atomic or orchestrated. When an orchestrated service

is requested, the BPEL execution is started. In an orchestrated service, the main service is started with a request for information (e.g., a Person's name and date of birth). This information is then used to generate service requests for a number of services. The owners of the services making up the composite service are then processed using their own local business and control logic accessing their own data. The main service then must correlate the responses so that the responses associated with each of the composited services are included in the response and compose the responses into a single response.

9.3.2. XML Transformations

While the VA SOA supports non-XML based services and service requests, SOA many of the services are expected to be exposed and accessed using XML. One of the main goals of BPEL processing is to create a new XML result:

- From pieces of several other XML documents
- From expressions
- From embedded Java
- From BPEL Variables

Nearly all require use of XPATH and XSLT. Thus, XPATH and XSLT processing becomes a critical components of the ESB processing and its ability to handle services.

9.3.3. XSLT

XSLT is an XML-based language used for the transformation of XML documents. XSLT is designed to transform XML documents into other XML documents. The original document is not changed; rather, a new document is created based on the content of an existing one. The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text. XSLT is most often used to convert data between different XML schemas or to convert XML data into HTML or XHTML documents for web pages, or into an intermediate XML format that can be converted to PDF documents.

As a language, XSLT's origins lie in functional language, and in text-based pattern matching languages in the tradition of SNOBOL and *awk*. Its most direct predecessor was DSSSL, a language that performed the same function for Standard Generalized Markup Language (SGML) that XSLT performs for XML. XSLT can also be considered as a template processor.

9.3.4. XPATH

Xpath (Xquery) is to XML what SQL is to relational DBMS'. Xpath (XML Path Language) is an expression language for addressing portions of an XML document, or for computing values (strings, numbers, or Boolean values) based on the content of an XML document. The Xpath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria. In popular use (though not in the official specification), an Xpath expression is often referred to simply as an Xpath. Originally motivated by a desire to provide a common syntax and behavior model between Xpointer and XSLT, Xpath has rapidly been adopted by developers as a small query language.

9.4. **Service Contracts**

Services are defined by a contract between a service provider and a service consumer. The relationship between the service provider and the service consumer is defined by a contract that specifies the nature of the service and the terms under which the service is provided. The service contracts are defined in terms of a Web Services Description Language (WSDL) description of the service. WSDL is one of the Web services standards and provides an XML description of the service. Even if the service is not a Web service and the message transfer will not be XML, an XML-based WSDL description of the service can still be provided.

The WSDL will also describe the message sets that will be used either in terms of an XSD or Document Type Definition (DTD), both of which are XML descriptions of the message transfer. The NIEM specification includes a number of namespaces, many of which are very large; therefore, if NIEM is being used, care must be taken to include only a small subset of the full NIEM set of namespaces.

The service contract also includes all required information regarding performance, response time, availability, etc. It documents all of the agreements between the service provider and the consumer. There may be occasions where specific services will require the inclusion of special performance, availability or other factors. These will also need to be specified in the service contract.

9.5. **Programming Synchronous vs. Asynchronous Services**

Synchronous services are the simplest form of service to program, develop and test. The client issues a request and blocks until the server has responded. The paradigm is much the same as the traditional Remote Procedure call (RPC). Because the requester waits until it receives a response there is no need to “correlate” responses with requests. Correlation is the process of aligning service responses to requests so that the correct responses are provided to each request. Synchronous services generally would use HTTP or RMI and operate in a standard request / response mode. The use of synchronous services can tie up large amounts of resources because of the potentially large number of open service requests. Asynchronous communication is the preferred approach for Web services, but it is not always possible to use asynchronous messaging.

With asynchronous services the client drops the thread when the request is launched. In this case, the ESB must maintain the state of the service request until the response is received. The term dehydration is used as the process by which state is saved and stored until the service request response is received. The dehydration process and the re-hydration process when the response is received are performed by the infrastructure and are transparent to the service developer.

If a service request fails, there may be a desire to send a message back to the originator in response to the failure. In cases in which the service request was long running or in which it was not expected that there could be a service failure, there may be no person or service to which to send the response. It will be the responsibility of the service to fully define the manner in which such service failures will be handled. It may be that the failure is logged, a message may be generated, or other actions may be taken. The service will need to define the action that it will take and assure that the service consumer is aware of those actions.

9.6. **Atomic Services**

Not all services need to be orchestrated. Atomic services are services that do not include other services.

9.7. **Orchestration**

Orchestration is the primary way in which services are composited to produce large more complex services. Orchestration provides flow control for services by controlling the sequence of execution. Composite as well as atomic services can be orchestrated allowing the orchestration to continue to any level.

Orchestrations are most commonly specified using a COTS tool known as a BPEL engine. Business Process Execution Language (BPEL) is an XML based language for describing business processes. These orchestrations can then be modeled or executed. This allows solutions to be quickly assembled tested and put into production. The BPEL engines also provide graphical tools for building Extensible Stylesheet Language Transformations (XSLT) which are needed for processing XML based messages. The engines also provide a rich set of resource adaptors – which provide a wide range of “partner links.”

9.8. **Compensating Services**

While services themselves do not maintain state, an orchestration may have state which is kept for the duration of the transaction. State management is kept for a single request / response pair for the service being called in a manner similar to that of the way session objects maintain state in a web server. Managing state is, perhaps, the hardest part of the developing orchestrations. What makes the management of state complex is that there is few transaction management capabilities yet defined in either the Web services or BPEL standards. This causes problems in the development of services. To some extent constructs such as “Try / Catch” can be used to provide some level of error recovery. However, these constructs do not provide the full level of recovery that is required. Because services are long running they cannot use two phase commit. This means that some services will have already executed and completed – possibly changing one or more databases before other services in the orchestration fail. However, it is not usually possible to roll back the services.

Cleaning up after failed services in an orchestration typically involves the execution of compensating services designed to undo the effects of previously completed services. This may, for example, require running a service to delete a record that a previous service had added. What makes managing state so hard is that the developer must be able to determine all the possible states the orchestration could be in for all possible failure modes and must identify the compensating transactions that must be developed and then those transactions must also be developed and tested.

9.9. **Correlation**

Correlation is putting the right response with the right request. A new thread must be started when the service resumes. However, when asynchronous services are used, there is no assurance that the service requests will return in the same order in which the requests were made. Therefore, the responses must be correlated so that they are matched with the correct request. This process will be managed by the service developer. The other situation in which the service developer will need to correlate responses will be when a service is implemented as an orchestration of multiple other services and each service request contains multiple subjects. Then the individual subjects must then be correlated so that the information about each of the subjects from each of the services can be combined together.

There are three standard techniques for message correlation:

- **WS-Addressing** – Uses the appropriate XSD to perform header correlation.

- **Messaging system correlation** – For services based on the use of a COTS messaging product, many of the COTS messaging products correlate messages based on their headers.
- **Body correlation** – Body correlation can be done at the BPEL level for XML based messages. The correlation value is found and extracted by XPATHing into both the request message and response to locate and extract a unique value.