Services and Components Based Architectures

A Strategic Guide for Implementing Distributed and Reusable Components and Services in the Federal Government

Version 3.5 Chapter 1: Executive Strategy

Last Updated: January 31st, 2006

Architecture and Infrastructure Committee,
Federal Chief Information Officers Council

January 2006

The Federal CIO Council Architecture and Infrastructure Committee

Components Subcommittee

In collaboration with the

Federal Enterprise Architecture Program Management Office

and

The Industry Advisory Council

Present:

Services and Components Based Architectures
Version 3.5: January 2006

Contributors

In alphabetical order:

- James Benson, AIC Contractor Support
- Richard von Bostel, OMB Contractor Support
- L. Reynolds Cahoon, National Archives & Records Administration
- Nathaniel A.F. Clark, AIC Contractor Support
- Josiah Cushing, AIC Contractor Support
- Bobby Jones, *Department of Homeland Security / Federal Emergency Management Agency
- Karen Kaye, Nuclear Regulatory Commission
- David R. Mayo, Industry Advisory Council/Everware
- Marion A. Royal, General Services Administration
- Adam Schwartz, Office of Management and Budget
- James W. Smith, Office of Management and Budget

* Services and Components Based Architectures Committee Leader

Executive Summary

This document serves as an "Executive Strategy" for planning and implementing modern information technology (IT) architectures within the Federal Government. The specific architecture it describes, **Services and Components Based Architecture (SCBA)**, leverages the Federal Enterprise Architecture (FEA) and builds upon the concepts, principles, and benefits of **Service Oriented Architecture (SOA) –** an architecture designed to maximize the reuse of components and services and one of the most promising and widely accepted architectural approaches to-date. SCBA represents a practical, results-oriented, approach to modernizing enterprises. It is intended to help organizations reduce long-term costs, improve quality of service, improve information sharing, and help achieve a vision of flexible business processes supported by customer-focused applications, which can be altered in a matter of days instead of months. SCBA builds upon traditional SOA principles in three ways:

- it is tightly **integrated with the Federal Enterprise Architecture**,
- it provides a **description of what the architecture is** (a collection of services designed and implemented to achieve an organization's mission), and
- it **identifies the organizational, cultural, and process elements**, as well as technological elements, that need to exist for these architectures to be successful.

The most important aspect of SCBA is its focus on **reuse of services and components – better referred to as Service Components**. **Service Components** are information technology assets that perform useful business functions through a well-defined interface. The main advantage of Service Components is that they enable practical reuse of assets both within and across organizations. Service components are superior to traditional software components in the following ways:

- one copy of the Service Component may be shared among all consumers, eliminating the need to manage and support multiple versions on different servers,
- the Service Component can be used by consumers on any technical platform (via a standard interface) eliminating the need for platform-specific versions, and
- the asset can evolve and improve without requiring consumers to modify their business processes or interfaces, since changes to the internal implementation of the component can be made without affecting the interface.

Despite its emphasis on services, SCBA still accommodates the concept of component reuse. Specifically, component reuse is necessary for those situations where cross-agency service sharing is not possible due to regulatory or security restrictions. Finally, SCBA emphasizes changes both in technology and in the following areas:

- **Policies**: the organization needs to alter its policies to support reusing assets from any source, and set specific, measurable goals for levels of reuse.
- **Strategies**: the organization needs to move from strategies that are narrowly focused on programs to ones focused on producing and integrating reusable services across the entire Federal government.
- **Processes**: the organization's software development and capital planning processes need to be altered to make looking for opportunities for reuse a core task.
- **Culture**: the organization's culture needs to change through a combination of executive recognition and incentive programs that strongly reward reuse.
- **Governance**: the organization's IT governance processes need to change to take into account that a service may be used by multiple organizations, not just local users, and put appropriate service level agreements in place.

This document is the first in a series of chapters that fully describe SCBA. Later chapters will further detail its technical and process characteristics and are described in Appendix B.

Document Replaces or Supersedes

This document is intended to replace the "Service Component-Based Architectures, Version 2.0" specification. This version of the document contains only the first of nine chapters that will fully describe SCBA. It describes a ready-to-implement strategy for implementing these architectures and explains their advantages. Later chapters will provide detailed descriptions of the technologies and processes that enable these architectures. Further details are provided in the "Intended Audience" section.

Intended Audience

This document is intended for individuals in various roles in government organizations. It is relevant to any individual interested in making better use of Federal system and process assets, but it specifically addresses the interests of:

- **CIOs, CTOs, and other Executives –** interested in innovative approaches to improve performance, reduce cost, and enable flexibility of their organization's information systems.
- **Functional / Business Line Managers –** focused on fielding systems that best support their mission and business needs and achieve the highest return on their IT investments.
- **Capital Planners –** responsible for defining and funding Service Components, using IT Exhibit 300s to support capital planning and investment control (CPIC). These documents target Federal projects or programs that may benefit from cross-agency collaboration and the reuse of agency assets.
- **Enterprise Architects –** responsible for the definition and target planning of an Agency's Enterprise Architecture, working with a variety of architectural implementations (e.g., SOAs; FEA reference models; intergovernmental architectures, such as the National Association of CIO's Enterprise Architecture Development Tool-Kit, etc.).
- **System and Solution Architects –** responsible for building and assembling Service Components that leverage existing capital assets, business services, and data across the government and industry.
- **System and Process Engineers –** tasked with implementing reusable services and modifying systems and processes to be reusable by others.

Given the differing focus areas of these individuals, SCBA has been organized into distinct chapters. Each chapter is specifically targeted at the needs and concerns of a sub-set of the overall audience. "Appendix B: Chapter Guide" describes each of these chapters and their intended audience.

Table of Contents

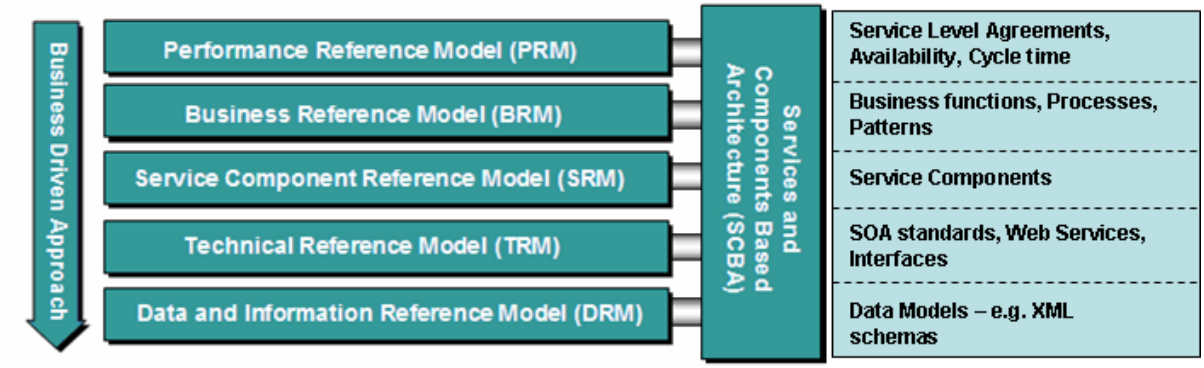
Index of Figures

1 Introduction

1.1 Overview

To facilitate efforts to transform the Federal government into one that is citizen-centered, results-oriented, and market-based, the Office of Management and Budget (OMB) has developed the Federal Enterprise Architecture (FEA). The FEA is a business-based framework for government-wide improvement. As illustrated in Figure 1, it takes the form of a collection of interrelated "reference models" designed to facilitate the identification of duplicative investments, gaps, and opportunities for collaboration within and across Federal agencies.

Figure 1 - The Federal Enterprise Architecture Reference Models



This document, developed by the Components Subcommittee of the Federal CIO Council Architecture and Infrastructure Committee (AIC), seeks to complement the FEA by acting as a practical guide for how to realize the benefits of business and software agility through the integration of Service Oriented Architecture (SOA) and Component-based architecture. It specifically corresponds to the FEA Service Component Reference Model (SRM), and provides an executable strategy, describing the organizational, cultural, process, technological, and systems changes needed to realize the benefit and outcome of the FEA and the SRM.

1.2 Background

In July 2002, the Federal Enterprise Architecture Program Management Office (FEA-PMO) released the Component-Based Architecture (CBA) Specification Version 1.0 to help the government understand the concepts behind re-usable components and their association and linkages to the Federal Enterprise Architecture (FEA) and newly formed e-Government initiatives. This document was the first step in helping the government understand the importance of re-usability and identified a suite of specifications, architectural frameworks, and interoperability guidelines that led to the creation of the FEA SRM and Technical Reference (TRM) models.

In February 2004, the Architecture and Infrastructure Committee (AIC) – in coordination with the FEA-PMO and the Industry Advisory Council (IAC) refreshed the CBA specification and released a document titled "Service Component-Based Architectures, Version 2.0". This document built upon the CBA specification by expanding on the importance and value proposition of re-usability, explaining its alignment to the FEA, and creating a technical foundation to support government-wide improvement.

Today the AIC is engaged in an activity to update the Service Component-Based Architecture specification in order to reflect both the evolution of the FEA and new technologies and architectural frameworks that have emerged and rapidly gained mainstream adoption. It is a further goal to alter the presentation of this architecture into a format that is simple, actionable, and provides tailored feedback to readers in roles varying from executives to developers.

1.3 The Future of Government

Over the last 25 years, information technology (IT) has had a tremendous influence on how large organizations operate. It has acted as both a "conduit" allowing disparate processes to interconnect and as an "enabler" allowing both new and old business processes to operate at never before seen speeds, scales, and efficiencies. The Federal government has been a particular beneficiary of IT, with almost all agencies implementing systems that have improved their ability to execute internal business processes. The evolution to an "e-enabled" government has progressed to the point where business processes not supported by IT systems are rare, and IT departments focus more on system improvements than new implementations.

The Federal Government is now advancing to the next stage of the "e-government" evolution in two ways:

- moving from government-centricity to **customer-centricity**, and
- moving from rigid business processes to **agile business processes**.

Today many government IT systems are traditional, pre-IT business processes translated into an IT format. Further, they focus only on the needs of the particular agency or program they are intended to support. The move to customer-centricity recognizes that in the view of citizens the Federal government is a single organization. At the discretion of the citizen, information given to one agency should be made available to all. If a process is e-enabled at one agency then it should be e-enabled at all agencies. While statutes and regulations do limit how much information can be shared – these limits are not common, and the potential benefits in improved speed and quality of service are substantial. The vision of **customer-centricity** is for no citizen to have to go to more than one location to accomplish a task or have to enter data twice.

> "The federal government can secure greater services at lower costs through electronic government (E-Government) and can meet high public demand for E-Government services. The goal is to champion citizen-centered electronic government that will result in a major improvement in the federal government's value to the citizen."
>
> *The President's Management Agenda*

The increased speed and scale of IT-enabled processes has been met with a proportional increase in the demand to modify processes to meet changing conditions. IT has accelerated the speed at which citizens, businesses, and organizations both operate and change. As more integrated government solutions evolve, citizens' expectations of government services rise. Government needs to be able to respond at an equivalently accelerated rate. Currently, most business processes cannot be altered without extensive alterations to the IT systems that enable them. These alterations are both time consuming and expensive, often taking months to complete. The vision of **agile business processes** is that changes to existing business processes will only take days to execute.

The fundamental shift that will allow these visions to be realized is a move to **Service Component-Based Architecture (SCBA**). Today, the IT industry has generally accepted **Service Oriented Architecture (SOA)** as the most promising architectural approach to-date.

SCBA complements traditional SOA approaches and is designed to provide an optimal, long-term service-oriented approach aligned with the FEA that recognizes the value of component-based service delivery. SCBA builds upon SOA in three key ways:

- it is tightly **integrated with the Federal Enterprise Architecture**,
- it provides a **description of what the architecture is** (a collection of services designed and implemented to achieve an organization's mission), and
- it **identifies the organizational, cultural, and process elements**, as well as technological elements, that need to exist for these architectures to be successful.

SCBA also treats business processes and the IT systems in the same way, allowing both to be reused across organizations. In order for a business process or technical system to be a "Service Component," and thus participate in the overall architecture, it must offer a well-defined interface with well-defined functionality. These two characteristics represent the minimum criteria needed for a business process or technical system to be reused. Service Components are intended be a subset of the "components" defined in the FEA SRM. The SRM does not require rigorous interface or functionality descriptions, only systems and processes with these descriptions are both SRM Components and SCBA Service Components. SCBA attempts to realize the potential of the SRM by requiring that business processes and IT systems be designed or modified to make them easy to reuse. Later sections of this document will describe SCBA in further detail.

1.4 The Value Proposition

Successful SCBAs will greatly enhance Federal agencies' ability to accomplish their fundamental mission of serving customers (e.g., citizens, other agencies, other levels of government, and industries). SCBA, through its focus on both reuse and on the flexible composition of Service Components into specific solutions, delivers the vision of agile business processes by reducing the cost and time needed to make business process changes. As business processes change, the services supporting them can be evolved or replaced. Since business processes and IT systems can be reused across organizations, costs for development and maintenance of similar systems at multiple agencies do not need to be replicated. SCBA also helps to achieve the **customer-centricity** vision by focusing on the modeling of both data and business processes from a customer point of view. As an example, SCBA could potentially speed a government response to a major natural disaster. New benefits programs could be more quickly deployed by reusing processes and IT services that support existing benefits administration programs. Business processes and supporting IT systems could be more quickly adapted to meet needs encountered personnel in the field.

> "The President's Management Agenda and the E-Government Act of 2002 identify the overall goals for implementing E-Government: to better perform government services, and at lower cost. This SCBA paper lays out an approach that can be used to help accomplish both. While agencies are not required to use the approach described in this document, a services and components-based approach is an essential piece of an agency's target architecture. As such, SCBA is included as a criteria in version 2.0 of the EA Assessment used by FEA PMO to evaluate federal agency EAs in 2006."
>
> *Dick Burk*
> *Dir., Federal Enterprise Architecture Program, OMB*

Experience with component-based architectures has shown that reuse can be successful when the reuse efforts focus on large-scale components in a collaborative environment that includes system owners, capital planners, business leaders, and enterprise architects. SCBA focuses on exactly this type of reuse.

1.5 Current Initiatives and How They Support This Vision

SCBA is directly supported by many major current Federal initiatives. These include:

- **Capital Planning and Investment Control (CPIC) and eCPIC:** The OMB CPIC process requires all Federal proposed IT projects be centrally evaluated and approved. Each of these business cases should be evaluated to ensure that they are not duplicative, and to look for reuse opportunities. Many agency business cases are accessible and searchable through the eCPIC system. It specifically supports SCBA by providing a mechanism by which initiatives can be evaluated to discover if they are duplicative to other, pre-existing Service Components. eCPIC can be accessed at: http://www.ecpic.gov/.

- **Core.gov**: The "Component Organization and Registration Environment," or "Core.gov," is the central system for registering Service Components across the Federal government. It provides a mechanism for the discovery of pre-existing Service Components, publication of new ones, and collaboration over their use. CORE.gov also incorporates a vetted submission process for reviewing and approving components. It supports SCBA by facilitating Service Component discovery. Core.gov can be accessed at: https://www.core.gov/

- **e-Government Act of 2002**: The goal of the e-Government Act of 2002 is to enhance the management and promotion of electronic Government services and processes. It establishes a broad framework of measures that require using Internet-based information technology to enhance citizen access to Government information and services. A copy of this act is available at http://thomas.loc.gov/cgi-bin/query/z?c107:H.R.2458.ENR:

- **FEA Assessment 2.0**: The Federal Enterprise Architecture Program Management Office (FEAPMO) has created version 2.0 of the federal EA Assessment Framework. This framework serves as the basis for EA maturity assessments performed by OMB. It helps OMB and agencies assess how well EA programs guide and inform IT investments in support of agency strategic objectives. SCBA is related to several specific criteria in the framework's capability areas and outlines an approach that can be used to achieve the outcomes identified in the assessment. The assessment criterion with the most obvious relationship is 'Service Component Architecture' (section 1.3.4 in the assessment), within the 'Completion' area. The higher levels of maturity (levels 4 and 5) for this criterion require outcomes that are addressed by SCBA. These specifically require: 1) the existence of a target Service Component architecture, 2) that agency SDLC and CPIC processes address the standardization and reuse of components, 3) that Service Components are monitored, and 4) that Service Component reuse be measured. A copy of the FEA Assessment is available at http://www.whitehouse.gov/omb/egov/a-2-EAAssessment.html.

- **Federal Enterprise Architecture (FEA):** The FEA is a business-driven framework designed to facilitate government-wide improvement. It provides a framework to categorize and classify IT investments to support the identification and discovery of re-usable assets. The five FEA reference models (BRM, SRM, DRM, PRM, TRM) directly support the development of a service-oriented architecture. More information on the FEA is available at: http://www.egov.gov

- **Federal Enterprise Architecture Management System (FEAMS):** is a web-enabled system that provides agencies with access to government-wide initiatives aligned to the FEA. The objective of FEAMS is to promote sharing of information about

approved IT investments among federal agencies to identify opportunities for cross-agency collaboration and reuse. More information on FEAMS is available at: http://www.feams.gov

- **FirstGov**: FirstGov is an enterprise portal that provides a common web interface for the discovery of all Federal citizen-centric IT systems and services. FirstGov directly implements the vision of customer-centricity and is a very public example of reuse in action. FirstGov can be accessed at: http://www.firstgov.gov/

- **Presidents Management Agenda (PMA):** The President's e-Government Strategy has identified several high return government-wide initiatives to integrate agency operations and information technology investments. The goal of these initiatives is to eliminate redundant systems and significantly improve the government's quality of service. SCBA directly supports these goals. A copy of the PMA is available at http://www.whitehouse.gov/omb/budintegration/pma_index.html.

2 Services, Components and Architecture

To achieve the vision described in the previous section, significant business, process and technology changes are required. In this section, we present an overview of the basic concepts that provide the foundation for these changes. In some ways, the changes are profound; in other ways, the changes are part of a natural evolution.

2.1 The Evolution of Systems Development – Increasing Abstraction

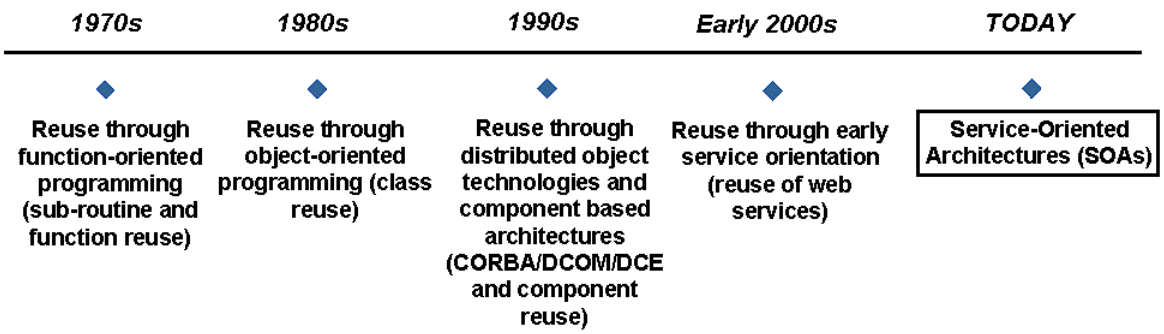
Since the beginning of software systems development, progress has been measured in terms of increasing levels of abstraction. The concept of abstraction is similar to that of modeling in that it presents a simplified view of something – depicting only the relevant aspects and ignoring unimportant detail. However, in abstraction, there is a conscious effort to generalize as you simplify, resulting in solutions applicable to a broader scope than the problem analyzed and more suitable for reuse in similar problem spaces across a broader range of domains. The concrete result of abstraction in software development is a reduction in the number of "lines of code" required to accomplish a given task – each line of code accomplishes a greater amount of work. For each major advance in software development, a significant decrease in the lines of code was achieved.

Figure 2 shows the evolution of reuse in software development. From the 1970s through the 1980s software development progressed from "machine language" to assembly language, to higher level, compiled languages (known as 3rd generation languages) to 4th generation languages and CASE tools (computer assisted software engineering), the level of abstraction increased dramatically. These gains were due to abstraction applied within the languages and tools used to develop software, thereby resulting in order-of-magnitude increases in software developer productivity.

In the mid-90s, an additional development emerged: component based development (CBD). CBD takes the concept of abstraction in a new direction. Rather than reducing lines of code, CBD separates various aspects of the functionality into isolated units that can be produced and managed independently of the other aspects. This again allows a developer or consumer of the functionality to deal with one aspect at a time – ignoring the other aspects. In a component-based architecture the various aspects are organized into layers, most commonly: presentation, orchestration, business logic, data management, security, and infrastructure.

Service-oriented architecture represents a generalization of the component model by dealing directly with what is offered, rather than how it is packaged. The tiered architecture enables components to be easily incorporated in solution architectures. The evolution of services standards (e.g., WSDL, UDDI, and SOAP) and the maturity of distributed computing architectures (e.g., Java Enterprise Edition and .Net) have enabled "single-copy reuse" via shared services. Thus, SOA is a tiered framework that empowers solution developers to employ abstraction techniques, at all architectural tiers, without having to struggle with many of the interoperability and multiple implementation challenges faced by previous generations of reuse proponents. The future of software development will likely consist of complete assembly of applications from services and components – often referred to as "true software manufacturing."

Figure 2 - Evolution of Reuse in Software Development



2.2 Components and Services

Most mature industries eventually evolve to a component-oriented paradigm. For example, the automobile industry uses components (also known as modules or assemblies) to manufacture cars. In fact, the final production stage is known as "assembly." In the housing construction industry, the same phenomenon is present: roofs are created from trusses; window and door assemblies are pre-manufactured and installed on site. Even the personal computer (PC) hardware industry relies on a component approach, which permits the upgrading of individual pieces of the PC without affecting the rest of the computer. The component approach is effective and efficient – both from the design and production perspective as well as the consumption perspective. For example, it is usually more efficient to design an engine management component for many car models than to design a new one for each model. In addition, assembling components to produce cars is more efficient than handcrafting automobiles.

2.2.1 Components

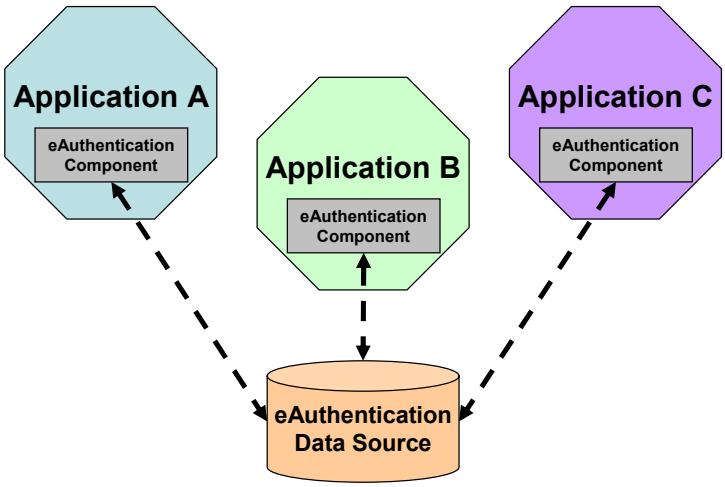
The software industry has tried to emulate this approach, but until recently the industry standards to enable it did not exist. About a decade ago, the component based development (CBD) movement began to create standards to apply in specific cases, but the standards were not sufficiently broad to enable widespread adoption of the approach. More recently, the technology and standards have matured to the point that CBD is a viable and common way to develop software applications in the commercial world.

Software components are units of software that provide business or technical functionality. These units are independently deployable; that is, they are self-contained and can be deployed virtually anywhere on the network. Business components execute business logic, enforce business rules, and manage corporate data. Technical components provide the platform or infrastructure capabilities that the business components rely on such as messaging, error handling, security, etc.

Software components are the reusable building blocks for application development. A software component typically consists of: (1) a specification (process and data model representing the user or consumer's view) that defines what the component does, (2) an implementation which is the internal design for the component, (3) an executable (run-time) module that gets deployed, (4) one or more interfaces that provide access to the component's functionality. The key concept behind components is that the implementation is hidden behind the interface – the consumer of the component does not need to know the details of the implementation to exploit the capabilities offered. Thus, in general

components are an approach to provisioning capabilities that are highly flexible. Component-based reuse is illustrated in Figure 3.

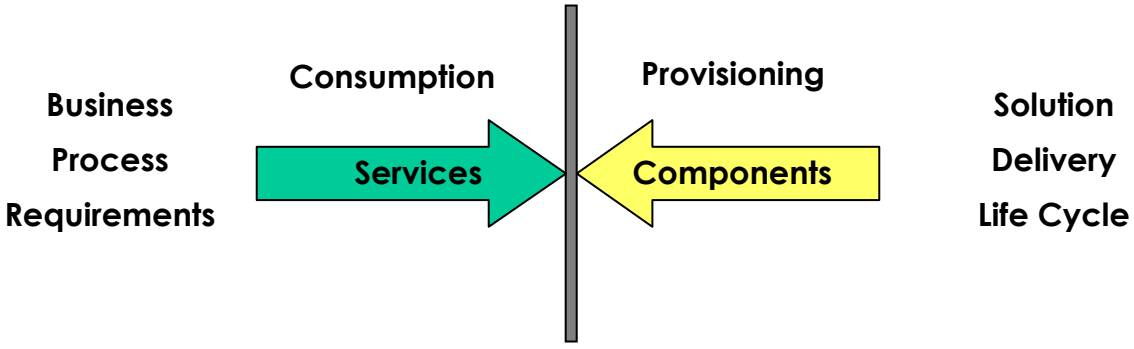
Figure 3 - Illustration of Component-Based Reuse



2.2.2 Services

Services are focused on satisfying business or technical requirements based on a **provider/consumer** model. Services represent a broader concept than components. They are the activities executed in response to a request (or an event) in order to deliver some result. Both concepts employ the notion of an interface that defines the set of activities (or services) offered. However, whereas all components offer functionality as services, not all services are implemented as components. Figure 4 illustrates the distinction between services and components – services are driven from business requirements, whereas components are a method of providing services. For example, one way to implement a service is to put an interface on some legacy functionality. The legacy system may be very unstructured (and not divided into independent components), yet the interface may offer the services required by other and new applications. As discussed in the roadmap section below, creating interfaces to access services from legacy systems is one common way to begin to implement an SOA.

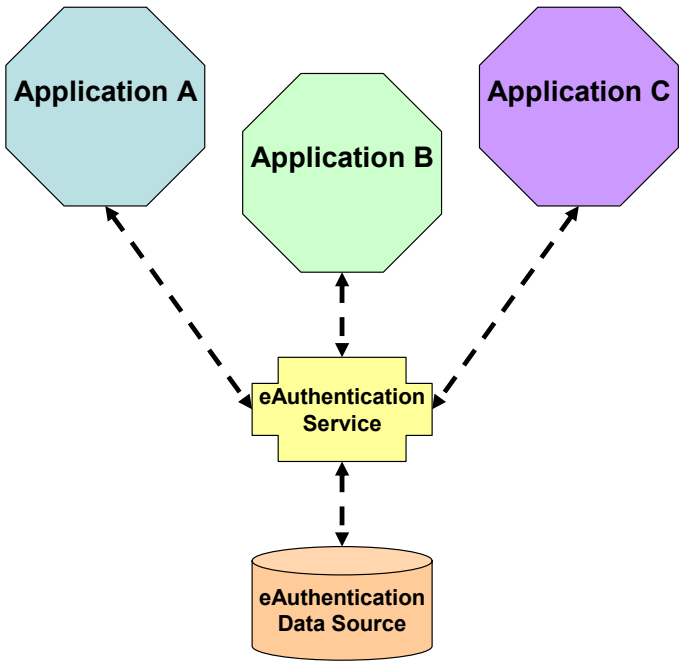
Figure 4 - Differences between Services and Components



Services and components enable **reuse**, although with slightly different twists. Components are typically designed to be redeployed or integrated into multiple different applications. In fact, commercial-off-the-shelf (COTS) components are typically licensed to be embedded in applications. Reuse is achieved by producing multiple instances of the component and building them into applications. Services can be exploited in this way, but also offer the possibility of **shared-services** – running a single instance of a service that can be called by other applications across the network, as shown in Figure 5.

An important additional concept is **service discovery**, or the ability to find (either manually at design-time or automatically at run-time) and access existing services. Many technologies exist today that enable this, and a coordinated strategy for identifying and categorizing services across the enterprise is critical. In the Federal government, the FEA provides the first step in implementing this strategy. Since services may be thought of as an abstraction of components, it is not too difficult to consider them both when describing reusable government-wide assets.

Figure 5 - Illustration of Service-Based Reuse



2.3 The Role of Architecture

There are many approaches to architecture. One approach that has generated a great deal of recent industry attention and business benefit is SOA. Although there is some confusion about what exactly SOA is, the main thing to keep in focus is that SOA is *architecture*. That is, SOA is an architectural approach to understanding and modeling a business, as well as an approach to implementing the capabilities to satisfy the business requirements. SOA is a layered architecture with services defined at the business/application-specific, common business capabilities, infrastructure, and platform levels. This layered approach allows services to be consumed in multiple contexts and allows services to consume lower level services. This layered architecture provides the greatest potential for reuse and flexibility. The ability to manage service dependencies by using the layered approach is critical to achieving agility for service-based applications. The applications that are created using the SOA approach are often referred to as "composite applications" – they are assembled, or composed, from services and the services can be replaced to change the characteristics of the application. This is conceptually similar to the reuse of identical parts by automobile-manufacturers across several car-models.

Note that SOA is often confused with **web services**. Web services represent one popular implementation approach for services, but not the only approach. Web services will be discussed in more detail in "Chapter 8 – Service Production, Discovery, and Consumption."

3 What Needs to Change

To fully embrace the goals and objectives of Service Component reuse, several strategies and supporting departmental programs need to evolve. Reuse needs to be built into every facet of the system development and integration processes. These include Enterprise Architecture, Portfolio Management, Performance Management, Capital Planning, and Cyber Security. Without these changes, missions and programs will continue to struggle to discover re-usable Service Components and negate the ultimate value proposition. This document briefly describes several areas that need to change and evolve.

3.1 Strategic Changes

First and foremost, organizations need to rethink architecture efforts to include Enterprise Architecture (EA) in the context of reuse and enabling processes and services within the Federal government. Future architectures need to be actionable, discoverable, and consumable while driving efficiency, cost reduction, ROI and the elimination of duplicative systems. This strategy corresponds with OMB direction and is one of the primary reasons the FEA was developed. Reuse should be considered as an overarching **strategy** as opposed to the outcome of an effective architecture, solution, or product. Creating and embracing such a strategy will require organizations to make the following specific changes:

- redesign business and transaction models so that they are based on collaborating services instead of silo applications,
- accommodate reuse both within and outside the immediate organization,
- embrace cross-organization collaboration (vs. enabling duplication),
- remove the barriers that create stove-pipes and build a culture that rewards and motivates reuse,
- strive to achieve true assembly by eliminating as much custom coding as possible in the development and deployment of applications, and
- create and leverage architectural patterns that offer the "best of breed" Service Components.

This strategy will require organizations to balance a top-down (Strategic) and bottom-up (Technical) approach that blends and integrates business and process demands with the availability of re-usable components, services, and IT in general. Organizations should consider how Service Components are produced and how they can ultimately be discovered and consumed – as well as what service-level agreements will be required to support their use. Finally, it is important to remember that SOA's are not produced by a specific vendor, hardware, or software product. Re-usable strategies embrace standards and interoperability, and are founded on the concept of "loosely" coupled services that increase the agility and flexibility of IT.

3.2 Policy and Organizational Process Changes

To support an overarching reuse strategy, organizations will need to evolve their traditional investment, architecture, and systems development processes. Some of these processes are listed below along with brief recommendations as to change concepts and industry patterns:

Acquisition and Procurement Process – enabling the reuse of Service Components should stimulate changes to acquisition and procurement processes in several areas. First, incentive programs should be created to encourage vendors and contractors to produce reusable Service Components. Service Components provide a mechanism to take advantage of existing shared-cost savings incentives programs, but these programs should be modified to make this explicit. They should be further modified to indicate that cross-vendor

collaboration on the production and use of Service Components provides a mechanism for further shared cost savings benefits (i.e., through "win-win-win" based incentives). Second, RFI, RFP, or RFQ processes should change to embrace reuse (e.g., by integrating reuse concepts into questionnaires and decision criteria). Possible questions to add to decision criteria include:

- Does a vendor or contractor's technical approach embrace re-usability?
- Can the requirements for this project support any other organizations?
- Will the outcome result in new Service Components that can be registered in Core.Gov?

Third, incentive programs should also be put in place to reward government program managers who succeed in encouraging reuse through Service Component based acquisition. Finally, procurement guidelines for vendor or contractor organizational conflict of interest should be reviewed relative to future procurements that may be required to leverage a Service Component that had been previously produced and registered as a reusable Service Component.

Capital Planning and Investment Control (CPIC) – traditional CPIC processes require organizations to select, control, and evaluate an investment. While these processes are still valid when embracing a reuse strategy, they will need to be supplemented with discovery and Service Level Agreement (SLA) management functions. For instance, prior to the select and control processes, organizations will need to discover what components and services are available within and outside the enterprise. Doing so will require changes to investment processes so that investment managers can better assess the applicability of a service or component relative to the demands of the program or mission. Moreover, key new questions should be asked: can these services be consumed by my business process? What is the service level agreement that governs its use and reliability? What is the net benefit of leveraging a service as opposed to building my own? Control and evaluate process will need to evaluate these SLAs to ensure they meet the existing demand of the process and are prepared to scale in the event of growth or expansion.

Service components in the DHS Enterprise Architecture

The Homeland Security EA incorporates a set of business service definitions (component capabilities) as part of the Target Architecture. These Service Components are derived in a top-down manner by clustering the elements in the target business and data architectures based on their interaction. This is possible because the business and data architectures are identified using a technique known as "parallel decomposition."

The EA Service Components are reusable building blocks for the development of "composite applications" – combinations of capabilities specific to an individual user role. The Service Components are assigned to portfolios so that the development or provisioning of the services can be managed on behalf of the entire Department. The result is a set of Service Components within an adaptable architecture. The benefit of this approach is that the applications that support business processes can easily be modified to reflect changes in the business processes themselves. In addition, the Service Components can be reused by programs across the Department, resulting in better interoperability, consistency, and cost savings.

Solution Development Life Cycle (SDLC) – traditional SDLC processes were created to manage and govern the life-cycle of major systems development. They begin with the system requirements and end with the retirement or sun-setting of the system or application. This life-cycle is typically managed within the bounds of a specific organization and does not extend outside the immediate domain. Future SDLCs will need to embrace the discovery of services or components relative to the requirements and demands of the business need. This step will

likely occur prior to the design and development activities to enable developers to incorporate Service Components into the design of the system, as opposed to adding them on at the end of implementation. Further, when developing or planning to reuse components or services, a Service Component life cycle will be required that allows an organization to effectively plan reuse, develop reusable assets, publish the components or services to a local or government-wide registry and manage them relative to the defined SLA and consumption patterns. Finally, in order to avoid interrupting dependant processes and systems, services require robust processes governing ongoing maintenance and change management. It is especially important to track who the users of service are in order to gather requirements and perspectives for modifications and fixes.

Enterprise Architecture (EA) – the SCBA strategy will influence virtually all layers of the EA with the greatest impacts on the Application and Technology layers. While baseline (as-is) architectures will not be significantly affected, organizations need to ensure that their target architectures reflect a service-based approach. Traditionally, EAs are a collection or repository of interrelated layers of business processes, applications, business information, and technical data. EAs include layers describing producers of services (e.g., business processes), and linkages into government-wide registries (e.g., FEAMS) that publish this information. Going forward, architectural elements should be tightly integrated into capital planning and economic processes so that true EA analytics can be performed on the viability and feasibility of reuse, as well as the costs and benefits of doing so. Best practice frameworks will emerge (e.g., CRM, PRM, and other Service Components in the FEA-SRM) and will be overlaid on existing EAs to assess gaps, redundancy in applications, and interoperability issues and constraints. Finally, although it is critical to have an overall target architecture, it is equally critical that organizations have an actionable and realistic transition strategy, complete with a sequencing plan.

Governance – the production, discovery, and consumption of services and components will require new policies and processes that promote and ensure compliance with reuse, service level agreements, security, and interoperability standards. Organizations will need to publish proposal processes and standards by which industry can adapt to the changing dynamics of an agile business. Design conventions must emphasize interoperability, standards compliance, and review processes. Coupled with CPIC, organizations will need to assess whether components and services are already in existence before starting development programs. Finally, business process registries will emerge and governance procedures will guide the posting, use and attributes of consumable components and services.

IT Tasking – traditional IT tasking is system centric, generally awarding Systems Development, Systems Redesign, and Systems Maintenance tasks to contractors. This is a fundamental driver for the "stove-pipe" architectures observed today throughout federal government architectures. To enable SCBA, agencies will have to transform their IT tasking from a system-centric to a component-centric model. To fully realize the enterprise-wide potential of SCBA, organizations will need to establish teams that are responsible for providing services to multiple applications. This component-based tasking model will require agencies recognize the value of separating traditional systems tasking into at least two types: one for establishing "Component Service Provider" teams and another for establishing "Application Assembly teams." Providers would then focus on consolidating data and existing systems into enterprise components that offer all the services required by the Application Assembly teams as they work to automate business processes.

Cyber Security – security processes focus on ensuring the validity, timeliness, and distribution of information, and on authenticating and authorizing users. SCBA complicates these tasks by distributing information over a greater number of systems, and potentially exposing information to an inter-agency user population. Security processes, policies, and

infrastructure will need to support these changes. Security techniques such as "chains-of-authority," distributed user-authentication systems, shared credentials, and information-use policies can all be used to ensure security in service-oriented systems.

By combining strategic, policy, and organizational changes with architectures that are designed for reuse, the vision of agile and citizen-centric business processes can be realized. SCBA is such an architecture.

4 Enabling Reuse of Services and Components

Achieving successful reuse involves more than simply architecting systems a certain way. It involves integrating reuse into all aspects of how an enterprise operates. Successful reuse programs share many common characteristics, and these are incorporated into SCBA. These characteristics are grouped into five categories:

- **culture** that actively encourages and rewards reuse,
- **designs** for processes and systems that assume they will be reused,
- **tools** that enable the discovery and tracking of reusable assets,
- **infrastructure** that supports sharing of reusable assets, and
- **processes and policies** to ensure reusable assets are harvested, shared, and reused.

Each of these are further expanded and explained in the following sections, as well as issues commonly encountered when executing reuse strategies.

4.1 Processes and Policies for Reuse

All mature organizations have a documented system or solution development process, commonly referred to as a "System Development Lifecycle (SDLC)" that governs how systems are created and modified. Whatever form of SDLC or software development methodology that exists within an organization (waterfall, Rational Unified Process (RUP), Agile, Extreme, other) reuse should be examined during each phase. During proposal phases, the entire process or system should be compared against other existing processes and solutions to determine what Service Components might be reused in the solution. During the architecture and design phases, any Service Components being newly developed should be reviewed to ensure they take potential reuse into account. During implementation, any modifications planned to a reused Service Component should be done in such a way as to make them easy to republish to all other users of the Service Component. During maintenance phases, any reused Service Components should be reviewed to determine if updated versions of those Service Components have been published, and consideration should be given to incorporating those updates.

Example – Project Reuse Quotient

It is a best practice to collect metrics to track and control process performance. In a similar manner, the degree to which a project reuses components may be measured and controlled by a "reuse metric." This simple metric is defined in Figure 6. Values close to zero represent low reuse; values equal to one or above represent high reuse. Reuse quotients below a threshold should trigger additional project reviews. The metrics serves the dual goals of tracking the organization's progress towards greater reuse, and driving desired behaviors.

Figure 6 - Project Reuse Quotient

$$\text{reuse quotient} = \frac{\left(\begin{array}{l}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused FROM other}\\ \text{systems or processes}\end{array}\right) + \left(\begin{array}{l}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused IN other}\\ \text{systems or processes}\end{array}\right)}{\left(\text{total number of "service-components" in project}\right)}$$

This metric purposefully neglects many factors that can affect reuse levels, such as uniqueness of requirements, mission criticality, and the scope of services. Accounting for these factors would lead to an overly complex metric, which would detract from this measure being easy-to-compute, track, and use. Projects that have low reuse levels for good reasons should simply note those reasons when submitting their reuse quotient. Not all projects should have a high reuse quotient – the specific program and mission goals involved must be taken into account when setting goals and targets. See Appendix C for several examples of how to compute reuse quotients.

During all phases, decision makers should challenge any decision to develop new processes or systems. New systems and processes will always need to be developed, and the decision to do so is often well justified. However, virtually all processes and systems have elements that have been developed before.

Finally, a process for tracking and supporting other organizations that are utilizing Service Components should be put into place.

4.2 Design for Reuse

Industry research has documented that designing an asset so that it can be easily reused usually adds up to 50%[†] to its overall cost. Research also shows that successful reuse can save a project 25-30% in its development costs[‡]. Given these economics, even if a Service Component is reused only a few times, the return on this investment is realized. Designing for wide-scale reuse involves five basic principles:

- **generalizing functionality for broader applicability,**
- **creating well defined interfaces,**
- **loose-coupling,**
- **ensuring well documented functionality,** and
- **using cross-platform technologies** (if developing a system).

For a system or process to be reused, well-defined interfaces are critical. All systems and processes have various "entry" and "exit" points. A well-defined interface encapsulates one set of these entry and exit points into a format that can be easily understood and accessed by a third party. This involves making sure that the data, steps needed to access some functionality are documented, and a mechanism for initiating an action is exposed in an accessible way.

The GovBenefits Rules Engine – An Example of Design for Reuse

While creating the "GovBenefits" benefits search system in 2003, the Department of Labor had specific requirements to ensure that the "rules" that determine whether a user is a candidate for a particular benefit be easily changeable. The development team responded to the requirement by creating a generic "rules engine" that allowed a wide variety of business rules to be expressed. The development team went further, though, by designing the rules engine as a reusable component. This investment paid off when, in 2004, the Department of Energy created the "GovLoans" system, which reused this functionality. This reuse saved the Department and its partner agencies an estimated 50% on the overall development costs of the system.

More information on GovBenefits and GovLoans can be found at http://www.govbenefits.gov and http://www.govloans.gov.

Closely connected with this, a consumer can only realistically reuse a system or process if what the system or process does is well documented. SCBA calls for all Service Components to provide written documentation that describes what functionality the Service Component offers, a basic summary of how it accomplishes that functionality (process flow, algorithm, etc), and exposure of all internal details (such as source-code – where not feasible for legal or security reasons). This documentation should be publicly available via a web site, kept under version control, and should be easily accessible from the same system that publishes the existence of Service Component.

[†] Measuring Software Reuse, Jeffrey S. Poulin, 1996

[‡] "Asset Based Software Engineering" Charles M. Stack, Flashline, Inc.

If the Service Component being designed is an IT system, its interfaces should be packaged in way that is accessible via multiple technical architectures. A proliferation of technical architectures exists today, and will probably exist for the near to distant future. However, a key recent development is the penetration and acceptance of technologies, specifically web services, which allow functionality to be easily exposed and consumed. While technologies such as "Remote Procedure Call (RPC)" and "Common Object Request Broker Architecture (CORBA)" have existed for many years, the availability of web services on virtually all computing platforms, combined with many tools facilitating there use, finally make practical the concept of reuse across architectures.

Finally, Service Components should also be designed in such a way as to allow for easy re-configuration of its behavior to suit specific users. For IT systems, this usually entails allowing parameters that will reasonably vary from consumer to consumer (e.g., interest rates, tax levels, security permissions) to be changed easily (e.g., through API calls).

4.3 Tools for Reuse – Registries, Repositories, and the SRM

To reuse something, a consumer must know that it exists. Registries are databases that allow a potential Service Component consumer to search known Service Components and review the functionality that they offer. Repositories are alternate tools that offer the search capabilities of registries, but go further by actually containing copies of the Service Components themselves. Due to the difficulty in designing a repository that can account for all the various forms a Service Component can take, registries are a best practice for most industry reuse efforts. Core.gov is the Federal Service Component registry, and is where all SCBA Service Components may be registered.

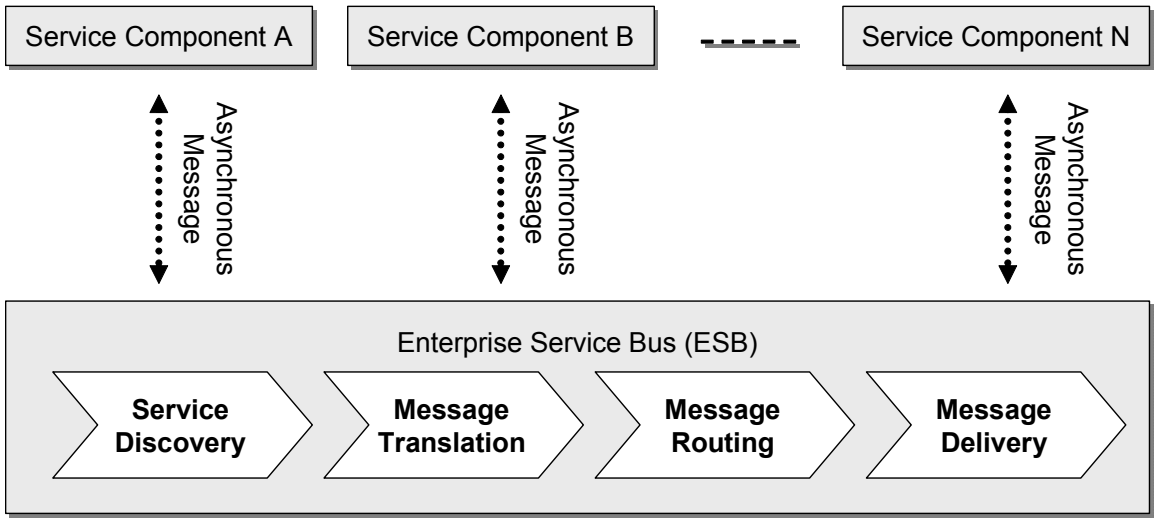
The SRM is another important tool for helping to discover components. It is a business-function independent framework for classifying Service Components across the federal government. It divides all Service Components into commonly defined "service domain," "service type," and "component" categories. These categories indicate the capabilities that the Service Component offers. All federal agencies are required to align their EAs to the SRM, making it a powerful tool for discovering reuse opportunities. The EAs of all Federal Agencies can be searched via FEAMS, providing a mechanism for finding potential consumers or sources of Service Components and search mechanisms exist within Core.Gov to find SRM mapped Service Components that have been registered and approved.

4.4 A Reuse Infrastructure – The Enterprise Service Bus – Example

Fundamentally, a SCBA is a collection of loosely coupled Service Components collaborating to accomplish business objectives. "Loosely coupled" means that the Service Components can be individually modified without affecting the functionality of the services that they expose to other Service Components. This is a critical characteristic core to accomplishing the vision of agile business processes. It allows elements not only to be reused but also to evolve over time without breaking.

The Service Components collaborate by exchanging information primarily via asynchronous messages. Asynchronous messaging is a mature and highly scalable technology that exists on almost all technology platforms. Where a SCBA is different from traditional Message Orient Middleware (MOM) is the existence of an Enterprise Service Bus, or "ESB". An ESB is similar to a MOM broker, but takes on the additional responsibility of translating, routing, and delivering messages from service to service. These additional steps place these responsibilities outside the scope of the Service Components, and further the goal of keeping all Service Components loosely coupled. The role of the ESB in SCBA is shown in Figure 7.

Figure 7 - Enterprise Service Bus (ESB)



While an ESB may seem relevant only to technical components – it is also critical to the reuse of processes. All Service Components, whether technical or business, need to establish well-defined interfaces. The ESB is the mechanism by which these interfaces are exposed.

4.5 A Culture of Reuse

The final, and most important, aspect of all successful reuse programs is a culture that actively encourages and rewards reuse. Processes, tools, and design create the basics of an environment that allows reuse. However, reuse efforts will only succeed if an organization's culture is transformed such that "reuse" is the first thought individuals have when faced with a problem. Beyond setting up the basic processes and tools, a reuse culture has two main requirements:

- senior leadership support, and
- a rewards system.

Senior leaders must champion reuse by expecting that assets be reused, recognizing projects and individuals that successfully reuse assets or publish them, and by making reuse a priority. Rewarding individuals and projects who successfully publish Service Components or have high reuse rates helps to accelerate this cultural change. A common commercial technique for accomplishing this is to set up a monetary reward for Service Component producers that pays a bonus to the producer every time their Service Component is successfully reused. A parallel reward for projects with a high reuse quotient is another common technique. While a monetary award may not be a feasible option in a government setting, a rewards system of some type can be created. Finally, all of the elements of the reuse program must be communicated to all stakeholders through a communications plan.

4.6 Governance and Responsibility Issues

Individuals have many common questions when considering reusing an asset from another organization. These issues can be addressed by documenting the commitments that the Service Component producer has made. Creating documents, called service level agreements (SLAs), should be a core part of the component publication process.

Common questions, and how they are addressed by SLAs, are indicated in Table 1.

Table 1 - Common Questions Around Reuse

REUSE QUESTIONS	RESPONSES
"How do I know that a Service Component will continue to be supported?"	All Service Components should have an SLA that specifically documents the support commitments that a Service Component provider has agreed to. Additionally, several best practices should be used in these agreements to help ensure ongoing support: • cost-sharing arrangements between the consuming and providing organizations help to ensure resources are allocated on an ongoing basis, • provisions to grant "ownership" of the Service Component to the consumers if the sponsor is unable to meet the commitments indicated in the SLA, • if the Service Component is an IT system, publication of the source-code for the Service Component to the consumer, so that (failing other options) the consuming organization could support the Service Component itself.
"What do I do if I need modifications?"	This is another area that should be covered by the Service Component's SLA. Two general models exist for handling modifications: • **Central Control**: under this model the Service Component provider makes all modifications, and the SLA documents the general schedule and procedure for requesting modification. A best practice is to have special procedures for handling emergency changes. This model provides clear lines of authority and easy to follow procedures for modifications, but can be slow. • **Open Source**: under this model the Service Component provider or any consumer can make a modification to the Service Component, and then have that update appear in the next tested release of the Service Component. This model is less structured, but can be much more rapid and responsive to the needs of the consumers.

REUSE QUESTIONS	RESPONSES
"Am I allowed to use this in my organization?"	This question is partially answered by the Service Component's SLA, and partially answered by the potential consumer's organization. The SLA should document any restrictions that exist on reusing this component in other organizations. The consumer's organization should clearly document in its reuse policy any restrictions that exist within the organization, and any procedures necessary to getting approval to reuse something.
"Are there any fees associated with using this?"	The SLA for the Service Component should clearly document any fees associated with use of the Service Component. Additionally, if the consumer plans to use the Service Component on a large scale, discussions should be held between the consumer and provider to determine if any impacts to the provider's cost structure will occur.
"This Service Component contains COTS software – does its license agreement allow me to use it?"	The SLA for the Service Component should clearly define any restrictions stemming from commercial license agreements. When a Service Component is offered for general use consideration should be given to renegotiating license agreements to facilitate cross-department and cross-agency use.
"Can't I just write this myself? It'll be quicker and less problematic, and besides, my requirements are unique."	Industry research has proven the value of reusing quality assets over re-creating them. The value derived from this reuse is proportional to the size and scope of what is being reused. In addition, although all organizations have different requirements, these differences are usually small enough that either adapting the requirements to what is available, or making small modifications to the Service Component is a superior solution to developing something new. In addition, if the Service Component is modified the advantages of these improvements can be potentially extended to all users.
"Who is responsible for funding the service component?"	The SLA for the service component should clearly lay out the responsibilities for funding maintenance and support costs. Larger scale reuse is best supported by cost-sharing schemes, and smaller scale by allocations.

5 Implementation Strategies

Many possible strategies exist for implementing an SCBA – the following sections describe some of the most common approaches. Implementation strategies can be grouped according to their starting point: top-down, bottom-up, and middle-out. These three approaches are not mutually exclusive and each will provide different benefits to different SCBA stakeholders.

5.1 Top-Down

The "**Top-Down**" implementation strategy involves approaching SCBA from the master blueprint of the organization – the Enterprise Architecture. This approach provides the greatest long-term benefits because it takes a holistic view of the business processes and services required by the organization. By developing the EA from a services perspective or analyzing an existing EA at the BRM, SRM, and DRM levels to determine potential areas for reuse and interoperability, the organization can begin to identify Service Components that can be provided to meet a variety of requirements.

All elements that are in supporting roles and not reused should be the organization's top targets for reuse. Each of these elements should be intensively reviewed to determine if equivalent elements are offered by other organizations. Elements that are core to the mission of the organization should likewise be reviewed to determine if they are potential candidates for reuse by other organizations, or if better alternatives exist within other organizations. Based on these reviews, the target architecture of the organization should be modified to replace appropriate elements with reused Service Components from other organizations. Additionally, plans should be made to modify elements that can potentially be reused into full reusable Service Components. Over time, reusable elements identified in the EA should be consolidated into Service Components.

5.2 Bottom-Up

The "**Bottom-Up**" strategy involves creating a collection of reusable Service Components that can be leveraged across the organization. In some cases, this will involve identifying services offered by external organizations; in other cases, it will involve creating Service Components from scratch or "fronting" existing functionality with an interface to create a Service Component. The result of this approach is to build a repository of reusable Service Components available to solution developers. This approach has the advantage of quick execution (creation of common Service Components can begin immediately), although without a well thought-out plan, the solution development projects may not be able to exploit these services.

5.3 Middle-Out

The "**Middle-Out**" approach is systems-oriented. It involves adding Service Componentization and examples such as the Enterprise Service Bus (ESB) integration tasks to work ongoing or already scheduled for systems or processes. This "organic" approach allows all elements in an organization to be modified as per the normal course of maintenance and enhancements, and avoids special projects or capital investments (thus keeping incremental costs low). The bottom-up approach can also be directed at reusable functionality found in legacy systems that are still relevant to the organization, but which use technical architectures different from the organization's target enterprise architecture. The technical gap between these systems and the target architecture can be bridged by the cross-platform advantages of an SCBA. The middle-out approach has the additional

advantage of gradually building organizational support for the architecture by allowing small pilot projects to pioneer the involved techniques and technologies.

5.4 Choosing a Strategy

It is impractical for all systems and processes in an enterprise to be migrated to an SCBA simultaneously – and so it is recommended that organizations move them in "waves." Given the natural alignment between SCBA and IT systems, it is best to start this migration with a selected group of IT systems that are already undergoing modifications, or for which there is a significant reuse demand. Alternately, "deep dives" can be done for high-priority business lines that offer the most chance for benefit and optimization.

These approaches, while not mutually exclusive, are targeted at helping agencies implement a SCBA. Each approach provides agencies with a "starting point" that should be balanced with the maturity of technologies and processes within the agency. For instance, agencies who embrace SCBA from a strategic perspective will likely choose a "Top-Down" approach and bundle it into their EA and CPIC processes. Others, who elect to create ad-hoc components and services for a specific business process, will follow the "Bottom-Up" approach. Those who are interested in blending each of these approaches will likely choose a "Middle-Out" approach. The "right" approach to choose is dependant on the conditions within the agency.

6 Getting Started

Achieving the objectives of the service-oriented architecture approach is more of a journey than a destination. Because of the pervasive changes required in thinking about software applications and in the solution life cycle, it is important to set achievable goals for the short, medium, and long term. In the early stages, the foundation must be laid and momentum established that would carry over into the subsequent stages. This section addresses the roadmap for service-based architectures and recommends some initial steps that can assist in getting started[§].

6.1 Implementation Framework

Several areas or work streams must be considered and kept in balance to make effective progress in implementing an SCBA:

- Planning and management,
- Architecture (includes security),
- Infrastructure,
- Process, and
- Projects.

Planning and management - deals with determining the overall strategy for SCBA, establishing the policies for coordinating the multiple activities and organizations involved, creating the funding mechanisms for cross-program services, and implementing the monitoring and reporting schemes to enable the services environment.

The **architecture stream** is concerned with developing the overall, layered service model, defining the security framework, adopting architectural and design patterns, establishing the set of semantics, and implementing the governance structure and guidelines.

The **infrastructure stream** is responsible for the technical platform that the service-based applications rest on, including, the hosting platform, middleware for interoperability and translation, workflow and business process management, development environment and tools, and the asset management repositories and directories.

The **process stream** deals with the reuse initiative and the revisions necessary in the solutions development life cycle (SDLC) to enable SCBA. The SDLC must be modified to incorporate the "twin-track" development paradigm (recognizing that provisioning services and assembling services into application solutions are two distinct process paths). It also addresses business integration (services as business products), the certification and publishing of services, the security and trust process, and the development of acquisition guidelines and templates (language to incorporate into procurements that specifies a services implantation).

The **projects stream** is responsible for the overall project master plan and the development of service based project plan templates, project-scoping guidelines, guidelines for service acquisition/provisioning decisions, and for the overall successful execution of the project. The master project plan should describe the series of sub-projects that will help transition to SCBA. Standard project types include service harvesting from legacy systems, provisioning technical components and enterprise level services, and assembling services into capabilities to meet business requirements.

---

§ This section borrows from the CBDi Forum report: "Web Services Roadmap: Guiding the Transition to Web Service and SOA," 2003.

Table 2 presents a recommended set of activities for agencies to consider as they begin to implement service and component architectures. All of these activities are important, but some may be omitted based on the implementation approach selected (e.g., a "bottom-up" approach may omit several activities in the "Planning and Management" stream).

Table 2 - "Getting Started" Strategies for SOA Areas

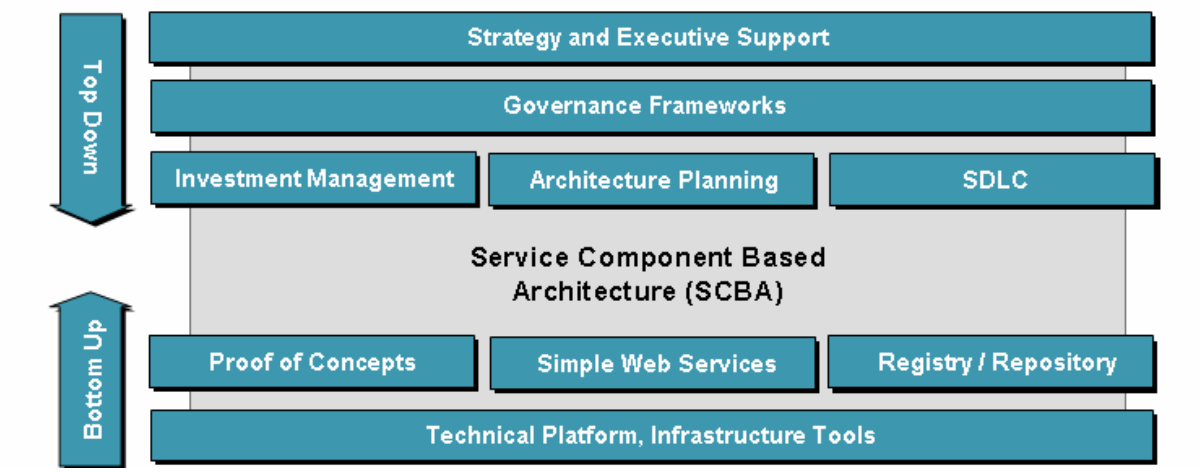
SOA STREAM	RECOMMENDED STEPS FOR GETTING STARTED
Planning and Management	• Establish overarching strategy to implement SCBA (based on federated management) • Define enterprise policies to guide programs/projects to provision and consume services and assemble solutions • Fund development of initial common services • Establish governance activities, roles and responsibilities • Establish metrics to measure the performance of all other streams • Define roles and responsibilities and training requirements • Include SCBA in target EA, and lay out SCBA project in the EA transition strategy and sequencing plan
Architecture	• Establish layered services model with phasing of common services • Define security framework • Establish initial governance structure
Infrastructure	• Implement hosting platform and middleware services • Establish repository/directory for asset management
Process	• Update the SDLC to reflect services paradigm • Establish reuse program (including incentives & rewards) • Establish service certification and publishing process • Develop acquisition guidelines and templates
Projects	• Define series of 100-day projects to provision services and deliver solutions • Develop template project plans and scoping guidelines

As discussed above, what is important is to recognize that several areas must be evolved simultaneously – a managed approach will accelerate the achievement of SCBA objectives and produce **measurable** results.

6.2 Specific Steps for Getting Started

Several specific possible approaches exist for getting started with SCBA. These are illustrated in, Figure 8 and described in the following sections.

Figure 8 – SCBA "Getting Starting" Approaches



6.2.1 Establish Basic Environment

Implementation of a SCBA is dependant on first ensuring that the organization has the basic environment for SCBA described in section 4. This includes:

- senior leadership support,
- common, documented SDLC is used,
- integrating reuse reviews into the SDLC,
- tracking the reuse quotients of all projects,
- reuse policies and rewards program, and
- Service Component SLA process.

These are best accomplished by appointment of a "SCBA champion." This individual should report directly to the CIO, CTO or Chief Architect of the organization, and be charged with establishing this environment in a given period of time.

6.2.2 Establish Enterprise Service Bus (ESB) – Example

After establishment of the environment, the process of modifying the architecture of existing systems into an SCBA can begin. One approach is to establish an ESB, and to create organization-wide standards requiring use of the ESB for new system development. When using an ESB approach, a single ESB needs to be established by the organization to realize an SCBA.

6.2.3 Migrate Systems to SCBA

The final goal after establishment of the environment is increasing the reuse quotients (see page 1-12) of all processes and systems, and their integration into the ESB. Increasing reuse quotients largely involves repacking potentially reusable systems and processes as Service Components, and in taking advantage of existing Service Components.

Repackaging **system functionality** as Service Components is usually a straightforward IT exercise, and involves the following steps:

- identifying what functionality is potentially of use to others,
- making sure that functionality is exposed through some interface,
- creating documentation describing the Service Component's functionality, and
- registering the Service Component in Core.gov.

Integration into the ESB and reuse can be greatly facilitated by exposing the interface to the Service Component using web services technology. Web services technologies are a collection of technologies that allow services to expose interfaces in ways that are discoverable, network accessible, and cross-platform. By exposing interfaces in this way, they will be usable in the widest variety of environments.

Repackaging **processes** as Service Components follows a similar procedure, but the exposed interface may or may not be technology based. In fact, the processes may be expressed using modeling tools.

Finally, integrate individual Service Components into the organization's ESB. In most cases, **system** interactions that had taken place via other technologies can be directly translated to the ESB format. **Process** interactions without technology interfaces can also be moved to the ESB by creating small "adapter" systems that allow those involved in the process to exchange information using the ESB.

6.3 Case Studies

Examples often provide the best education on how to initiate new programs. Several examples of successful reuse are given in Appendix D: Case Studies.

7 Conclusion

SCBA is a powerful architecture that combines the inter-organizational reuse of CBA, the cross-organization reuse of the FEA, and the agility of SOA. Implementation of SCBA is enabled by recent technological and architectural advancements that are rapidly gaining industry acceptance, and are a practical approach to achieving the dual visions of agile business processes and citizen centricity.

Government leaders should use the resources and guidance provided by the CIO Council, FEA, and other government-wide efforts, as well as their own agency resources, to establish service component reuse programs in their agencies. Service and component reuse reduces costs and increases service quality when implemented effectively. Service orientation takes reuse further by enabling business processes to be changed rapidly to adapt to changing needs.

Further chapters in SCBA will provide further details on how to establish such programs (see "Appendix B: Chapter Guide" for a list of these chapters).

Appendix A: Glossary

Term	Source	Definition
Abstraction	IEEE, 1983	A view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information.
Application Programmable Interface (API)	Webopedia	A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.
Architecture	SCBA v2	Representation of the structure of a system that describes the constituents of the system and how they interact with each other.
Architecture, Application	SCBA v2	Representation of an application and its parts, their inter-relationships and functions.
Architecture, Component	SCBA v2	Internal structure of a component described in terms of partitioning and relationships between individual internal units.
Certification		A formal process for making certain that an individual is qualified in terms of particular knowledge or skills, or that and IT system or business process meets certain criteria. Within the context of the FEA this refers to process buy which a system or business process is identified as an SCBA Service Component and listed in Core.gov.
Component	SCBA v2	Independently deployable unit of software that exposes its functionality through a set of services accessed via well-defined interfaces. A component is based on a component standard, is described by a specification, and has an implementation. Components can be assembled to create applications or larger-grained components.
Component Based Architecture (CBA)	CAF Glossary	An architecture process that enables the design of enterprise solutions using pre-manufactured components. The focus of the architecture may be a specific project or the entire enterprise. This architecture provides a plan of what needs to be built and an overview of what has been built already.
Component Based Development (CBD)		Approach to software development that consists of producing or acquiring components for assembly into applications.
Component, Business	IAC Succeeding, CAF Glossary	Component that offers business related services – applying business rules and accessing business data.
Component, COTS		A component supplied by a commercial vendor. See "COTS".

Term	Source	Definition
Component, Enterprise	IAC Succeeding	A large-grain business component. Typically consume smaller grained components. Examples include Customer Management, Case Tracking, etc.
Component, Infrastructure	SCBA v2, CAF Glossary	A technical component that provides application functionality not related to traditional business functionality (finance, accounting, human resources, etc.), such as error/message handling, audit trails, or security.
Component, Notional	SCBA v2, CAF Glossary	Set of services packaged into a component, derived from requirements definition. A "desired" component, prior to implementation.
Component, SRM	Service Component Reference Model, Version 1.0	A self-contained business process or service with predetermined functionality that may be exposed through a business or technology interface.
Component, Technical	SCBA v2	Independently deployable unit of software that exposes its functionality through a set of automated services accessed via well-defined interfaces. A component is based on a component standard, is described by a specification, and has an implementation. Components can be assembled to create applications or larger-grained components.
Consumption, Service		The process of interfacing with an utilizing the functionality of, and or providing functionality to, another Service Component.
COTS	Whatis.com	COTS (commercial off-the-shelf) describes ready-made products that can easily be obtained. The term is sometimes used in military procurement specifications.
Coupling		Coupling is a measure of the level of interdependency between two components. "Loose Coupling" (low interdependence) is good, as it maximizes system flexibility. "Tight coupling" (high interdependence) is bad, as it restricts system flexibility.
Design by Contract	Meyer, Bertrand (1997), Object Oriented Software Construction, Prentice Hall, Englewood Cliffs, NJ, ISBN 0136291554	Design by Contract views the relationship between a class and its clients as a formal agreement, expressing each party's rights and obligations. This precise and largely immutable definition of every module's claims and responsibilities is seen as vital to developing large software systems.

Term	Source	Definition
Design Pattern		See "Pattern"
Directory		A type of database that stores information in a hierarchical format.
Encapsulation	SCBA v2	Hiding implementation details within a component so that an implementation is not dependent on those details.
Enterprise Service Bus (ESB)	Bitpipe.com	An enterprise integration architecture that allows incremental integration driven by business requirements, not technology limitations.
Enterprise Architecture	CAF Glossary	(A) means—''(i) a strategic information asset base, which defines the mission; ''(ii) the information necessary to perform the mission; ''(iii) the technologies necessary to perform the mission; and ''(iv) the transitional processes for implementing new technologies in response to changing mission needs; and ''(B) includes—''(i) a baseline architecture; ''(ii) a target architecture; and ''(iii) a sequencing plan;
Extensibility	SCBA v2	Ability to extend the capability of a component so that it handles additional needs of a particular implementation.
Factoring		The process of dividing a IT solution down into the fundamental Service Components that will comprise that solution.
FEA		See "Federal Enterprise Architecture"
Federal Enterprise Architecture	www.egov.gov, FEA PMO Action Plan	The Federal Enterprise Architecture is an Office of Management and Budget initiative to comply with the Clinger-Cohen Act and provide a common methodology for information technology acquisition in the U. S. federal government. It is designed to ease sharing of information and resources across federal agencies, reduce costs, and improve citizen services. The FEA consists of a set of interrelated reference models designed to facilitate cross-agency analysis and the identification of duplicative investments, gaps, and opportunities for collaboration within and across agencies. These include the Performance Reference Model, the Business Reference Model, the Service Component Reference Model, the Data Reference Model, and the Technical Reference Model.
Framework	CAF Glossary	A logical structure for classifying and organizing complex information.

Term	Source	Definition
Gap-Fit Analysis	SCBA, CAF Glossary	1) Examination of components within the context of requirements and to make a determination as to the suitability of the component. 2) The difference between projected outcomes and desired outcomes
Granularity		The size of the service or component under consideration. The term generally refers to the level of detail or abstraction of the service.
Harvesting		(1) The process of evaluating and organizations businesses processes and IT assets in an effort to discover Service Components (2) The process of repacking of useful business functionality as a Service Component
Intellectual Property	SCBA v2	A product of the intellect that has commercial value, including copy-righted property such as literary or artistic works, and ideational property, such as patents, appellations of origin, business methods, and industrial processes.
Interface, Component or Service	SCBA v2	Mechanism by which a component describes what it does and provides access to its services. This is important because it represents the "contract" between the supplier of services and the consumer of the services.
Legacy System	CAF Glossary v0	An automated system built with older technology that may be unstructured, lacking in modularity, documentation and even source code.
Model Driven Architecture (MDA)	OMG MDA Guide 1.0.1	An approach to IT system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform.
Loose Coupling	Skyway Software	Loose coupling is a key attribute of SOA solutions, in that it means there are minimal dependencies among services and this allows the quick assembly of different business solutions from different combinations of business services from a variety of systems.
Pattern	Whatis.com	In software development, a pattern (or *design pattern*) is a written document that describes a general solution to a design problem that recurs repeatedly in many projects. Software designers adapt the pattern solution to their specific project. Patterns use a formal approach to describing a design problem, its proposed solution, and any other factors that might affect the problem or the solution. A successful pattern should have established itself as leading to a good solution in three previous projects or situations.

Term	Source	Definition
Pattern, e-Business		A pattern that focuses on an e-business problem.
Post Conditions	Meyer, Bertrand (1997), Object-Oriented Software Construction, Prentice Hall, Englewood Cliffs, NJ, ISBN 0136291554	A post condition states the properties that the routine guarantees when it returns. A post condition guarantees that the routine will yield a state satisfying certain properties, assuming it has been called with the precondition satisfied. The post condition puts onus on the class: it specifies the conditions that must be ensured by the routine on return. It is a benefit for the client and an obligation for the supplier.
Pre Condition	Meyer, Bertrand (1997), Object-Oriented Software Construction, Prentice Hall, Englewood Cliffs, NJ, ISBN 0136291554	A precondition states the properties that must hold whenever the routine is called. A precondition applies to all calls of the routine, both from within the class and from clients. A correct system will never execute a call in a state that does not satisfy the precondition of the called routine. The precondition places onus on the client: it defines the conditions where a call is legitimate. It is an obligation for the client and a benefit for the supplier.
Provisioning	Services Provisioning Markup Language Specification	The automation of all the steps required to manage (setup, amend, and revoke) user or system access entitlements or data relative to electronically published services.
Registry		A database providing information describing and categorizing objects, but which does not contain the objects themselves. Registries usually provide information as to how to access the object they describe.
Repository		A storage mechanism; typically a storage and retrieval mechanism for components and service information.
Repository, Component	CAF Glossary	Application designed to store component specifications and implementations. Provides facilities to efficiently search for and retrieve components for evaluation against desired component specifications.
Repository, Architecture	CAF Glossary (TEAF)	An information system used to store and access architectural information, relationships among the information elements, and work products
Reuse	SCBA v2	Any use of a preexisting software artifact (component, specification, etc). in a context different from that in which it was created.

Term	Source	Definition
SCBA		See "Service Component Based Architecture"
SDLC		See "System Development Lifecyle."
Service	SCBA v2, CAF Glossary	Discrete unit of functionality that can be requested (provided a set of preconditions is met), performs one or more operations (typically applying business rules and accessing a database), and returns a set of results to the requester. Completion of a service always leaves business and data integrity intact.
Service Component	SCBA v2	A self-contained business process or service with predetermined and well-defined functionality that may be exposed through a well defined and documented business or technology interface. Well-designed Service Components are "loosely coupled" and collaborate primarily by exchanging messages.
Service Component Based Architecture		Services and Components Based Architecture (SCBA) leverages the Federal Enterprise Architecture (FEA) and builds upon the concepts, principles, and benefits of Service Oriented Architecture (SOA). SCBA represents a practical, results-oriented, approach to modernizing enterprises. It is intended to help organizations reduce long-term costs, improve quality of service, improve information sharing, and help achieve a vision of flexible business processes supported by customer-focused applications, which can be altered in a matter of days instead of months. SCBA builds upon SOA principles in three ways: • it is tightly integrated with the Federal Enterprise Architecture, • it provides a description of what the architecture is (clarifying the varying descriptions that exist), and • it identifies the organizational, cultural, and process elements, as well as technological elements, that need to exist for these architectures to be successful. The most important aspect of SCBA is its focus on reuse of services and components – better referred to as Service Components.

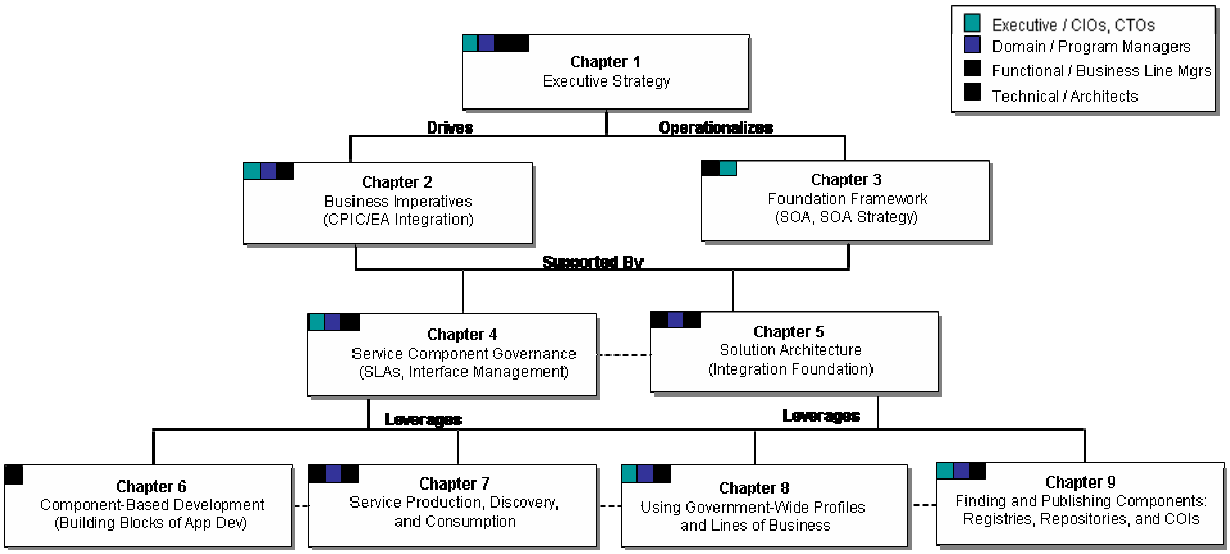
Term	Source	Definition
Service Level Agreement (SLA)	SCBA v2	A contract or memorandum of agreement between a service provider and a customer that specifies, usually in measurable terms, what services the service provider will furnish. Information technology departments in major enterprises have adopted the idea of writing a service level agreement so that services for their customers (users in other departments within the enterprise) can be measured, justified, and perhaps compared with those of external (sourcing) service providers.
Service Oriented Architecture (SOA)	SCBA v2 modified, EA Assessment Framework 1.5 CBDiForum Essential Guide	1) Architecture that describes an entity (e.g., application or enterprise) as a set of interdependent services. SOA provides for reuse of existing services and the rapid deployment of new business capabilities based on exploiting existing assets. 2) Representation of a system where the functionality is provided as a set of services called by other parts of the system 3) Policies, practices and frameworks that enable application functionality to be provided and requested as sets of services published at a granularity relevant to the service Requestor, which are abstracted away from the implementation using a single, standards based form of interface
SOAP	CAF Glossary	Simple Object Access Protocol - A World Wide Web Consortium (W3C) specification that facilitates the interoperability between a broad mixture of programs and platforms.
Solution Assembly	SCBA v2	Process of implementing a solution by assembling the necessary services into a complete solution. This process often involves additional "glue" code to integrate the assembled components.
System Development Lifecycle (SDLC)	Wikipedia	System Development Life Cycle, or SDLC, is the process used by a systems analyst to develop an information system, including requirements, validation, training, and user ownership through investigation, analysis, design, implementation and maintenance. SDLC is also known as information systems development or application development. An SDLC should result in a high quality system that meets or exceeds customer expectations, within time and cost estimates, works effectively and efficiently in the current and planned Information Technology infrastructure, and is cheap to maintain and cost-effective to enhance. SDLC is a systems approach to problem solving and is made up of several phases, each comprised of multiple steps.

Term	Source	Definition
Test Harness	SCBA v2	Software that automates the software testing process to test software services or components as thoroughly as possible before using them on a real application.
UDDI	CAF Glossary	Universal Description, Discovery and Integration is a an online directory that gives businesses and organizations a uniform way to describe their services, discover other companies' services and understand the methods required to conduct business with a specific company.
Use Case	Jacobson92	A use case is a narrative document that describes the sequence of events of an actor (an external agent) using a system to complete a process.
Web Service	SCBA v2 modified	Specific method of implementing a service, using the Internet (XML, TCP/IP) as the transport mechanism and conforming to a specific set off standards (WSDL, SOAP, etc).. Can be internally provided or can be offered externally.
Wrapping		Isolating the code to create an independently deployable unit of software and creating an interface around legacy code that exposes functionality as services via interfaces that conform to a component specification.
WSDL	CAF Glossary	Web Services Description Language is a specification that is published to a UDDI directory. WSDL provides interface/implementation details of available Web services and UDDI Registrants. It leverages XML to describe data types, details, interface, location and protocols.
XML	CAF Glossary	Extensible Markup Language is a non-proprietary subset of SGML (Standard Generalized Markup Language). It is focused on data structure and uses tags to specify the content of the data elements in a document.

Appendix B: Chapter Guide

Given the differing focus areas of individuals in its target audience, SCBA has been organized into distinct chapters. Each chapter is specifically targeted at the needs and concerns of a sub-set of the overall audience. Figure 9 is a guide indicating which chapters are most relevant to which groups, and how these chapters interrelated. It is intended that individuals read the chapters pertinent to their needs without having to review the entire document.

Figure 9 - Chapter Guide



The following is a detailed description of each of the chapters in this document.

Chapter 1 – Executive Strategy

This chapter provides an overview of Services and Components Based Architectures, explains their advantages and origins, and describes a ready-to-implement strategy for their implementation. It specifically covers the roles of services and components in modern architectures, what strategic and policy changes need to be implemented to enable service oriented architectures, and a concrete implementation strategy. Setting up a reuse-focused organization is also covered, and includes details on processes and polices, design for reuse, focusing an organization's culture on reuse, and reuse governance and responsibilities.

Chapter 2 – Business Imperatives (SRM/CPIC/EA Integration)

This chapter will discuss the business and economic imperatives of integrating service and component reuse into existing government-wide policies and practices. It specifically addresses the SRM, Capital Planning and Investment Control (CPIC), Enterprise Architecture (EA) and their integration. This chapter will discuss the concepts surrounding the integration of CPIC and EA processes in driving a component-based architecture. This will include discussions on how to discover Service Components, how to identify re-usable assets, how to leverage existing investment control processes to enable reuse, and what new governance models will be needed. Last, this chapter will discuss how to use EA analysis to perform "what-if" and scenario-based modeling to fully assess the feasibility, viability, reliability and economic impact of reuse.

Chapter 3 – Foundational Framework (SOA, SOA Strategy)

This chapter will define Service-Oriented Architecture (SOA) and discuss the need for business managers to think of SOA relative to an overarching enterprise strategy for enabling the reuse of Service Components. It will focus on the importance of standards and specifications, such as XML, WSDL, UDDI, and SOAP, and also cover the following related topics:

- Industry SOA best practices on where to start, what policies and procedures need to be in place, what outcomes to consider, and what design factors to evaluate now and in the future.
- The Services Evolution Life-Cycle (SELC), consisting of a suite of activities around planning, publishing, consuming, discovering, and managing Service Components.
- How to calculate the costs and timeframes for developing services,
- New governance and compliance models needed to govern reuse policies and practices with an SOA focused organization.
- The concept of a Business Services Registry (BSR), processes for publication of Service Components into them, and criteria for Service Component classification (e.g., mission critical, non-mission critical, etc.).

Chapter 4 – Service Component Governance

This chapter will discuss the governance processes that must exist to allow Service Components to be effectively developed and managed, and how those processes map to organizational structures. The importance of interface-centric management will be described, and how to create supportive but unrestrictive SLAs. Multi-generational-service planning and decision-gate based program management will be reviewed as best practices, and the importance of proper service launch, maintenance, and sun-setting will be reviewed.

Chapter 5 – Solution Architecture

This chapter will discuss how to create a Solution Architecture that directly supports the realization of re-usable services and components across an enterprise. This chapter will introduce Solution Architecture concepts and principles, describes the goals, objectives and outcomes of a Solution Architecture, present best practices, case studies and lessons learned, and describe how agencies should bundle Solution Architectures into EA, CPIC and procurement processes.

Chapter 6 – Component-Based Development

This chapter will review the technical details of Component-Based Development (CBD). Specifically, it will briefly review the fundamental design concepts of interfaces and encapsulation, and the most popular technical frameworks for doing general component development (e.g., .com and JavaBeans) and enterprise component development (e.g., .NET and Java EE). It will focus heavily in two areas (1) component scope and interface design and (2) on how to expose Web Services based interfaces. The Web-Services section will provide guidance on how to ensure that these interfaces are fast, maintainable, secure, and fully cross-platform accessible. It will conclude with recommendations as to when not to use Web Services, and how to provide useful component documentation.

Chapter 7 – Service Production, Discovery and Consumption

This chapter will introduce the dual concepts of "Service Provider" and "Service Consumer" and cover the following key areas of Service Component production, discovery, and consumption:

- the roles, responsibilities and attributes that needs to be considered (e.g., security, access policies and descriptors) when enabling services for reuse,
- how to discover Service Components through public and private Universal Description, Discovery and Integration (UDDI) directories, component registries and repositories, and service registries,
- how to leverage and engage communities of interest in the production of services and components,
- how to effectively consume (or use) Service Components – including the necessary governance controls and Service-Level Agreements that will be needed to analyze the performance of reuse relative to the demands of the business, programs and missions.

Chapter 8 – Using Government-Wide Profiles and Lines of Business

This chapter will discuss how to use Government-Wide Profiles, when to consider the use of a profile, and how profiles can help to reduce costs and improve organizational performance. This chapter will provide examples such as the Records Management Profile and describe how missions, programs, and enterprises can leverage these profile services and components within their existing business processes (e.g., Capture Record Component, Record Archive Component, etc). The chapter will also discuss how to discover Service Components from LoB initiatives (e.g., Human Resources, Financial Management, and Grants Management) and SRM component lists. Finally, a set of guidelines for how to leverage LoB and SRM information will be provided.

Chapter 9 – Finding and Publishing Components: Registries, Repositories, and COIs

This chapter will discuss the concepts of component registries and repositories, highlight the differences between each, discuss their use and how to leverage them, and describe how to use them to publish Service Components. The importance of engaging communities of interest when developing and publishing a component and service is described, as well as techniques for facilitating these communities. How to join LoB communities of interest to further assist in developing cross-agency, shared-service business models will also be discussed. Finally, this chapter will present the life cycle of sharing a component, including how components are certified, discovered, and reused.

Appendices

In addition to the main text, two appendices are envisioned that would evolve as each chapter is produced. The first would be a master glossary of SOA terms, and the second a master list of SOA reference material.

Appendix C: Reuse Quotient Examples and Notes

Example 1 – Project both Using and Producing Service Components

An agency's web site reuses an existing search and a customer authentication component from other project. At the same time, the project creates a new geospatial management Service Component. When the system is completed, the only project using the new Service Component is the project itself. The project's reuse component would be equal to .66, computed as shown in Figure 10. The "total number of Service Components in project reused IN other systems or processes" is equal to zero because, although a new Service Component has been created, it has not actually been reused yet.

Figure 10 - Project Reuse Quotient at Project Delivery

$$\text{reuse quotient} = \frac{\begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused FROM other}\\ \text{systems or processes}\end{pmatrix} + \begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused IN other}\\ \text{systems or processes}\end{pmatrix}}{(\text{total number of "service-components" in project})} = \frac{(2)+(0)}{(3)} = .66$$

The project team then publishes the geospatial-management Service Component in a component registry. Over time, two other agencies discover the new Service Component and incorporate it into two of their projects. When this occurs, the reuse quotient for the project will increase, as shown in Figure 11.

Figure 11 - Project Reuse Quotient When New Component Reused

$$\text{reuse quotient} = \frac{\begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused FROM other}\\ \text{systems or processes}\end{pmatrix} + \begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused IN other}\\ \text{systems or processes}\end{pmatrix}}{(\text{total number of "service-components" in project})} = \frac{(2)+(2)}{(3)} = 1.33$$

A critical observation to take away from this example is that a project's reuse quotient can change over time.

Example 2 – Project Exists Solely as a Service Component

An agency decides to produce a new security-assessment tracking database, intended for reuse across multiple agencies. The Service Component is designed for integration into other agency's security systems, and not to actually function by itself (it offers no user interface). The Service Component does not reuse any other Service Components. When the Service Component is first created, its reuse quotient will be equal to zero, as computed in Figure 12.

Figure 12 - Project Reuse Quotient at New Service Component Creation

$$\text{reuse quotient} = \frac{\begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused FROM other}\\ \text{systems or processes}\end{pmatrix} + \begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused IN other}\\ \text{systems or processes}\end{pmatrix}}{(\text{total number of "service-components" in project})} = \frac{(0)+(0)}{(1)} = 0$$

Six other agencies decide to reuse this new Service Component. When they complete their integration work, the project's reuse quotient jumps to 6, as computed in Figure 13.

Figure 13 - Project Reuse Quotient after New Service Component Reused

$$\text{reuse quotient} = \frac{\begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused FROM other}\\ \text{systems or processes}\end{pmatrix} + \begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused IN other}\\ \text{systems or processes}\end{pmatrix}}{(\text{total number of "service-components" in project})} = \frac{(0)+(6)}{(1)} = 6$$

Example 3 – Project Repackages Functionality as a Service Component

A department within an agency has an internally successful business process for obtaining approval for budget requests, and decides that this process may be reusable by other departments. At the start, this business process is not well documented and offers no external interfaces. Because of this, the business process is not a Service Component, and its reuse quotient is undefined, as computed in Figure 14.

Figure 14 - Project Reuse Quotient Before Modification to bel Service component

$$\text{reuse quotient} = \frac{\begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused FROM other}\\ \text{systems or processes}\end{pmatrix} + \begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused IN other}\\ \text{systems or processes}\end{pmatrix}}{(\text{total number of "service-components" in project})} = \frac{(0)+(0)}{(0)} = \textit{undefined}$$

Over time, the business process is modified to become a Service Component. When this is completed, the business process's reuse quotient changes to zero, as shown in Figure 15.

Figure 15 - Project Reuse Quotient After Modification to bel Service component

$$\text{reuse quotient} = \frac{\begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused FROM other}\\ \text{systems or processes}\end{pmatrix} + \begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused IN other}\\ \text{systems or processes}\end{pmatrix}}{(\text{total number of "service-components" in project})} = \frac{(0)+(0)}{(1)} = 0$$

Finally, the Service Component is adopted by one other department. At this time, its reuse quotient changes to one, as shown in Figure 16.

Figure 16 - Project Reuse Quotient Project Reuse Quotient After Business Process Reused

$$\text{reuse quotient} = \frac{\begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused FROM other}\\ \text{systems or processes}\end{pmatrix} + \begin{pmatrix}\text{total number of "service-}\\ \text{components" in project}\\ \text{reused IN other}\\ \text{systems or processes}\end{pmatrix}}{(\text{total number of "service-components" in project})} = \frac{(0)+(1)}{(1)} = 1$$

Cautions on Use

It is critical to note that this metric does not necessarily show how much a project increases reuse; only how much reuse it involves. Compare two hypothetical projects competing for resources. Project A is a small but contentious modification to a service that is already reused in 10 different places. Project B is the first step towards making a legacy system

available as a service. Project A will have a high score and B a low score, but B will result in an increase in sharing, while A will only be an in-place improvement. Going for the high score is not necessarily the right thing to do. The quotient is a useful metric, but good management and common sense should be exercised by decision makers in all cases.

Appendix D: Case Studies

Case Study 1 – Authentication Service Component (ASC)*

The ASC is a government-wide authentication solution that includes technical and policy subcomponents to identify users on the internet. It includes policy sub-components that establish standard levels of risk for applications, standard levels of assurance for credentialing services, and standard technical specifications for system interactions. The ASC has internal governance components to manage, maintain, and ensure uniform application of these standards. The ASC also includes operational systems, including a portal and a Certificate Authority that issues certificates used for server-to-server authentication.

The ASC enables organizations to participate in a federation in which members can rely on each other's credentialing systems securely, enabling end user identity to be portable across Internet domains. This eliminates the need for every web-enabled application to establish its own identity management and credentialing system, resulting in enormous savings for the government and easier online access to government services for citizens. The Program Management Office (PMO) assigns relationship managers and technical subject matter experts to assist organizations implementing, or considering implementing, the ASC. The PMO also operates an interoperability lab that tests relevant Commercial off the Shelf (COTS) products to ensure interoperability within in the ASC.

Case Study 2 – Housing and Urban Development (HUD) Electronic Case Binder†

In 2005, the Department of Housing and Urban Development created two service components designed to help facilitate housing-related financial documentation. The components help authorized lenders endorse Federal Housing Authority mortgage loans for insurance without a pre-endorsement review by HUD. Before these service components were developed, lenders applying for insurance had to transmit insurance data and mail paper case binders to HUD's Homeownership Centers. With these new service components, companies handle the endorsement themselves and send only electronic case binders to HUD when requested. This new, re-designed process reduces processing time by one third and decreases direct insurance expenses by as much as 25 percent. This Web-based service is shared by HUD, the Veterans Affairs and Education departments. HUD has estimated that it has saved taxpayers more than $500 million in reduced loan losses.

Case Study 3 – Department of Labor, Business Rules Engine

When creating the GovBenefits.gov citizen online benefit search engine portal in 2003, the Department of Labor specified a requirement that the rules expressions that define whether a user is a possible candidate for a particular federal or state benefit program are easily modified or extended. The development team responded to the requirement by creating a generic rules engine that allowed for a wide variety of business rules to be expressed. The development team went further, though, by designing the rules engine as a reusable component. This investment paid off, when, in 2004, the Department of Energy created the GovLoans.gov portal, which reused this component in its entirety. This reuse saved the

---

* This text adapted from the June 2005 ASC application from registration in Core.gov

† Information from Government Computer News, September 27, 2005 "Service-oriented components advance transformation", by Wyatt Kash and September 26, 2005 HUD news release "HUD ANNOUNCES LENDER INSURANCE INITIATIVE"

Department of Energy and its partner agencies an estimated 50% of the overall development cost of the system. Additionally, OMB is exploring the possibility of reusing this component again to create a new system for a natural disaster emergency application system. This rules engine component is available to government through CORE.gov.