Systems Engineering at MITRE

SERVICE-ORIENTED ARCHITECTURE (SOA) SERIES

Leveraging Federal IT Investment Using SOA

Geoffrey Raines

MITRE

Executive Summary

Service-oriented architecture's (SOA) value proposition—SOA builds on computer engineering approaches of the past to offer an architectural approach for enterprise systems, oriented around the offering of services on a network of consumers. A focus of this service-oriented approach is on the definition of service interfaces and predictable service behaviors. A set of industry standards, collectively labeled "Web Service" standards in this paper, provide and implement the general SOA concept and have become the predominant set of practical tools used by enterprise engineers for current SOA projects. Some Web Service standards have become foundational and more widely adopted, while many are still seeking broad industry or Government acceptance.

SOA, as implemented through the common Web Services standards, offers Federal senior leadership teams a path forward, given the diverse and complex information technology (IT) portfolio that they have inherited, allowing for incremental and focused improvement of their IT support systems. With thoughtful engineering and an enterprise point of view, SOA offers positive benefits such as:

- **Language-neutral integration:** The foundational contemporary Web Services standards use extensible markup language (XML), which is focused on the creation and consumption of delimited text. Regardless of the development language that your systems use, these systems can offer and invoke services through a common mechanism. Programming language neutrality is a key differentiator from past integration approaches.
- **Component reuse:** Given current Web Service technology, once an organization has built a software component and offered it as a service, the rest of the organization can then utilize that service. With proper service governance, emphasizing topics such as service provider trust, service security, and reliability, Web Services offer the potential for aiding the more effective management of an enterprise portfolio, allowing a capability to be built well once and shared. Multiple components can be combined to offer greater capabilities in what is often termed "orchestration."
- **Organizational agility:** SOA defines building blocks of software capability in terms of offered services that meet some portion of the organization's requirements. These building blocks, once defined and reliably operated, can be recombined and integrated rapidly.
- **Leveraging existing systems:** One common use of SOA is to define elements or functions of existing application systems and make them available to the enterprise in a standard agreed-upon way, leveraging the substantial investment already made in existing applications. The most compelling business case for SOA is often made regarding leveraging this legacy investment, enabling integration between new and old systems components.

The benefits mentioned above will accrue only as the result of comprehensive engineering and a meaningful architecture at the enterprise level. SOA as a service concept in no way eliminates the need for strong software development practices, requirements-based lifecycles, and an effective enterprise architecture. While SOA done right offers valuable benefits, SOA without structured processes and governance will lead to traditional large software system problems.

Choosing to initiate an enterprise-wide SOA brings with it several key considerations for a senior leadership team. SOA offers a means to effectively leverage

decades of software investment, while providing a growth path for new capabilities. Portions of legacy applications, which may have taken many years and substantial resources to build, can be "wrapped" and integrated into modern service frameworks, incrementally leveraging significant past investment, as resources allow. Web Services can provide a technical underpinning for structuring portfolios as a collection of discrete software services, each with a definable customer base, acquisition strategy, performance levels, and a measurable operational cost. However, in order to achieve these positive outcomes, architectural activities, such as standards selection, security architectures, and service cataloging, must occur at the enterprise level. A key architectural activity is deciding whether a wholly commercially based SOA approach is appropriate under the specific circumstances and requirements; it is not always the right choice.

The remainder of the paper focuses on a series of conceptual topics important to a Federal senior leadership team considering SOA, such as how SOA compares to integration approaches of the past, how component-based approaches have changed other large industries, how component reuse can benefit the enterprise as a whole, how enterprise standards can enable software component interoperability across an organization, and where the benefits of SOA tend to accrue. The following topics are examined in more detail:

Integration—Enterprise application integration (EAI) is a field of study in computer science that focuses on the integration of "systems of systems" and enterprise applications. With the span of attempted systems integration and data sharing expanding in large organizations, the EAI engineering discipline has become increasingly central to senior leadership teams managing portfolios of applications. SOA can be considered another important step in the over 30 year history of EAI technologies. The various historical technologies have differed in the ease with which integration could occur from a programmer's point of view, underlying network configurations (e.g., ports required to be open on a network), the quantity of enterprise equipment to operate, and general design approaches to fault tolerance when failures occur.

Using components—Historic analogy with integrated circuits—During the 1970s electronics engineers experienced an architectural and design revolution with the introduction of practical, inexpensive, and ubiquitous Integrated Circuits (ICs). This revolution in the design of complex hardware systems is informative for contemporary software professionals now charged with building enterprise software systems using the latest technologies of Web Services in the context of SOAs. Like SOA, the IC revolution was fundamentally a distributed, multi-team, component-based approach to building larger systems. Through the commercial marketplace, corporations successfully built components that could be described, procured, and reused by engineering teams distributed around the world.

Reuse—Reuse of a service differs from source code reuse in that the external service is called from across the network and is not compiled into local system libraries or local executables. The provider of the service continues to operate, monitor, and upgrade the service, while the consumer of the service still needs to trust the reliability and correctness of the producer's service. The consumer must be able to find the service and have adequate documentation accurately describing the behavior and interface of the service. Performance of the service is still key.

Mature SOAs should measure reuse as part of a periodic portfolio management assessment. The assessment of reuse can be effectively integrated into the information repository used for service discovery in the organization, called the "enterprise catalog." Since changes to a service over time will require that the service's consumers be remembered and notified, it is a small step further to quantify the current consumers for a service for the purposes of portfolio management and reuse assessment.

Creating a generic reusable software component for a broad audience takes more resources (20 percent to 100 percent more) than creating a less generic point solution. The cost of reuse, therefore, shifts to the service providers and benefits the consumers. Consequently, as the enterprise decides to fund service providers, there is great benefit in maximizing the number of consumers for an operational service.

Acquiring reuse—Many of the current trends in performance-based contracting work well with the acquisition of SOA services. For example, Office of Management and Budget (OMB), performance-based service contracting (PBSC) is true to the underlying spirit and architecture of an SOA's service, which focuses on the result of the service,

not on specifying an implementation or "how" the service's work is to be done.

Reuse of services on an enterprise scale is a team effort, but Government leadership has a singular responsibility to strategically guide enterprise IT expenditures. Planned acquisitions must match the overall portfolio goals of the organization, and many organizations are establishing review boards for this purpose. If a service is meant to be reused as a common component for a series of programs or projects, contract language and incentives must be explicitly organized around that goal.

Enterprise standards—When many components are being simultaneously developed by individual teams, it becomes critical for the interface of a provider's service to match up to the "call" of a consumer. Similarly, it helps everyone involved if the interfaces across services have some commonality in structure and security access mechanisms. Choosing and communicating a comprehensive set of enterprise standards is a responsible approach to aid in enterprise SOA integration.

Where SOA works best—The Web Service technologies commonly used today to implement SOA concepts have certain design presumptions. They work best when the underlying network is robust, reliable, and available. This is not to say that any deficiency in the underlying network can not be compensated for by thoughtful engineering and the use of standard queuing and buffering communications methods. However, employing these alternative approaches to compensate for the underlying network will take a project further from the mainstream commercial implementations of Web Services.

Agility—When we discuss "agility" as it relates to SOA, we are often referring to organizational agility, or the ability to more rapidly adapt a Federal organization's tools to meet their current requirements. An organization's requirements of IT might change over time for a number of reasons, including changes in the business or mission, changes in organizational reporting requirements, changes in the law, new technologies in the commercial marketplace, and attempts to combine diverse data sources to improve the organization's operational picture. The larger promise of an enterprise SOA is that once a sufficient quantity of legacy-wrapped components exists and is accessible on the IP wide area network (WAN), the components can be re-assembled more rapidly to solve new problems.

SOA's beneficiaries—Efforts that benefit the chief information officer's (CIO's) enterprise, and look good to the senior leadership team of an organization, do not necessarily benefit the small software projects in an agency. Transitioning a legacy application to expose a set of Web Services, and putting the services in place with a robust infrastructure of redundant 24x7 reliable servers with full support as well as a service discovery mechanism, is an expensive task, hopefully enabled by enterprise level infrastructure efforts. If, as a result of creating a good service, an individual project then picks up many more consumers than it had previously, then clearly the day-to-day demands on the project's IT infrastructure increase. The common result of service success is higher local operational costs for the project offering a service. At the enterprise level, this can be a significant benefit because it means that more customers are reusing the same shared services, instead of rebuilding them.

In summary, SOA is an enterprise effort, and the local perspective of individual legacy projects will not justify an enterprise SOA effort, but this should not be allowed to stop the enterprise SOA from occurring. The SOA benefits accrue largely at the enterprise's level in cost avoidance through reuse, and increased data exchange and agility. Consequently, a corresponding investment is required at the enterprise level, where the benefit is found.

For more information on SOA, see http://www.mitre.org/soa.

Table of Contents

THE BIG PICTURE: SOA builds on computer engineering approaches of the past to offer an architectural approach for enterprise systems, oriented around the offering of services on a network of consumers. SOA, as implemented through the common Web Services standards, gives Federal senior leadership teams a path forward, allowing for incremental and focused improvement of their IT support systems.

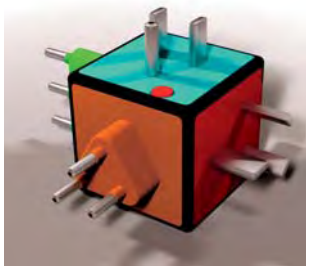
Leveraging Federal IT Investment Using SOA

Geoffrey Raines

SOA—Value Proposition

Contemporary issues for Federal IT decision makers—Similar to the nation's Fortune 500 leadership, today's Federal leadership teams often find themselves facing significant IT investment and portfolio challenges. They have inherited a computing infrastructure that is often not uniform and whose technologies span the recent history of computing. The IT infrastructures tend to have the following characteristics:

- **Diverse environments:** Mainframe systems, client/server systems, and multi-tier Web-based systems sit side by side, demanding operations and maintenance resources from a technology marketplace in which the cost of niche legacy technical skills continues to rise. The portfolio of systems is generally written in a number of different software development languages such as COBOL, Java, assembly, Ada, and C, requiring heterogeneous staff skill sets and experience in a variety of commercial products, some of which are so old that they no longer offer support licenses.
- **Complex business logic:** The systems often conform to a set of complex business logic that has developed over a number of years in response to evolving legal requirements, Congressional reporting mandates, changes in contractor teams, and refinement of business processes. While some systems are new and robust, many are brittle and hard to modify, relying on technical skills not common in the marketplace that become increasingly more expensive. The maintenance tail on these systems is surprisingly high and competes for resources with required new functionality.
- **Inconsistent interfaces:** Interfaces between systems have grown up spontaneously without enterprise planning, over many years.[1] The interfaces are the result of unique singular negotiations between various parts of the organization and have been designed using many varied technologies during the organization's IT history, following no consistent design pattern. Recent enterprise architecture efforts have documented the enterprise interfaces in diagrams that resemble a Rorschach inkblot test.
- **Limited sustainment budgets:** Even without the continuous downward pressure on IT budgets brought by competing national requirements, and the view that IT should be increasingly viewed as a commodity, there are not enough budget resources or human resources to recast the portfolio of systems to be modern and robust in one action. "According to analysts at Forrester Research, there are some 200 billion lines of COBOL, the most



"After creating islands of automation through generations of technology, users and business managers are demanding that seamless bridges be built to join them." [1]

– David Linthicum

popular legacy programming language, still in use. Nor is it going away: maintenance and modifications to installed software increase that number by five billion lines a year. IBM meanwhile claims its CICS mainframe transaction software handles more than 30 billion transactions per day, processes $1 trillion in transaction values, and is used by 30 million people." [2] Given budget constraints, an incremental approach seems to be required.

SOA's value proposition— SOA builds on computer engineering approaches of the past, to offer an architectural approach for enterprise systems, oriented around the offering of services on a network of consumers. A focus of this service-oriented approach is on the definition of service interfaces and predictable service behaviors. A set of industry standards, collectively labeled "Web Service" standards in this paper, are employed to implement the general SOA concept, and have become the predominant set of practical tools used by enterprise engineers for current SOA projects. Some Web Service standards have become foundational and more widely adopted, while many are still seeking broad industry or Government acceptance.

SOA, as implemented through the common Web Services standards, offers Federal senior leadership teams a path forward, given the diverse and complex IT portfolio that they have inherited, allowing for incremental and focused improvement of their IT support systems. With thoughtful engineering and an enterprise point of view, SOA offers positive benefits such as:

- **Language-neutral Integration:** Web-enabling applications with a common browser interface became a powerful tool during the 1990s. In the same way that HTML defined a simple user browser interface that almost all software applications could create, Web Services define a programming interface available in almost all environments. The HTML interface at the presentation layer became ubiquitous because it was easy to create, being composed of textual characters. Similarly, the foundational contemporary Web Services standards use XML, which again is focused on the creation and consumption of delimited text. The bottom line is that regardless of the development language your systems use, your systems can offer and invoke services through a common mechanism. However, note that XML does not by itself solve issues with data's semantic consistency across organizations. The Rosetta Stone, an Egyptian artifact that was instrumental in advancing our translation of ancient writing, has text that is made up of three translations of a single passage.[3] The Stone allowed translators to understand text in unknown languages by utilizing languages they knew. Contemporary Web Service standards provide a "Rosetta Stone" across programming languages and software development environments and can be leveraged for the purpose of enterprise systems integration. The term Rosetta Stone has become idiomatic as something that is a critical key to a process of translation of a difficult problem. SOA, as implemented through Web Service standards, provides a common enterprise integration technology for the multiple computing environments, and languages that arise in the typical Federal IT portfolio. Enterprise integration standards and their use in a large SOA effort are discussed further below.

- **Component reuse:** Given current Web Service technology, once an organization has built a software component and offered it as a service, the rest of the organization can then utilize that service. Given proper service governance, including items such as service provider trust, service security, and reliability, Web Services offer the potential for aiding the more effective management of an enterprise portfolio, allowing a capability to be built well once and shared, in contrast to sustaining redundant systems with many of

the same capabilities (e.g., multiple payroll, trouble ticket, or mapping systems in one organization). Reuse, through the implementation of enterprise service offerings, is further discussed below.

- **Organizational agility:** SOA defines building blocks of software capability in terms of offered services that meet some portion of the organization's requirements. These building blocks, once defined and reliably operated, can be recombined and integrated rapidly. Peter Fingar stated, "Classes, systems, or subsystems can be designed as reusable pieces. These pieces can then be assembled to create various new applications."[4] Agility, the ability to more rapidly adapt a Federal organization's tools to meet their current requirements, can be enhanced by having well-documented and understood interfaces and enterprise-accessible software capabilities. Organizational agility, as enhanced by a consistent enterprise-scoped SOA, is discussed below.
- **Leveraging existing systems:** One common use of SOA is to encapsulate elements or functions of existing application systems and make them available to the enterprise in a standard agreed-upon way, leveraging the substantial investment already made. The most compelling business case for SOA is often made regarding leveraging this legacy investment, enabling integration between new and old systems components. When new capabilities are built, they are also designed to work within the chosen component model. Given the size and complexity of the installed Federal application system base, being able to get more value from these systems is a key driver for SOA adoption. David Litwack writes, "The movement toward Web Services will be rooted not in the invention of radical new technology, but rather in the Internet-enabling and re-purposing of the cumulative technology of more than 40 years. Organizations will continue to use Java, mainframe and midrange systems, and Microsoft technologies as a foundation for solutions of the future."[5]

The benefits mentioned above, however, accrue only as the result of comprehensive engineering and a meaningful architecture at the enterprise level. SOA and service concepts in no way eliminate the need for strong software development practices, requirements-based lifecycles, and an effective enterprise architecture. While SOA done right offers valuable benefits, SOA without structured processes and

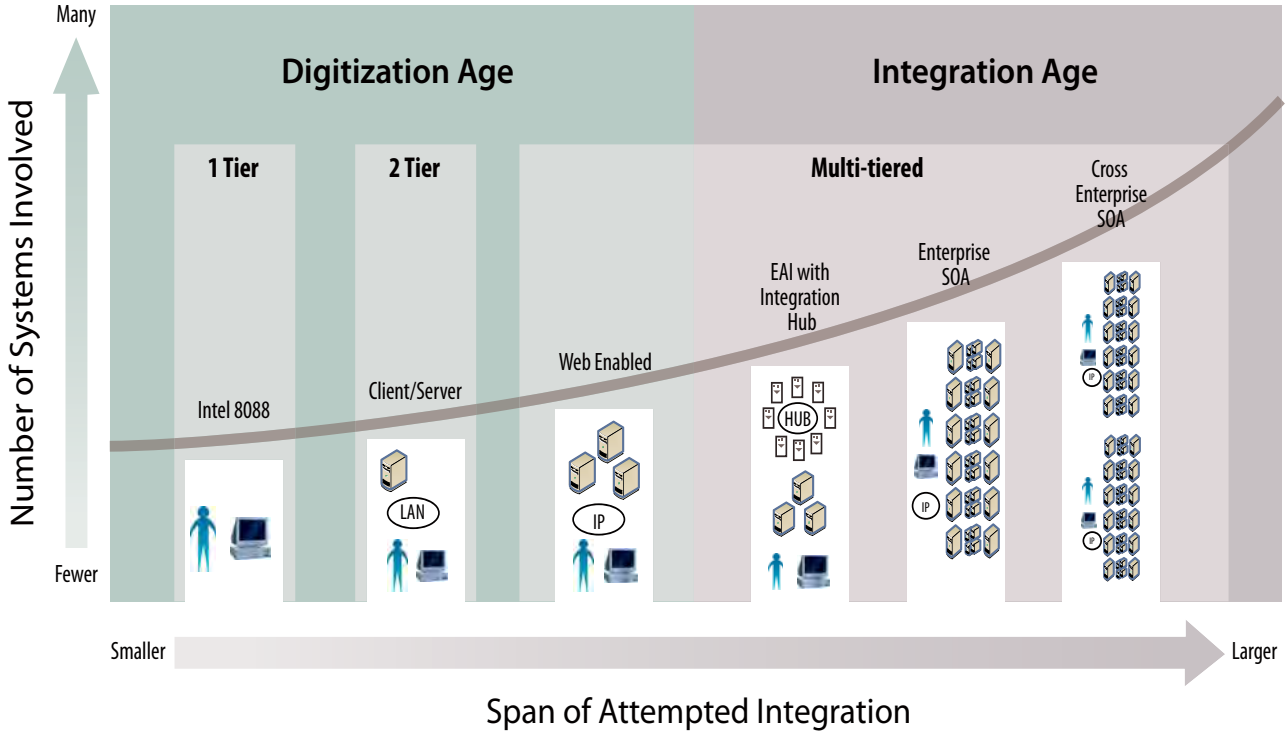


Figure 1: Integration Is Increasing In Scope and Complexity

governance will lead to traditional software system problems.

SOA—Why now?—SOA and its implementing standards, such as the Web Services standards, come to us at a particular point in computing history. While several key improvements, such as language neutrality, differentiate today's Web Service technologies, there has been a long history of integrating technologies with qualities analogous to Web Services, including a field of study often referred to as enterprise application integration (EAI). One of the key trends driving the adoption of Web Services is the increasing span of integration being attempted in organizations today. Systems integration is increasing both in complexity within organizations and across external organizations. We can expect this trend to continue as we combine greater numbers of data sources to provide higher value information. Ronan Bradley writes, "CIOs often have difficulty in justifying the substantial costs associated with integration but, nevertheless, in order to deliver compelling solutions to customers or improve operational efficiency, sooner or later an organization is faced with an integration challenge."[6] Figure 2 above depicts a few waypoints in the trend toward increasing systems integration complexity.

Drawing Parallels—Past Is Prologue

During the 1970s electronics engineers experienced an architectural and design revolution with the introduction of practical, inexpensive, and ubiquitous integrated circuits (ICs). This revolution in the design of complex hardware systems is informative for contemporary software professionals now charged with building enterprise software systems using the latest technologies of Web Services in the context of SOAs.

Like SOA, the IC revolution was fundamentally a distributed, multi-team, component-based approach to building larger systems. Through the commercial marketplace, corporations built components for use by engineering teams distributed around the world. Teams of engineers created building blocks in the form of IC components that could then be described, procured, and reused.

What can we learn from the IC revolution of the 1970s? How can component-based architectures change the approach of an entire industry?

Like software services, every IC chip has a defined interface. The IC interface is described in several ways. First, the chip has a **defined function**—a predictable behavior that can be described and provides some value for the consumer. Next the physical dimensions of the chip are enumerated. For example, the number and shape of pins is specified. Further, the electronic signaling, timing, and voltages across the pins are specified. All these characteristics make up the total **interface definition** for the IC. Of course, software services do not have an identical physical definition, but an analogous concept of a comprehensive interface definition is still viable. Effective software components also possess a predictable and definable behavior.

Introducing and using ICs included the following considerations:

- **Who pays?** Building an IC chip the first time requires a large expenditure of resources and capital. The team who builds the IC spends considerable resources. The teams who reuse an IC instead of rebuilding it save considerable time and expense. A chip might take $500K to build the first time and might be available for reuse in a commercial catalog for $3.99. The creation of the chip the first time involves many time-consuming steps, including requirements analysis, behavior definition, design layout, photolithography, testing, packaging, manufacturing and marketing.[7] The team who gets to reuse the chip instead of re-building it, saves both time and dollars. At the time, designs of over 100,000 transistors were reported as requiring hundreds of staff-years to produce manually.[8]
- **Generic or specialty components?** Given the amount of investment required to build a chip,

designs were purposely scoped to be generic or specific, with particular market segments and consumer audiences in mind. Some chips only worked for very specific problem domains, such as audio analysis. Some were very generic and intended to be used broadly, like a logic multiplexer. The bigger the market, and the greater the potential for reuse, the easier it was for a manufacturer to amortize costs against a broader base, resulting in lower costs per instance.

- **Increased potential design scope:** By combining existing chips into larger assemblies, an engineer could quickly leverage the power of hundreds of thousands of transistors. In this way, IC reuse expanded the reach of the average engineer, allowing the engineer to leverage resources and dollars spent far in excess of the local project budget.
- **Design granularity:** The designer of an IC had to decide how much logic to place in a chip to make the chip most effective on the marketplace. Should the designer create many smaller function chips, or fewer larger function chips? Families of chips were often built with the intention of their functions being used as a set, not unlike a library of software functions. Often these families of chips had similar interface designs, such as consistent signal voltages.
- **Speed of integration:** As designers became familiar with the details of component offerings and leveraged pre-built functions, the speed at which an "integrated" product, built of many components, could come to market was substantially increased.
- **Catalogs:** When the collection of potential ICs offered became large, catalogs of components were then created, and classification systems for components were established. Catalogs often had a combination of sales and definitive technical information. The catalogs often had to point to more detailed resources for the technical audiences that they sold components to.
- **Testing:** Technical documents defined the expected behavior of ICs. Components were tested by both the manufacturer and the marketplace. Anomalous behavior by ICs became noted in errata in technical specifications.
- **Engineering support:** IC vendors offered advanced technical labor support to customers in the form of application engineers and other technical staff. Helping customers use the products fundamentally supported product sales.
- **Value chains:** Value chains consume raw components and produce more complex, value-added offerings. ICs enabled value chains to be created as collections of chips became circuit boards, and collections of circuit boards became products.
- **Innovation:** ICs were put together in ways not anticipated by their designers. Teams who designed chips could not foretell all the possible uses of the chips over the years. Componentized logic allowed engineers to create innovative solutions beyond the original vision of component builders.

Did it work?—One might ask, "Were electrical engineers successful with this component-based approach?" Certainly the marketplace was populated by a very large number of offerings based in some part on ICs. Certainly many fortunes and value chains were created. The cost effectiveness of the reuse approach was validated by the fact that it became the predominant approach of the electronics industry. In short, electronic offerings of the time could not be built to market prices if each chip, specification, module, or component had to be refabricated on each project. Reuse, through component-based methods, enabled by new technologies, led this revolution. Yet, the transformation took a decade to occur.

SOA analogy—In many ways the IC chip revolution described above is analogous to the effort underway with Web Services today. Clearly Web Service components have analogous interfaces definitions, and defined and documented behaviors that provide some benefit to a potential consumer. One can also reasonably expect that the team producing the Web Service will incur substantial expenses that consumers of the service will not. For example, high reliability requirements for the operation of a service and its server and network infrastructure can be a new cost driver for the provider. Historically, designing software for reuse generally drives the cost up by a factor of 1.15 to

2.0, and this may be an additional cost driver for a provider.[9] To continue the analogy, collections of service offerings are becoming sufficiently large to require some librarian function to organize, catalog, and describe the components. Many SOA projects use a service registry such as universal description, discovery, and integration (UDDI) for this purpose. Enterprise integration engineers are realizing the ability to more rapidly combine network-based service offerings, and a new paradigm, sometimes referred to a "mashup," is demonstrating the speed at which integration can now occur.[10] Value chains of data integration are already occurring in the marketplace. A data integrator can ingest the product of multiple services and produce a service with correlated data of greater value. Finally, it is also safe to say that service providers may be surprised at how their services get integrated over time, and they may be part of larger integration that they could not have foreseen during the original design. (Also note that this same component-based approach is now being examined for genetics work as well. The same interface definition, behavior, cataloging, and reuse discussions are currently occurring, creating a new genetic sub-field known as *synthetic genetics*.[11]) In summary, many aspects of the current SOA efforts follow similar component-based patterns, and many of the benefits realized historically by the IC revolution will be potentially realized by SOA efforts.

Reuse

Historic source code reuse—During the 1980s many organizations, including the Department of Defense (DoD), attempted to reuse source code modules with little success. For example, during the DoD's focus on the Ada language, programs were established to reuse Ada language functions and procedures across projects.[12] The basic reuse premise outlines a process where a producer of a source code module would post the source code to a common shared area along with a description of its purpose and its input and output data.[13] At that point, staff from another project would find the code module, download it, decide to invoke it **locally** in their source code, and actually compile it into their local libraries and system executables. As an example, the *Ada Quality and Style Guide* states that, "One of the design goals of Ada was to facilitate the creation and use of reusable parts to improve productivity. To this end, Ada provides features to develop reusable parts and to adapt them once they are available."[14] For example, *Project A* might create a high-quality sorting function, and *Project B* could then compile that function into their own software application.

Reuse is a critical characteristic of the SOA value proposition for a large organization, but we have to be careful how we characterize reuse.

Though well intentioned, the actual discovery and reuse of the source code modules did not happen on a large scale in practice. Reasons given for the lack of reuse at the time included lack of trust of mission-central requirements to an external producer of the source code, failure to show a benefit to the contractor "reuser" implementing later systems, inadequate descriptions of the behavior of a module to be reused, and inadequate testing of all the possible outcomes of the module to be reused.[15] All in all, the barriers to reuse were high.

Service reuse—The danger in describing the use of services as "reuse" is that the reader will assume we mean the source code reuse model of the 1980s described above. We don't. In fact, the nature of service reuse is closer to the model of the reuse of ICs by electrical engineers described above, though still having common issues of trust, defined behavior, and expected performance. In plain terms, reuse in the service context means **not rebuilding a service, but rather the using again, or invoking, of a service built by someone else**.

The enterprise as a whole saves resources every time a project decides to reuse a current software service, rather than creating redundant services based on similar underlying requirements and adding to an agency's maintenance portfolio. Since a system's maintenance costs often exceed the cost to build them, over their lifetime, the enterprise saves not only in the development and establishment cost of a new service but also in the 20-plus-year maintenance cost over the service's lifecycle. One web vendor stated, "Web services reuse is everything: on top of the major cost savings …, reuse means there are fewer services to maintain and triage. So reuse generates savings—and frequency of use drives value in the organization."[16] However, we should

not assume a straight-line savings, where running one service is exactly half as costly as running two services, because the cost of running a service is also impacted by the number of service consumers. Consolidation can make the remaining service more popular, with a greater demand on resources.

> *"Certainly if you were measuring SOA success, and you should of course, then an obvious measure is service reuse."*
>
> *—National Practice Director for SOA, Perficient, Inc.*[18]

Reuse of a service differs from source code reuse in that the external service is called from across the network and is not compiled into local system libraries or local executables. The provider of the service continues to operate, monitor, and upgrade the service as appropriate. Thanks to the benefits of contemporary Web Service technologies, the external reused service can be in another software language, use a completely foreign multi-tiered or single-tiered machine architecture, be updated at any time with a logic or patch modification by the service provider, represent 5 lines of Java or 5 million lines of COBOL, or be mostly composed of a legacy system written 20 years ago. In these ways service reuse is very different from source code reuse of the past.

Some aspects of reuse remain unchanged. The consumer of the service still needs to trust the reliability and correctness of the producer's service. The consumer must be able to find the service and have adequate documentation accurately describing the behavior and interface of the service. Performance of the service is still key. ZDnet stated, "Converging trends and business necessity—above and beyond the SOA "vision" itself—may help drive, or even force, reuse. SOA is not springing from a vacuum, or even from the minds of starry-eyed idealists. It's becoming a necessary way of doing business, of dispersing technology solutions as cost effectively as possible. And, ultimately, providing businesses new avenues for agility, freeing up processes from rigid systems." [17]

Mature SOAs should measure reuse as part of a periodic portfolio management assessment.[18] Actional wrote, "Reuse is not only a key benefit of SOA, but also something that you can quantify. You can measure how many times a service is being used and how many processes it is supporting, thus the number of items being reused. This enables you to measure the value of the service. With a little work, you can calculate the service cost savings for each instance of reuse, including saved architecture and design time, saved development time, and saved testing time." [19] The assessment of reuse can be effectively integrated into the information repository used for service discovery in the organization, the enterprise catalog. Since changes to a service over time will require that the service's consumers be remembered and notified, it is a small step further to quantify the current consumers for a service for the purposes of portfolio management and reuse assessment.

Reuse costs—Barry Boehm provided two useful formulas when estimating the costs of software systems reuse. One formula is from the provider's point of view, while the other is from the consumer's.[20]

Provider-focused formula:

$$\text{Relative Cost of Writing for Reuse (RCWR)} = \frac{\text{Cost of Developing Reusable Asset}}{\text{Cost of Developing Single-Use Asset}}$$

Consumer's formula:

$$\text{Relative Cost of Reuse (RCR)} = \frac{\text{Cost to Reuse Asset}}{\text{Cost to Develop Asset from Scratch}}$$

Jeffery Poulin examined large-scale SOA service providers to estimate the value ranges for these formulas in practice.[22] His data shows that RCWR ranges between 1.15 and 2.0 with a median of 1.2, while RCR ranges between .15 and .80 with a median of .50. In other words, Paulin's work suggests that creating a generic reusable software component for a broad audience takes more resources (15 percent to 100 percent more) than creating a less generic point solution. The cost of reuse therefore shifts to the providers and benefits the consumers. Consumers spend less (median 50 percent less) to reuse the service than to create their own. We can see from these formulas that as the enterprise decides to fund service providers, there is great ben-

efit in maximizing the number of consumers for an operational service.

Acquiring reuse—Many of the current trends in performance-based contracting sponsored by the incumbent administration work well with the acquisition of SOA services. For example, OMB performance-based service contracting (PBSC) is true to the underlying spirit and architecture of an SOA's service, which focuses on the result of the service, not on specifying an implementation or "how" the service's work is to be done. As a consumer of an SOA service we care most about the service's interface and its performance characteristics. Similarly, PBSC also focuses on the performance characteristics of the vendor's service to the Government. OMB states, "The key elements of a PBSC performance work statement (PWS) are: a statement of the required services in terms of **output**; a **measurable performance standard** for the output; and an acceptable **quality** level (AQL)." [23]

OMB writes, "Performance-based contracting methods are intended to ensure that required performance quality levels are achieved and that total payment is related to the degree that services performed meet contract standards." [24] The key is that service outcomes are to be measured and expectations are defined. OMB states further, "The definitions of standard performance, maximum positive and negative performance incentives, and the units of measurement should be established in the solicitation." Both these ideas have a parallel in an SOA service. As an SOA service provider, one carefully defines the offering to the enterprise. Service performance requirements drive the quantity of underlying infrastructure run by the service provider and therefore drive the provider's cost. If a contract is crafted to provide an SOA service to the enterprise, the expected service levels will drive the estimated cost of the service and should be considered carefully.

Reuse of services on an enterprise scale is a team effort, but Government leadership has a singular responsibility to strategically guide enterprise IT expenditures. Often these decisions are guided by an enterprise architecture (EA) effort. Planned acquisitions must match the overall portfolio goals of the organization, and many organizations are establishing review boards for this purpose. If a service is meant to be reused as a common component for a series of programs or projects, contract language and incentives must be explicitly organized around that goal. Goodwill or positive intentions are not sufficient. Portfolio management and scarce resources will demand that Government staff rein in desires of contractors or even project teams to create redundant systems and services. The Government must establish processes and organizations to assess and enforce prohibitions against the creation of redundant capability. This requires both technical skills to understand potential architectural solutions and contracting skills to structure existing Federal Acquisition Regulation (FAR)-based contracting tools with appropriate objective-driven language. Given the trend for the expansion of attempted integration as described above, redundancy of IT capability will only become more visible over time.

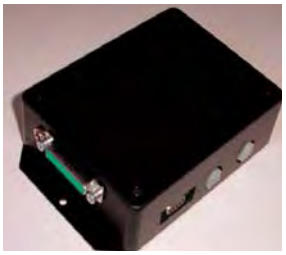
Using SOA for Enterprise Integration

EAI is a field of study in computer science that focuses on the integration of "systems of systems" and enterprise applications. Wikipedia states that, "EAI is a response to decades of creating distributed monolithic, single purpose applications leveraging a hodgepodge of platforms and development approaches. Attending to EAI involves looking at the system of systems, which involves large-scale inter-disciplinary problems with multiple, heterogeneous, distributed systems that are embedded in networks at multiple levels." [25] With the span of attempted systems integration and data sharing continually increasing in large organizations, the EAI engineering discipline has become increasingly central to senior leadership teams managing portfolios of applications.

The fundamental EAI tenets are based on traditional software engineering methods, though the scale is often considerably larger. While the traditional software coder focused on the parameters that would be sent to, and received from, a function or procedure, the EAI engineer focuses on the parameters that are exchanged with an entire system. The traditional coder might have been writing 100 source lines of code (SLOC) for a function, while the EAI engineer

might be invoking a system with a million SLOC and several tiers of hardware for operational implementation. However, the overall request/response pattern is the same, and the logic issues like error recovery must still be handled gracefully in either case.

Overall, the EAI engineer is looking for the following characteristics in an enterprise integration solution:



- **Open architecture:** An open architecture, independent of underlying programming languages, and application platforms. The architecture should focus on allowing systems to communicate in a loosely coupled fashion, allowing any application or system to map its own internal architecture to well defined external interfaces. Ronan Bradley writes, "It is with the introduction of 'loosely coupled' architectures that SOA has emerged as a truly viable means of delivering business and IT agility. In a loosely coupled system, each service simply presents a standard interface to a common infrastructure (the SOA itself). Implementation is hidden behind this interface, and as a consequence services can be swapped, adapted or reconfigured at will—hence the term loosely coupled; there is no tight link between the service implementation and the client requesting that service." [26]
- **Layered model:** Use of a layered model, with hierarchy and modularity to support the composition of smaller services in the creation of a larger and more fully functional service. The invocation of one service may lead to the invocation of other services that execute parts of the larger service request.
- **Exploit standards implemented in COTS:** Maximize use of current and emerging commercial-off-the-shelf (COTS) standards, technologies, and products. Minimize customization and modification of commercial products and focus research and development activity on unique organization missions and requirements. Services should be designed with minimal dependence on vendor proprietary implementations.
- **Scale to global proportions:** The architecture of the EAI integration layer needs to support graceful scaling to larger implementations with increased service capacity.
- **End-to-end management:** Services must be manageable, both in terms of their own status and performance, and in their interactions with other services. Using contemporary virtualization best practices, they should provide the means to be created, operated, and deployed in response to demand and operational needs.
- **Accommodate heterogeneity:** Services must accommodate different development models and languages. Anne Manes wrote of Web Services, "The first and most obvious bell ringer is the need to connect applications from incompatible environments, such as Windows and UNIX, or .NET and J2EE. Web services support heterogeneous integration. They support any programming language on any platform. One thing that's particularly useful about Web services is that you can use any Web services client environment to talk to any Web services server environment." [27]
- **Accommodate continual asynchronous change:** The scope of the IT infrastructure for large organizations ensures that there will always be changes occurring in some services. It will not be feasible to synchronize service changes and still remain responsive to changing user needs. Modifications to one service must not break the connections to other applications. It is unlikely that releases of new service builds will be coordinated across service providers. Of course, there will be a good deal of coordination between service providers and their current list of consumers.
- **Allow decentralized operations and management:** There will be many service providers in a large organization. An enterprise solution should support federation and interaction among the different parts comprising an end-to-end service offering.
- **Integrated, layered security:** Applications require a robust security framework that

Web Services, as a set of implementing standards for SOA, offer new value to the engineer attempting large-scale application integration.

accommodates the full spectrum of security services including authentication, authorization, integrity, confidentiality, and accountability.

SOA can be considered another important step in a 30-year history of EAI technologies. "SOA eliminates the traditional 'spaghetti' architecture that requires many interconnected systems to solve a single problem." [28] SOA's ability to run logic and functions from across a network is not new. Recent examples include Enterprise JavaBeans (EJB) by Sun Microsystems Inc., Common Object Request Broker Architecture (CORBA) by the Object Management Group, and Component Object Model (COM), Distributed Component Object Model (DCOM), and .NET from the Microsoft Corporation. The various methods have differed in the ease with which integration could occur from a programmer's point of view, the methods for conveying runtime errors, ports required to be open on a network, the quantity of enterprise equipment to operate, and general design approaches to fault tolerance when failures occur.

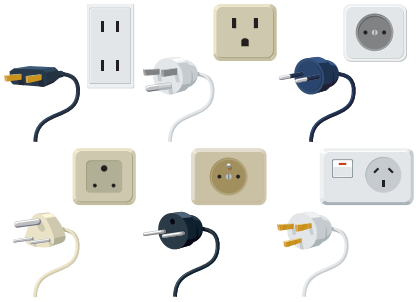
SOA as an integration concept, and Web Services as a set of implementing standards, offer something new to the EAI engineer. First and foremost, as described above, SOA Web Service implementations offer a language-neutral, platform-neutral means to connect services and systems. *DM Review* stated, "SOA provides the key to unlocking integration, by providing an enterprise-wide architectural approach to bridging applications and promoting a set of standards for rich interoperability. It's only a matter of time before this flexible way of thinking about applications makes integration technology a natural, fundamental aspect of IT infrastructure." [29]

"SOA is currently implemented through a complex set of sometimes overlapping standards, each supported by different large Industry partners."

Web Services also ease a significant enterprise integration challenge by utilizing common communications ports for integration. Individual Web Services are accessed through web servers, a common element in contemporary IT infrastructures. The key point here is that the ports and protocols to access web servers are usually already defined (e.g., port 80 HTTP), and open across an organization, both in policy and implementation. This means that the firewalls and access control points are more likely to be friendly to this type of data exchange, as compared to suggesting that an organization open up a whole new set of ports and protocols for integration.

Enterprise SOA Standards

The need for enterprise standards—SOA programs are most often enterprise-level endeavors involving "teams of teams" who control "systems of systems." Personnel experience ranges from experts in the organization's data sources and legacy systems, to EAI engineers with expertise in large-scale integration. Often teams in large enterprises are physically dispersed. This makes the ability to communicate the design and architecture specifications of a component an important organizational capability.



In this context, where many components are being simultaneously developed by individual teams, it becomes critical for the interface of a provider's service to match up to the call of a consumer. Similarly, it helps everyone involved if the interfaces across services have some commonality in structure and access mechanisms. The worst case would be a situation where programmer teams had to have individual staff meetings to understand interface designs with service providers every time they wanted to invoke a new service. In that situation, agility will slow to the speed of organizational dynamics, instead of the speed of coding and testing processes. Choosing and communicating a comprehensive set of enterprise standards is a good approach to aid in enterprise SOA integration.

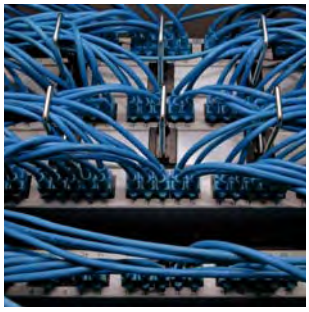
Example enterprise standards—Enterprise standards to support SOA fall into several general categories, and a typical enterprise set might look like the table below.

The current state of Web Service standards—At this time, despite the few selected in the table below, Web Service standards as a whole remain in flux. InfoQ writes, "A flurry of protocols, collectively named WS*, have also been introduced as extensions to SOAP[30] (and in some cases WSDL) to facilitate specific communication requirements and scenarios. The categories of WS* are broad, and it has reached a point where the sheer number of standards is so great that despite a core set being implemented in many platforms, many in the web service community are confused about which standards they should care about, when and why." [31] Consequently, while it is a valuable effort to select a group of standards for enterprise integration as shown in the table above, we can reasonably expect many revisions to this list in the next five years. These revisions will ripple through the community of service providers that work to comply with selected enterprise standards and the revisions will have attendant development costs.

Web Services Related	
URI	Uniform Resource Identifier (URI): Generic Syntax, January 2005
WSDL	Web Services Description Language (WSDL) 1.1, W3C Note, 15 March 2001
SOAP	Simple Object Access Protocol (SOAP) 1.1, W3C Note, 8 May 2000
HTTP	Hypertext Transfer Protocol (HTTP) 1.1, June 1999. IETF RFC 2616
Network/Network Management Related	
TCP	Transmission Control Protocol (TCP), September 1981, IETF Standard 7/RFC 793
IP	Internet Protocol (IP), September 1981. IETF Standard 5 with RFC's 791/950/919/922/792/1112
SNMP	Simple Network Management Protocol (SNMP), May 1990. IETF Standard 15/RFC 1157
Security Related	
SAML v2.0	SAML 2.0 OASIS Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard, 15 March 2005
PKI	X.509 Public Key Infrastructure Certificate
PKI CRL	X.509 Public Key Infrastructure Certifi cate and Certificate Revocation List (CRL) Profile, April 2002. IETF RFC 3280.
WS-Security	Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), OASIS Standard, March 2004.
SSL v3.0	Secure Sockets Layer (SSL) Version 3.0
XACML	eXtensible Access Control Markup Language (XACML) Version 2.0, OASIS Standard, 1 February 2005
OCSP	Online Certifi cate Status Protocol (OCSP), RFC 2560, June 1999
Registry/Directory	
UDDI v3.0.2	Universal Description, Discovery, and Integration Version 3.0.2 OASIS UDDI Spec, Dated 2004-Oct-19
LDAP v3.0	Lightweight Directory Access Protocol (v3): Technical Specification; September 2002
Data Standards	
XML	Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation 04 February 2004
XSLT	XSL Transformations (XSLT) Version 2.0, W3C Working Draft 4 April 2005
XPath	XML Path Language (XPath) 2.0, W3C Recommendation 23 January 2007
Syndication	
RSS v2.0	Really Simple Syndication (RSS) Version 2.0
Presentation Related	
HTML	HTML 4.01 Specification, W3C Recommendation, revised, 24 Dec 1999
CSS	CSS2:1998 Cascading Style Sheets, level 2 CSS2 Specification, W3C Recommendation 12 May 1998
WSRP	WSRP OASIS; OASIS Web Services for Remote Portlets Specification, August 2003
JSR-168	JSR-168; Java Specification Request (JSR) JSR-168, Portlet Specification API, Final Release ballot, Version 1.0, 06 October 2003

Where Does SOA Best Apply?

The Web Service technologies commonly used today to implement SOA concepts have certain design presumptions. They work best when the underlying network is robust, reliable, and available. Web Service standards have become an area of focus at this point in computing history because it is now conceivable to trust corporate networks in the continental United States to the task of running remote services with reasonable success. Fundamentally, Web Services allow the programmer to invoke code and application logic across the network, with input and output information. If the application under development is central to the mission of the organization, the network has to be sufficient to facilitate communication between the service provider and consumer. This is not to say that any deficiency in the underlying network cannot be compensated for by thoughtful engineering and the use of standard queuing and buffering communications methods. However, these approaches and standard design patterns to compensate for the underlying network will take a project further from the mainstream commercial implementations of Web Services. Several Federal projects work in environments where the underlying network is not on par with the CONUS corporate Internet, and those projects assume greater risk in diverging from mainstream standards in order to implement SOA. Web Services assume a reasonable network.

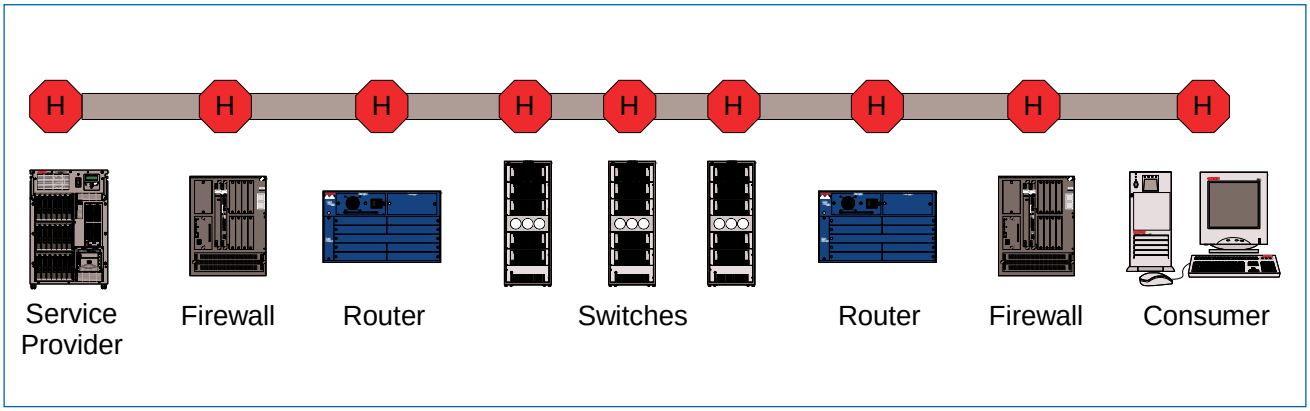


Unreliable or low-bandwidth networks—There are several characteristics that are important to defining the quality of the underlying network. The network can fail a Web Service implementation for several reasons such as, but not limited to:

- **Bandwidth:** Insufficient bandwidth to carry the large (and often inefficient) XML payloads between service provider and consumer within desired performance requirements.
- **Reliability:** Network components that lose a sufficient portion of the IP packets between a service provider and consumer so that performance requirements are not met.
- **Intermittent communications:** Sporadic communication between the service provider and consumer that turns what might have been a rapid request/response pair into a form of buffered asynchronous communications.

In these cases compensating software designs can be put in place to make up for the deficiencies in the underlying networks. Traditional methods to compensate for poor communications include extra error checking and error recovery logic, including the ability to retransmit messages or parts of messages when needed, and the ability to queue communications in buffering architectures until one of the parties can attend to it. For example, a web server offering standard HTTP on port 80 out of the box will not perform all these compensating functions. These designs will take the engineer further from the common commercial implementations of Web Services and make the application of COTS products less likely. In some extreme environments, such as the forward edge of a battle field, diverging from commercial products will be required, and that alone should not stop designers from being service oriented. However, we must recognize that as the software system becomes less based on industry standard approaches and patterns, and becomes more of a one-off custom design solution for one problem space, the risk profile for the project changes.

High reliability requirements—However, it's not just the extreme network cases in which Web Services offer some concern. Mary Brandel astutely points out that, "Before mission-critical Web services applications enter the mainstream, reliable messaging will have to become less complex and costly." [32] As discussed above, Web Services are being used as an integration tool by many organizations, and consequently they are being directly compared to many existing highly robust integration tools. For example, integration brokers are used in the banking industry to transfer large sums of money. This is an area where the software cannot get it wrong, and consequently the capabilities for assured delivery and non-repudiation are mature. There are ongoing attempts by several of the Web Services standards bodies to replicate these capabilities in standards that hope to be broadly adopted by industry. It is safe to say that given current Web



Service implementations, very high reliability delivery mechanisms are not sufficient. Of course, as was mentioned in the network discussion, thoughtful engineering can compensate for these issues, but the solutions become non-standard.

Real-time processing requirements—Given the state of contemporary Web Service technologies, real-time processing is a significant challenge. There are several performance issues with Web Services and the underlying premise of running services across a network. Performance challenges can include the marshaling of XML data, network propagation delays, and the underlying security design pattern especially in the area of services calling services, or service chaining. And while the definition of "real-time" can vary, the problems outlined below affect most classes of real-time systems.

> *"You want to be cautious when trying to use Web services in situations with stringent requirements for real-time performance."*
>
> *—Anne Thomas Manes* [33]

For example, several large-scale projects have reported that the marshaling of data, both in and out of Web Service calls, and rendering XML is a low-performance activity.[34] Converting organically binary data into ASCII formats for inclusion in XML is prohibitively slow for many real-time applications. Anne Manes writes, "XML is tremendously versatile, but it isn't the most compact or efficient mechanism for transferring data. A SOAP message is much bigger than a comparable native binary message used with RPC, RMI, CORBA, or DCOM. It also takes a lot more time to process an XML message than a binary message. Even with the best-performing implementations, SOAP messaging can take 10 to 20 times longer than RMI or DCOM." [35]

Web Service technologies share challenges that have existed for years with large distributed systems. On a contemporary IP network, the distance from a service provider to a service consumer is measured in "hops." As shown in the inset figure, at each hop, time is spent performing some action on a packet, such as routing it, or inspecting its contents. Some hops are fast (low latency), such as switches, while some hops are very slow (high latency), such as firewalls with content checking rules. Thirty or more hops would not be unusual for a typical packet. In total these hops add up to some network propagation delay from the point of view of the service-level software. The number and types of hops from the provider to the consumer directly affects perceived performance of the service.

Even though the service provider often cannot control the wide area network (WAN), the ability to effectively run a service is impacted by the service provider's location on the network topology. In the commercial world, service providers pay extra fees to host their servers a minimum number of hops off of the main IP exchange points on the Internet. Finally, also consider that the IP-based Internet is dynamically routed. This means that from moment to moment, and day to day, the path that the IP packet must take will change. For all these reasons, running services across a network can be risky for real-time applications.

An often overlooked point is that performance of each service provider is localized and unknown to the consumer, moment to moment. For example, a world event may cause thousands of end users to start their browsers and cause a particular service to be launched. All these service calls will come into the same service at about the same time. Each end user does not know that the same query might run a hundred times faster at another moment, but due to resource contention, the response will be momentarily poor.

In this sense, the consumer does not know, moment to moment, the status of the provider. There are local and global load balancing approaches that service providers put in place to compensate for this issue, but overall it is another reason why performance for real-time applications can be unpredictable.

Security designs can induce significant performance delays. For example, if a service access requires PKI validation, then a set of information exchanges must occur between the provider and a credential holder. Each of these exchanges occurs in the context of a dynamically routed, multi-hop packet exchange as described previously. In some enterprise designs, a service calling a service (service chaining) can initiate the same security information exchange. Many real-time applications could not successfully operate given the time required for all these security exchanges.

Performance implications, such as those discussed above, impact the design approach to services. For example, if the overhead to invoke a service across the network is substantial, between getting the data to the service, and consulting security, then it might make sense to have the service do more once it is running. This is the basic discussion of service granularity. Should you have a few bigger services or many little services?

SOA-Based Agility

When we discuss "agility" as it relates to SOA, we are often referring to organizational agility, or the ability to more rapidly adapt a Federal organization's tools to meet their current requirements. *SOA World* magazine explains, "The goal of IT is to put valuable systems in front of our users in a timely manner. Deploying and redeploying in a short time frame is essential to achieving agility." [36] The organization's requirements of IT might change over time for a number of reasons including changes in the mission, changes in organizational reporting requirements, changes in the law, new technologies in the commercial marketplace, attempts to combine diverse data sources to improve the organization's operational picture, and many other reasons. Advocates of SOA assert that, as compared to previous enterprise integration technologies, Web Services offer a more agile manner of interconnecting systems, and improve an organization's ability to retool IT to support change.

> *"The fact of the matter is that the core benefit of SOA is agility. If you have agility, then you have the ability to change the architecture as the business needs changes."*
> *—David Linthicum* [37]

Agility is most effectively discussed as a spectrum, not a true/false boolean value, and it can be assessed as change over a period of time. *The SOA Infrastructure Blog* recently stated, "Efficiency is optimizing for the known. Agility is optimizing for the unknown (i.e., optimizing your future efficiency)" [38] Many of the IT requirements an organization will fulfill in the next decade are not known at this time. Also consider that systems have a habit of living on for much longer than their original creators anticipate. And while we cannot anticipate all the requirements a software system will someday fulfill, or all the data sources the system will someday need to either consume or produce, it is safe to say that working with defined, standards-based, bounded components is easier than monolithic one-off solutions.

An example of agility—Claiming that component-based services offer more organizational agility requires you to compare this approach to a previous method. For example, for the purposes of

comparison, when considering a Web Service as an integration method to exchange data between systems, consider that many of the legacy interfaces between Federal systems are one-off negotiated point-to-point data exchanges. A common exchange method is to send an ASCII file with uniquely formatted data, at a pre-defined mutually agreed non-peak time of day. This legacy point-to-point interface between a data producer system and a data consuming system is labor intensive to code, often requires many staff meetings between both parties to implement, probably does not use standard representations for data, and often is not well documented. If the consuming system should decide to move to another source for the data, the amount of rework is substantial, and the speed of change will not be rapid, and this approach could be fairly tagged as less agile.

In contrast, an organization facing the same data exchange requirements could establish and socialize a data format defined by standard XML. A Web Service that offers that defined XML can be made operational through a web server and run on a nearly 24x7 basis, using SOAP and HTTP. A description of the service can be made available in a service registry for the entire organization to use. Finally, a service-level agreement (SLA), defining organizational commitments to service performance, can be developed and offered to all potential service consumers. With this overall approach a better documented, standards-based interface is created, and the organization as a whole can more quickly make use of this data source.

Probably Less Agile	Probably More Agile
A point-to-point one-off negotiated interface between two specific systems	A general standards-based interface for a community
A user-formatted ASCII data file	XML formatted data with a schema
A custom data exchange designed in the 1980's by staff who have retired	A Web Service standards-based function call
A data exchange understood by two programmers at a time	A data exchange used by 50 organizations with published documentation in a searchable registry
A custom data file exchanged at 1:00 a.m. when computer usage is low	A function available 24x7 on scaled redundant servers

Agility in an SOA context is enhanced by the following characteristics:

- **Architectural commonality among services:** This is best enabled by having a common set of enterprise-defined standards within which to offer services as described above. The worst case scenario requires the caller of a service function to have to call each provider and negotiate a one-off agreement or technical explanation when trying to invoke a service.
- **Ability to clearly define a service interface:** Being able to define the inputs, outputs, and expected behavior and performance of a service is crucial to helping consumer technical staff rapidly invoke a service.
- **Ability to find a service:** Services live on URI endpoints on the IP network. It is inevitable that during the lifetime of a service these endpoints will change. A common method for sharing information on offered services is a service registry. The community of consumers will require some common means of sharing service information.

The larger promise of an enterprise SOA is that once a sufficient quantity of legacy-wrapped components exists, and is accessible on the IP wide area network (WAN), the components can be reconnected more rapidly to solve new problems. *SOA World* magazine stated, "Marketing bologna aside, agility is directly related to the time and effort required to create new functions or to modify existing functions—and then to re-release those functions to the customers." [39] Well-defined SOA components allow programmers to more rapidly assemble components, as compared to one-off interfaces of the past. Russ Abbott writes, "We tend to build systems hierarchically. We formulate a top-level design that meets top-level requirements and then determine what components we need to implement it. We then decide how to build the components in terms of sub-components, etc. This approach doesn't take advantage of existing products and services **except when we use standard parts**—and we do that too rarely." [40]

Federal organizational agility will have a lot to do with the ease with which components can be found and recombined over the next decade. Dion Hinchcliffe blogs, "An important reason why the Web is now the world's biggest and most important computing platform is that people providing software over the Internet are starting to understand the *law of unintended uses*. Great websites no longer limit themselves to just the user interface they provide. They also open up their functionality and data to anyone who wants to use their services as their own. This allows people to reuse, and re-reuse a thousand times over, another service's functionality in their own software for whatever reasons they want, in ways that couldn't be predicted. The future of software is going to be combining the services in the global service landscape into new, innovative applications."[41]

Who is the key beneficiary of SOA? Do individual legacy software projects benefit, or does the enterprise as a whole benefit? Are the interests of the software Project Leader and the CIO the same?

Reaping the Benefits of SOA

A historic analogy— Interstate 95 (I-95), a 1,927-mile highway on the East Coast of the United States, was established by the Eisenhower administration with the Federal Highway Act of 1956 as a key piece of our national infrastructure.[42] The highway, and its considerable acquisition and construction expense, had two central purposes. First, the highway was to enable greater commerce, supporting the more efficient exchange of goods. Second, it supported the nation's defense by more efficiently allowing the movement of troops and their supporting equipment and supplies during the early Cold War. The parallel road, Route 1, which was at several points a single lane road and lined with small towns, was an alternative route at the time. Analyses during the late 1990s estimated that for every dollar spent on I-95, seven dollars have been returned to the general economy, in addition to the improved national defense characteristics that were provided. In retrospect I-95 seems to have been a good investment.

However, if in 1950 we took the approach of asking any of the small 4,000-person towns along Route 1, would they pay for a five- to ten-lane highway and an off ramp to their town, most would find their local town budgets orders of magnitude too small for such a project, and many might not even want to attempt it, as nearby Route 1 was already sufficient and in place. The interests of the "enterprise" and of the local towns did not necessarily align.

We can now more clearly estimate the economic benefits that many of these towns have accumulated since 1950 as a result of this large infrastructure expenditure. And we can also see the enabling effects of a more efficient exchange of goods to the larger economy. **Infrastructure spending enabled exchange** on a larger scale with less "friction." Analogously, we expect that IT infrastructure spending enables the agile exchange of information in an SOA.

Similarly, efforts that benefit the CIO's enterprise, and look good to the senior leadership team of an organization, do not necessarily benefit the small software projects in an agency. Transitioning a legacy application to include a set of Web Services, and putting the services in place with a robust infrastructure of redundant 24x7 reliable servers with full support as well as a service discovery mechanism, is an expensive task, hopefully enabled by enterprise-level infrastructure efforts. For example, SOA with contemporary Web Service implementations is directly enabled by the quality of the underlying IP network, and the server redundancy of the Web Service offerings. Real Web Service implementations often require multiple tiers of servers, such as Web Servers, logic servers, and databases, to all operate reliably to fulfill a mission.

If as a result of creating a good service, an individual project then picks up many more consumers than it

had previously, then clearly the day-to-day demands on the project's IT infrastructure increase. The common result of service success is higher local operational costs. At the enterprise level, this is a benefit because it means that more customers are reusing the same shared services instead of rebuilding them. The leadership team should be pleased. But the individual Federal software system project leader is likely to be on a fixed budget that may have been established well before the dynamic nature of the SOA producer/consumer model was noticed. And while a commercial corporation can be more nimble in responding to rapid usage changes, Federal programs can be less quick to measure and respond to such changes.

In summary, the local perspective of individual legacy projects will not justify an enterprise SOA effort, but this should not be allowed to stop the enterprise SOA from occurring. The SOA benefits accrue largely at the enterprise's level in cost avoidance through reuse, and increased data exchange and agility. Consequently, a corresponding investment is required at the enterprise level, where the benefit is found.

Enterprise standards compliance—Another interesting enterprise characteristic of SOA and I-95 is that both rely on standards compliance. Federal funding is the chief motivator for compliance with Federal standards for highways. "The American Association of State Highway and Transportation Officials (AASHTO) has defined a set of standards that all new Interstates must meet unless a waiver from the Federal Highway Administration (FHWA) is obtained. These standards have become more strict over the years…. The dominant role of the Federal government in road finance has enabled it to achieve legislative goals that fall outside its power to regulate interstate commerce. By threatening to withhold highway funds, the Federal government has been able to stimulate state legislatures to pass a variety of laws."[43]

Standards compliance has obvious benefits for a highway system and a set of enterprise services. As discussed above, Web Services can be defined by a set of industry standards that form a common framework for implementation. One of the chief concerns in this area are the standards and mechanisms established for security. Consequently, establishing the standards and a governance mechanism is a key part of implementing an enterprise SOA.

Agility is engendered by architecture commonality, which eases reuse across a large organization.

SOA market models—Senior leadership teams in large organizations often find themselves considering the philosophical underpinnings and organizational dynamics of IT portfolio management. In this final analogy, the SOA effort is discussed as an example of a market economy or a command economy. In practice, some mixture of the two approaches is most often needed. For example, individual service providers, who have the deepest understanding of their customers and data sources, must be allowed to offer the services that make sense from their market-oriented point of view. They can offer services that match their customer's needs, and they can enjoy the success of correctly matching customer requirements or endure the consequences of forecasting incorrectly. The enterprise CIO must also ensure from a command point of view that the enterprise has a reasonable IT portfolio, gaps in services capabilities are being filled somewhere in the organization, and architectural commonality is being preserved. Successful SOA efforts will support innovation by the participants, while also ensuring a comprehensive set of reused services and standards compliance. The challenge is finding the balance.

> *Natural ecologies and market economies are both examples of what we call innovative environments. The fundamental principle is that new things are built on top of existing things.*
>
> *—Russ Abbott*[45]

Conclusion

SOA offers Federal leadership teams a means to effectively leverage decades of IT investment, while providing a growth path for new capabilities. Contemporary SOA technologies, such as the Web Services standards, offer valuable new capabilities such as language-neutral integration, yet still require structured engineering processes and well-defined acquisitions, and enterprise portfolio management.

The Science of Computer Programming journal stated, "Executives of large organizations with substantial IT budgets learned the hard way that spending more is not the winning strategy. Some of them realized that after a long string of staggering IT investments plus their challenges, they must start to control their IT portfolios."[44] SOA provides a technical underpinning for structuring portfolios as a collection of discrete services, each with a definable customer base, acquisition strategy, performance levels, and a measurable operational cost.

A key current challenge for many Federal organizations is the structuring of their IT portfolio around a component-based service model and enforcing sufficient standards **within** their own organizational boundaries, which can be quite large. As the span of attempted integration continues to grow, the challenge of the next 10 years will be enabling that integration model to bridge multiple **external** organizations that undoubtedly will be using disparate standards and tools. After the first generation of standards-based service integrations has passed, and portfolios become defined, process driven, and manageable, translation and brokering will be the next set of key cross-enterprise services.

Acronyms

Acronym	Definition
AASHTO	American Association of State Highway and Transportation Officials
ASCII	American Standard Code for Information Interchange
CSS	Cascading Style Sheets
CICS	Customer Information Control System
CIO	Chief Information Officer
CORBA	Common Object Request Broker Architecture
COM	Component Object Model
COTS	Commercial-Off-The-Shelf
CRL	Certificate Revocation List
DCOM	Distributed Component Object Model
DoD	Department of Defense
EAI	Enterprise Application Integration
EJB	Enterprise JavaBeans
FHWA	Federal Highway Administration
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IC	Integrated Circuit
IP	Internet Protocol
IT	Information Technology
JSR	Java Specification Request
LDAP	Lightweight Directory Access Protocol
OCSP	Online Certificate Status Protocol
OMB	Office of Management and Budget
PBSC	Performance-Based Service Contracting
PKI	Public Key Infrastructure
PWS	Performance Work Statement
SAML	Security Assertion Markup Language
SLA	Service-Level Agreement
SLOC	Service Lines of Code
SNMP	Simple Network Management Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
RCR	Relative Cost of Reuse
RCWR	Relative Cost of Writing For Reuse
RSS	Really Simple Syndication
UDDI	Universal Description, Discovery, and Integration
URI	Uniform Resource Identifier
WAN	Wide Area Network
WS*	Web Services Standards
WSDL	Web Services Description Language
WSRP	Web Services for Remote Portlets
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language
XPath	XML Path Language

References

[1] David Linthicum, "Enterprise Application Integration," http://safari.oreilly.com/0201615835 November 12, 1999

[2] Loosely Coupled, David Longworth, "Service reuse unlocks hidden value" http://www.looselycoupled.com/stories/2003/reuse-ca0929.html 29 Sept. 2003

[3] Wikipedia, "Rosetta Stone," http://en.wikipedia.org/wiki/Rosetta_stone 28 March 2008

[4] Peter Fingar et al., "Next Generation Computing: Distributed Objects for Business," SIGS Books & Multimedia, New York, 1996

[5] Internet World Magazine, David Litwack, "Web Services Has the Biggest Hype Machine Behind it of any Technology Today. Here is Why You Should Be Excited Anyway" http://iw.com/magazine.php?inc=060102/06.01.02ebusiness1.html 1 June 2002

[6] GDS InfoCentre, Ronan Bradley, "Agile Infrastructures" http://gdsinternational.com/infocentre/artsum.asp?mag=184&iss=150&art=25901&lang=en 28 March 2008

[7] Intel, "How Chips Are Made", http://www.intel.com/education/makingchips/preparation.htm 28 March 2008

[8] Design World, Electronic Design, C. Panasuk, "Silicon Compilers Make Sweeping Changes in the VLSI," Sept. 20 1984, pp. 67-74.

[9] Jeffrey Poulin, "The ROI of SOA Relative to Traditional Component Reuse," Logic Library, 2006

[10] Programmable Web, "Mashup Dashboard," http://www.programmableweb.com/mashups 28 March 2008

[11] International Genetically Engineered Machine Competition (IGEM), "Registry of Standard Biological Parts," http://parts.mit.edu/registry/index.php/Main_Page 28 March 2008

[12] Department of Defense, Ada Joint Program Office, "Ada 95 Quality and Style Guide," http://www.adaic.com/docs/95style/html/sec_8/ 28 March 2008

[13] Boehm, B. W., et al. "An environment for improving software productivity." Computer, June 1984.

[14] Ada Joint Program Office

[15] Will. Traez "Software Reuse: Motivators and lnhibitors." Proceedings of COMPCON S'87, 1987.

[16] Progress Actional, "Web Services and Reuse" http://www.actional.com/resources/whitepapers/SOA-Worst-Practices-Vol-I/Web-Services-Reuse.html 28 March 2008

[17] Eric Roch, "SOA Service Reuse" http://blogs.ittoolbox.com/eai/business/archives/SOA-Service-Reuse—14699 28 March 2008

[18] ZDnet, Joe McKendrick, "Pouring cold water on SOA 'reuse' mantra" http://blogs.zdnet.com/service-oriented/?p=699 30 August 2006

[19] Eric Roch, "SOA Service Reuse" http://blogs.ittoolbox.com/eai/business/archives/SOA-Service-Reuse—14699 28 March 2008

[20] Progress Actional, "Web Services and Reuse" http://www.actional.com/resources/whitepapers/SOA-Worst-Practices-Vol-I/Web-Services-Reuse.html 28 March 2008

[21] Barry Boehm, DARPA Workshop, "Software Reuse Economics" http://sunset.usc.edu/GSAW/gsaw99/pdf-presentations/breakout-2/boehm.pdf 14 January 1997

[22] Jeffery Poulin, "The ROI of SOA Relative to Traditional Component Reuse," Logic Library, 2006

[23] Office of Management and Budget, Office of Federal Procurement, "Policy Performance-Based Service Acquisition," http://www.whitehouse.gov/omb/procurement/0703pbsat.pdf July 2003

[24] Ibid.

[25] Wikipedia, "Enterprise Application Integration,"
http://en.wikipedia.org/wiki/Enterprise_application_integration 28 March 2008

[26] GDS InfoCentre, Ronan Bradley, "Agile Infrastructures,"
http://gdsinternational.com/infocentre/artsum.asp?mag=184&iss=150&art=25901&lang=en

[27] Computer World, Anne Thomas Manes, "When to Use Web Services,"
http://www.computerworld.com/printthis/2004/0,4814,94886,00.html 16 August 2004

[28] Ebiz, Dr. Chris Harding, "Achieving Business Agility through Model-Driven SOA"
http://www.ebizq.net/topics/soa/features/6639.html 29 January 2006

[29] DM Review, Integration Consortium, "Integration Everywhere – How SOA is Altering the Direction of EAI - Thoughts from the EAI Consortium"
http://www.dmreview.com/news/8229-1.html 4 March 2004

[30] Note that SOAP is no longer considered an acronym. For more information, see
http://www.w3.org/TR/2003/REC-soap12-part0-20030624/#L1153

[31] InfoQ, Michele Leroux Bustamante, "Making Sense of all these Crazy Web Service Standards,"
http://www.infoq.com/articles/ws-standards-wcf-bustamante 16 May 2007

[32] Computer World, Mary Brandel, "Message Received? Companies that require highly reliable Web services are building in their own guarantees"
http://www.computerworld.com/action/article.do?command=viewArticleTOC&specialReportId=620&articleId=95221

[33] Computer World, Anne Thomas Manes, "When to Use Web Services,"
http://www.computerworld.com/printthis/2004/0,4814,94886,00.html 16 August 2004

[34] PushToTest, Frank Cohen, "Discover SOAP encoding's impact on Web service performance"
http://www.ibm.com/developerworks/webservices/library/ws-soapenc/ 1 March 2003

[35] Computer World, Anne Thomas Manes, "When to Use Web Services,"
http://www.computerworld.com/printthis/2004/0,4814,94886,00.html 16 August 2004

[36] SOA World Magazine, Jeff Schneider, "SOA Web Services: Does Your SOA Achieve Agility?"
http://webservices.sys-con.com/read/143900_2.htm 10 November 2005

[37] David Linthicum, "Real World SOA"
http://weblog.infoworld.com/realworldsoa/archives/2007/11/using_it_backlo.html?source=rss 28 March, 2008

[38] SOA Infrastructure Blog, Dan Foody, "So what is SOA agility anyway?"
http://blogs.progress.com/soa_infrastructure/2007/08/what-is-agility.html 29 August 2007

[39] SOA World Magazine, Jeff Schneider, "SOA Web Services: Does Your SOA Achieve Agility?"
http://webservices.sys-con.com/read/143900.htm 10 November 2005

[40] Russ Abbott, "Putting Complex Systems to Work"
http://64.233.169.104/search?q=cache:wunzA2V5_l8J:cs.calstatela.edu/wiki/images/7/7e/Abbott.doc 28 March 2008

[41] Social Computing Magazine, Dion Hinchcliffe, "Social Aggregators Emerge To Manage Digital Lifestyles"
http://web2.socialcomputingmagazine.com/ 28 March 2008

[42] Wikipedia, "Interstate Highway System"
http://en.wikipedia.org/wiki/Interstate_Highway_System 28 March 2008

[43] Ibid.

[44] C. Verhoef, "Quantitative IT Portfolio Management", Science of Computer Programming, Volume 45 Issue 1
http://www.cs.vu.nl/~x/ipm/ipm.pdf 28 March 2008

[45] Russ Abbott, "Putting Complex Systems to Work"
http://64.233.169.104/search?q=cache:wunzA2V5_l8J:cs.calstatela.edu/wiki/images/7/7e/Abbott.doc 28 March 2008

Additional Photo Credits

Inset photos used under license from iStockPhoto.com

Except the following:

Page 4: Rosetta Stone, Public Domain - Wikipedia

Page 11: Black Box photo – used with permission of Performance Trends Inc.
http://www.performancetrends.com/

Page 18: President Eisenhower with Clay Commission photo – courtesy of the Eisenhower Presidential Library and the National Park Service – used with permission
http://www.eisenhower.utexas.edu/

www.mitre.org

Approved for Public Release
Distribution Unlimited
Case Number: 08-1696
Document Number: MTR080331

MITRE