Understanding Service-Oriented Architecture

01/14/2009 • 22 minutes to read

In this article

David Sprott and Lawrence Wilkes
CBDI Forum

January 2004

Summary: Gives a concise explanation of service-oriented architecture, what it is, and how it affects what architects, CIOs, project managers, business analysts, and lead developers do. (13 printed pages)

Contents

Introduction

It seems probable that eventually most software capabilities will be delivered and consumed as services. Of course they may be implemented as tightly coupled systems, but the point of usage—to the portal, to the device, to another endpoint, and so on, will use a service-based interface. We have seen the comment that architects and designers need to be cautious to avoid everything becoming a service. We think this is incorrect and muddled thinking. It might be valid right now given the maturity of Web Service protocols and technology to question whether everything is implemented using Web services, but that doesn't detract from the need to design everything from a service perspective. The service is the major construct for publishing and should be used at the point of each significant interface. service-oriented architecture allows us to manage the usage (delivery, acquisition, consumption, and so on) in terms of, and in sets of, related services. This will have big implications for how we manage the software life cycle—right from specification of requirements as services, design of services, acquisition and outsourcing as services, asset management of services, and so on.

Over time, the level of abstraction at which functionality is specified, published and or consumed has gradually become higher and higher. We have progressed from modules, to objects, to components, and now to services. However in many respects the naming of SOA is unfortunate. Whilst SOA is of course about architecture, it is impossible to constrain the discussion to architecture, because matters such as business design and the delivery process are also important considerations. A more useful nomenclature might be Service Orientation (or SO). There are actually a number of parallels with object orientation (or OO) and component-based development (CBD):

- Like objects and components, services represent natural building blocks that allow us to organize capabilities in ways that are familiar to us.
- Similarly to objects and components, a service is a fundamental building block that

    1. Combines information and behaviour.
    2. Hides the internal workings from outside intrusion.
    3. Presents a relatively simple interface to the rest of the organism.

- Where objects use abstract data types and data abstraction, services can provide a similar level of adaptability through aspect or context orientation.
- Where objects and components can be organized in class or service hierarchies with inherited behaviour, services can be published and consumed singly or as hierarchies and or collaborations.

For many organizations, the logical starting place for investigating service-oriented architecture is the consideration of Web services. However Web services are not inherently service oriented. A Web service merely exposes a capability that conforms to

Web services protocols. In this article we will identify the characteristics of a well formed service, and provide guidance for architects and designers on how to deliver service oriented applications.

Principles and Definitions

Looking around we see the term or acronym SOA becoming widely used, but there's not a lot of precision in the way that it's used. The World Wide Web Consortium (W3C) for example refers to SOA as 'A set of components which can be invoked, and whose interface descriptions can be published and discovered'. We see similar definitions being used elsewhere; it's a very technical perspective in which architecture is considered a technical implementation. This is odd, because the term architecture is more generally used to describe a style or set of practices—for example the style in which something is designed and constructed, for example Georgian buildings, Art Nouveau decoration or a garden by Sir Edwin Lutyens and Gertrude Jekyll .

CBDI believes a wider definition of service-oriented architecture is required. In order to reach this definition, let's start with some existing definitions, and compare some W3C offerings with CBDI recommendations. We'll begin by looking at definitions of basic Service concepts.

Service

- A Component capable of performing a task. A WSDL service: A collection of end points (W3C).
- A type of capability described using WSDL (CBDI).

A Service Definition

- A vehicle by which a consumer's need or want is satisfied according to a negotiated contract (implied or explicit) which includes Service Agreement, Function Offered and so on (CBDI).

A Service Fulfillment

- An instance of a capability execution (CBDI).

Web service

- A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a format that machines can process (specifically WSDL). Other systems interact with the Web service in a

manner prescribed by its description using SOAP messages, typically conveyed using HTTP with XML serialization in conjunction with other Web-related standards (W3C).
- A programmatic interface to a capability that is in conformance with WSnn protocols (CBDI).

From these definitions, it will be clear that the W3C have adopted a somewhat narrower approach to defining services and other related artefacts than CBDI. CBDI differs slightly insofar as not all Services are Components, nor do they all perform a task. Also CBDI recommends it is useful to manage the type, definition and fulfilment as separate items. However it is in the definition of SOA that CBDI really parts company with the W3C.

Service-Oriented Architecture:

- A set of components which can be invoked, and whose interface descriptions can be published and discovered (W3C).

CBDI rejects this definition on two counts: First the components (or implementations) will often not be a set. Second the W3C definition of architecture only considers the implemented and deployed components, rather than the science, art or practice of building the architecture. CBDI recommends SOA is more usefully defined as:

The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards-based form of interface. (CBDI)

CBDI defines SOA as a style resulting from the use of particular policies, practices and frameworks that deliver services that conform to certain norms. Examples include certain granularity, independence from the implementation, and standards compliance. What these definitions highlight is that any form of service can be exposed with a Web services interface. However higher order qualities such as reusability and independence from implementation, will only be achieved by employing some science in a design and building process that is explicitly directed at incremental objectives beyond the basic interoperability enabled by use of Web services.

SOA Basics

It's would be easy to conclude that the move to Service Orientation really commenced with Web services—about three years ago. However, Web services were merely a step along a much longer road. The notion of a service is an integral part of component thinking, and it is clear that distributed architectures were early attempts to implement service-oriented architecture. What's important to recognize is that Web services are part

of the wider picture that is SOA. The Web service is the programmatic interface to a capability that is in conformance with WSnn protocols. So Web services provide us with certain architectural characteristics and benefits—specifically platform independence, loose coupling, self description, and discovery—and they can enable a formal separation between the provider and consumer because of the formality of the interface.

Service is the important concept. Web Services are the set of protocols by which Services can be published, discovered and used in a technology neutral, standard form.

In fact Web services are not a mandatory component of a SOA, although increasingly they will become so. SOA is potentially much wider in its scope than simply defining service implementation, addressing the quality of the service from the perspective of the provider and the consumer. You can draw a parallel with CBD and component technologies. COM and UML component packaging address components from the technology perspective, but CBD, or indeed Component-Based Software Engineering (CBSE), is the discipline by which you ensure you are building components that are aligned with the business. In the same way, Web services are purely the implementation. SOA is the approach, not just the service equivalent of a UML component packaging diagram.

Many of these SOA characteristics were illustrated in a recent CBDI report, which compared Web services published by two dotcom companies as alternatives to their normal browser-based access, enabling users to incorporate the functionality offered into their own applications. In one case it was immediately obvious that the Web services were meaningful business services—for example enabling the Service Consumer to retrieve prices, generate lists, or add an item to the shopping cart.

In contrast the other organization's services are quite different. It implemented a general purpose API, which simply provides Create, Read, Update, and Delete (CRUD) access to their database through Web services. While there is nothing at all wrong with this implementation, it requires that users understand the underlying model and comply with the business rules to ensure that your data integrity is protected. The WSDL tells you nothing about the business or the entities. This is an example of Web services without SOA.

SOA is not just an architecture of services seen from a technology perspective, but the policies, practices, and frameworks by which we ensure the right services are provided and consumed.

So what we need is a framework for understanding what constitutes a good service. If, as we have seen in the previous example, we have varying levels of usefulness, we need some Principles of Service Orientation that allow us to set policies, benchmarks and so on.

We can discern two obvious sets here:

- Interface related principles—Technology neutrality, standardization and consumability.
- Design principles—These are more about achieving quality services, meeting real business needs, and making services easy to use, inherently adaptable, and easy to manage.

Interestingly the second set might have been addressed to some extent by organizations that have established mature component architectures. However it's certainly our experience that most organizations have found this level of discipline hard to justify. While high quality components have been created perhaps for certain core applications where there is a clear case for widespread sharing and reuse, more generally it has been hard to incur what has been perceived as an investment cost with a short term return on investment.

However when the same principles are applied to services, there is now much greater awareness of the requirements, and frankly business and IT management have undergone a steep learning curve to better understand the cost and benefits of IT systems that are not designed for purpose. Here we have to be clear—not all services need all of these characteristics; however it is important that if a service is to be used by multiple consumers, (as is typically the case when a SOA is required), the specification needs to be generalized, the service needs to be abstracted from the implementation (as in the earlier dotcom case study), and developers of consumer applications shouldn't need to know about the underlying model and rules. The specification of obligations that client applications must meet needs to be formally defined and precise and the service must be offered at a relevant level of granularity that combines appropriate flexibility with ease of assembly into the business process.

Table 1 shows principles of good service design that are enabled by characteristics of either Web services or SOA.

Table 1. Web services and SOA

Enabled by Web services	Technology neutral	Endpoint platform independence.
	Standardized	Standards-based protocols.
	Consumable	Enabling automated discovery and usage.
Enabled by SOA	Reusable	Use of Service, not reuse by copying of code/implementation.
	Abstracted	Service is abstracted from the implementation.
	Published	Precise, published specification functionality of service

		interface, not implementation.
	Formal	Formal contract between endpoints places obligations on provider and consumer.
	Relevant	Functionality presented at a granularity recognized by the user as a meaningful service.

If the principles summarized in Table 1 are complied with, we get some interesting benefits:

- There is real synchronization between the business and IT implementation perspective. For many years, business people haven't really understood the IT architecture. With well designed services we can radically improve communications with the business, and indeed move beyond alignment and seriously consider convergence of business and IT processes.
- A well formed service provides us with a unit of management that relates to business usage. Enforced separation of the service provision provides us with basis for understanding the life cycle costs of a service and how it is used in the business.
- When the service is abstracted from the implementation it is possible to consider various alternative options for delivery and collaboration models. No one expects that, at any stage in the foreseeable future, core enterprise applications will be acquired purely by assembling services from multiple sources. However it is entirely realistic to assume that certain services will be acquired from external sources because it is more appropriate to acquire them. For example authentication services, a good example of third party commodity services that can deliver a superior service because of specialization, and the benefits of using a trusted external agency to improve authentication.

Process Matters

As indicated earlier, CBDI advises that good SOA is all about style—policy, practice and frameworks. This makes process matters an essential consideration.

Whilst some of the benefits of services might have been achieved by some organizations using components, there are relatively few organizations that rigorously enforce the separation of provision and consumption throughout the process. This gets easier with services because of the formality of the interface protocols, but we need to recognize that this separation needs managing. For example it's all too easy to separate the build processes of the service and the consumer, but if the consumer is being developed by the same team as the service then it's all too easy to test the services in a manner that reflects understanding of the underlying implementation.

With SOA it is critical to implement processes that ensure that there are at least two different and separate processes—for provider and consumer.

However, current user requirements for seamless end-to-end business processes, a key driver for using Web Services, mean that there will often be clear separation between the providing and consumer organizations, and potentially many to many relationships where each participant has different objectives but nevertheless all need to use the same service. Our recommendation is that development organizations behave like this, even when both the providing and consuming processes are in-house, to ensure they are properly designing services that accommodate future needs

For the consumer, the process must be organized such that only the service interface matters, and there must be no dependence upon knowledge of the service implementation. If this can be achieved, considerable benefits of flexibility accrue because the service designers cannot make any assumptions about consumer behaviours. They have to provide formal specifications and contracts within the bounds of which consumers can use the service in whatever way they see fit. Consumer developers only need to know where the service is, what it does, how they can use it. The interface is really the only thing of consequence to the consumer as this defines how the service can be interacted with.

Similarly, whilst the provider has a very different set of concerns, it needs to develop and deliver a service that can be used by the Service Consumer in a completely separate process. The focus of attention for the provider is therefore again the interface—the description and the contract.

Another way of looking at this is to think about the nature of the collaboration between provider and consumer. At first sight you may think that there is a clear divide between implementation and provisioning, owned by the provider, and consumption, owned by the consumer. However if we look at these top level processes from the perspective of collaborations, then we see a very different picture.

What we have is a significant number of process areas where (depending on the nature of the service) there is deep collaboration between provider and consumer. Potentially we have a major reengineering of the software delivery process. Although we have two primary parties to the service-based process, we conclude there are three major process areas which we need to manage. Of course these decompose, but it seems to us that the following are the primary top level processes.

- The process of delivering the service implementation.
  - 'Traditional' Development
  - Programming
  - Web Services automated by tools

- The provisioning of the service—the life cycle of the service as a reusable artefact.
  - Commercial Orientation
  - Internal and External View
  - Service Level Management
- The consumption process.
  - Business Process Driven
  - Service Consumer could be internal or external
  - Solution assembly from Services, not code
  - Increasingly graphical, declarative development approach
  - Could be undertaken by business analyst or knowledge worker

The advantage of taking this view is that the collaborative aspects of the process are primarily contained in the provisioning process area. And the provisioning area is incredibly important because the nature of the agreement has a major influence on the process requirements. There are perhaps two major patterns for designing consumer/provider collaborations:

- Negotiated—Consumer and Provider jointly agree service When new services are developed though, there is an opportunity for both provider and consumer to agree what and how the services should work. In industries where there are many participants all dealing with each other, and where services are common to many providers, it is essential that the industry considers standardizing those services. Examples include:
  - Early adopters
  - New Services
  - Close partners
  - Industry initiative—forming standards
  - Internal use
- Instantiated—This is it. Take it or leave it One party in the collaborative scenario might simply dictate the services that must be used. Sometimes the service will already exist. You just choose to use it, or not. Examples include:
  - Dominant partner
  - Provider led—Use this service or we can't do business
  - Consumer led—Provide this service or we can't do business
  - Industry initiative—standards compliance
  - Existing system/interface

Architectures

This process view that we have examined at is a prerequisite to thinking about the type of architecture required and the horizons of interest, responsibility and integrity. For SOA

there are three important architectural perspectives as shown in Figure 1.

- The Application Architecture. This is the business facing solution which consumes services from one or more providers and integrates them into the business processes.
- The Service Architecture. This provides a bridge between the implementations and the consuming applications, creating a logical view of sets of services which are available for use, invoked by a common interface and management architecture.
- The Component Architecture. This describes the various environments supporting the implemented applications, the business objects and their implementations.

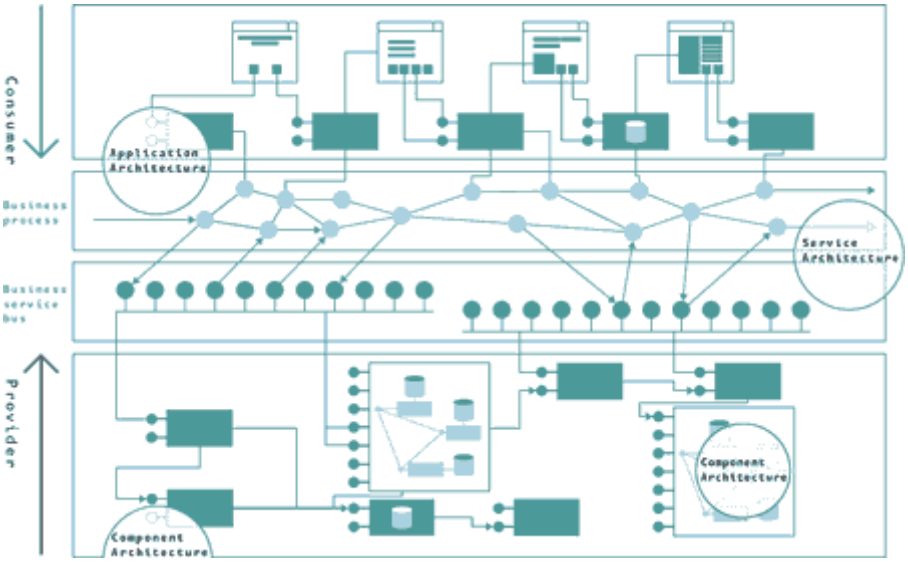

Figure 1. Three Architectural Perspectives

These architectures can be viewed from either the consumer or provider perspective. Key to the architecture is that the consumer of a service should not be interested in the implementation detail of the service—just the service provided. The implementation architecture could vary from provider to provider yet still deliver the same service. Similarly the provider should not be interested in the application that the service is consumed in. New unforeseen applications will reuse the same set of services.

The consumer is focused on their application architecture, the services used, but not the detail of the component architecture. They are interested at some level of detail in the general business objects that are of mutual interest, for example provider and consumer need to share a view of what an order is. But the consumer does not need to know how the order component and database are implemented.

Similarly, the provider is focused on the component architecture, the service architecture, but not on the application architecture Again, they both need to understand certain information about the basic applications, for example to be able to set any sequencing

rules and pre and post conditions. But the provider is not interested in every detail of the consuming application.

The Service Architecture

At the core of the SOA is the need to be able to manage services as first order deliverables. It is the service that we have constantly emphasized that is the key to communication between the provider and consumer. So we need a Service Architecture that ensures that services don't get reduced to the status of interfaces, rather they have an identity of their own, and can be managed individually and in sets.

CBDI developed the concept of the Business Service Bus (BSB) precisely to meet this need. The BSB is a logical view of the available and used services for a particular business domain, such as Human Resources or Logistics. It helps us answer questions such as:

- What service do I need?
- What services are available to me?
- What services will operate together? (common semantics, business rules)
- What substitute services are available?
- What are the dependencies between services and versions of services?

Rather than leaving developers to discover individual services and put them into context, the Business Service Bus is instead their starting point that guides them to a coherent set that has been assembled for their domain.

The purpose of the BSB is so that common specifications, policies, etc can be made at the bus level, rather than for each individual service. For example, services on a bus should all follow the same semantic standards, adhere to the same security policy, and all point to the same global model of the domain. It also facilitates the implementation of a number of common, lower-level business infrastructure services that can be aggregated into other higher level business services on the same bus (for example, they could all use the same product code validation service). Each business domain develops a vocabulary and a business model of both process and object.

A key question for the Service Architecture is 'What is the scope of the service that is published to the Business Service Bus?' A simplistic answer is 'At a business level of abstraction'. However this answer is open to interpretation—better to have some heuristics that ensure that the service is the lowest common denominator that meets the criteria of business, and is consumer oriented, agreed, and meaningful to the business. The key point here is that there is a process of aggregation and collaboration that should probably happen separately from the implementing component as illustrated in Figure 2. By making it separate, there is a level of flexibility that allows the exposed service(s) to be adjusted without modifying the underlying components. In principle, the level of

abstraction will be developed such that services are at a level that is relevant and appropriate to the consumer. The level might be one or all of the following:

- Business Services
- Service Consumer Oriented
- Agreed by both Provider and Consumer
- Combine low-level implementation-based services into something meaningful to business
- Coarser Grained
- Suitable for External Use
- Conforms to pre-existing connection design

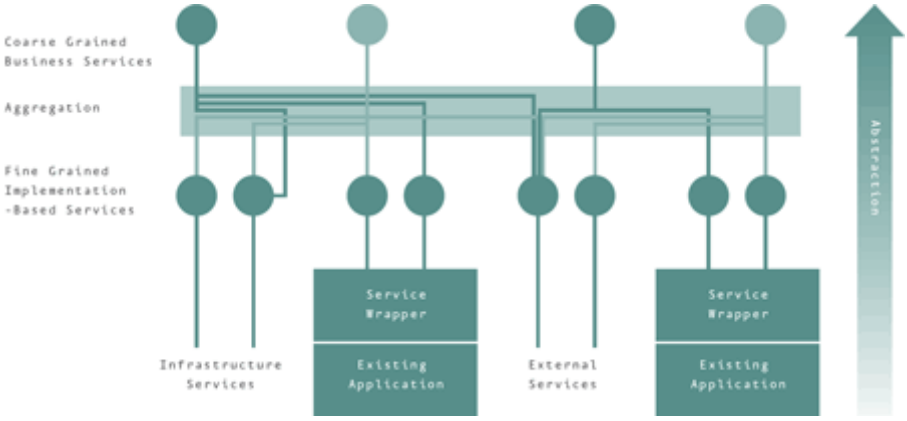

Figure 2. Levels of Abstraction

The SOA Platform

The key to separation is to define a virtual platform that is equally relevant to a number of real platforms. The objective of the virtual platform is to enable the separation of services from the implementation to be as complete as possible and allow components built on various implementation platforms to offer services which have no implementation dependency.

The virtual SOA platform comprises a blueprint which covers the development and implementation platforms. The blueprint provides guidance on the development and implementation of applications to ensure that the published services conform to the same set of structural principles that are relevant to the management and consumer view of the services.

When a number of different applications can all share the same structure, and where the relationships between the parts of the structure are the same, then we have what might be called a common architectural style. The style may be implemented in various ways; it might be a common technical environment, a set of policies, frameworks or practices. Example platform components of a virtual platform include:

- Host environment
- Consumer environment
- Middleware
- Integration and assembly environment
- Development environment
- Asset management
- Publishing & Discovery
- Service level management
- Security infrastructure
- Monitoring & measurement
- Diagnostics & failure
- Consumer/Subscriber management
- Web service protocols
- Identity management
- Certification
- Deployment & Versioning

The Enterprise SOA

The optimum implementation architecture for SOA is a component-based architecture. Many will be familiar with the concepts of process and entity component, and will understand the inherent stability and flexibility of this component architecture, which provide a one to one mapping between business entities and component implementations. Enterprise SOA (ESOA) brings the two main threads—Web services and CBD (or CBSE)—together. The result is an enterprise SOA that applies to both Web services made available externally and also to core business component services built or specified for internal use. It is beyond the scope of this article to explore ESOA in more depth. For more on this topic there is a five part CBDI Report Series on Enterprise SOA.

Summary

The goal for a SOA is a world wide mesh of collaborating services, which are published and available for invocation on the Service Bus. Adopting SOA is essential to deliver the business agility and IT flexibility promised by Web Services. These benefits are delivered not by just viewing service architecture from a technology perspective and the adoption of Web Service protocols, but require the creation of a Service Oriented Environment that is based on the following key principals we have articulated in this article;

- Service is the important concept. Web Services are the set of protocols by which Services can be published, discovered and used in a technology neutral, standard form.

- SOA is not just an architecture of services seen from a technology perspective, but the policies, practices, and frameworks by which we ensure the right services are provided and consumed.
- With SOA it is critical to implement processes that ensure that there are at least two different and separate processes—for provider and consumer.
- Rather than leaving developers to discover individual services and put them into context, the Business Service Bus is instead their starting point that guides them to a coherent set that has been assembled for their domain.

For further guidance on planning and managing the transition to Web Services and SOA, CBDI are providing the 'Web Services Roadmap', a set of resources that are freely available at http://roadmap.cbdiforum.com/.

About the Authors

David Sprott, CEO and Principal Analyst, CBDI Forum is a software industry veteran, a well-known commentator and analyst specializing in advanced application delivery. Since 1997 he has founded and led CBDI, an independent analyst firm and thinktank, providing a focus for the industry on best practice in business software creation, reuse and management. David can be reached at david.sprott@cbdiforum.com.

Lawrence Wilkes, Technical Director and Principal Analyst, CBDI Forum is a frequent speaker, lecturer and writer on Web services, Service-Oriented Architecture, Component-Based Development and Application Integration approaches. Lawrence has over 25 years experience in IT working both for end user organizations in various industries, as well as for software vendors and as a consultant. Lawrence can be reached at lawrence.wilkes@cbdiforum.com.

This article was published in the Architecture Journal, a print and online publication produced by Microsoft. For more articles from this publication, please visit the Architecture Journal website.