Getting Started with Service-Oriented Architecture (SOA) Terminology

Grace Lewis

September 2010

This white paper presents basic terminology related to Service-Oriented Architecture (SOA).[1] The goal of the paper is to establish a baseline of terms for service-oriented systems.

Service-Oriented Architecture and Service-Oriented Systems

Service-Oriented Architecture (SOA) is a way of designing, developing, deploying, and managing systems, in which

- Services provide reusable business functionality via well-defined interfaces.
- There is a clear separation between service interface and service implementation.
- Service consumers are built using functionality from available services.
- An SOA infrastructure enables discovery, composition, and invocation of services.
- Protocols are predominantly, but not exclusively, message-based document exchanges.

Service-Oriented Architecture (SOA) is a way of designing, developing, deploying, and managing systems … it is neither a system architecture nor a complete system.

From a more technical point of view, SOA is an architectural style or design paradigm; it is neither a system architecture nor a complete system. Systems that are built based on the SOA characteristics listed above are called service-oriented systems. A high-level view of a service-oriented system is presented in Figure 1.

Services

Services are *reusable components* that represent business or operational tasks, such as customer lookup, credit card validation, weather lookup, or line-of-sight calculation. Reusable is a key element of this definition because it is what enables the creation of new business and operational processes based on these services. Services expose their capabilities via well-defined, standard service interfaces. In a service-oriented environment, service interface definitions are available in some form of *service registry*.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

[1] The basic terminology has been extracted from the SEI course Service-Oriented Architecture: Best Practices for Successful Adoption. For more information about the course, visit http://www.sei.cmu.edu/training/p81.cfm.

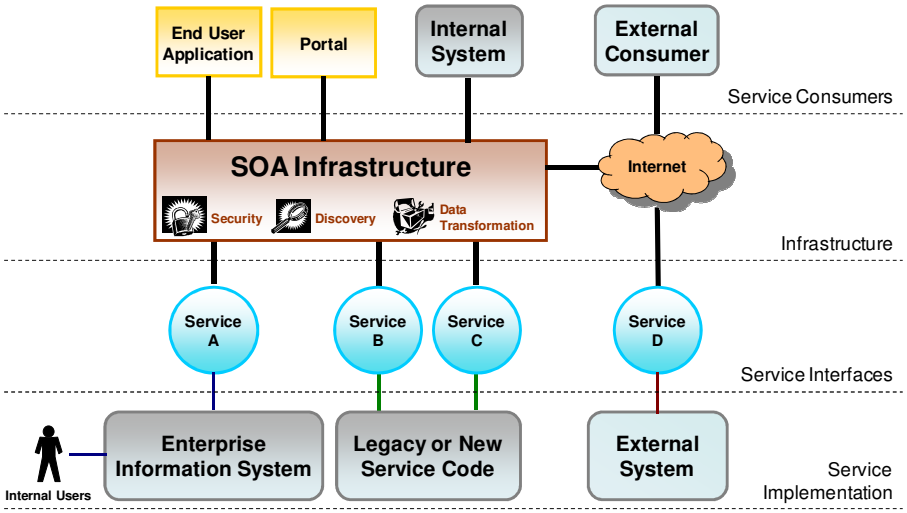

Figure 1: High-Level View of a Service-Oriented System

The *service implementation* corresponds to the actual system capabilities that implement the service interface. Service implementations typically reside in enterprise information systems, such as ERP (Enterprise Resource Planning) systems; in legacy or new service code built specifically to provide service capabilities; or in external systems. This separation between service interface and service implementation is what allows platform independence—*service consumers* access the services via standardized service interfaces that hide the complexity and diversity of service implementations from consumers. The organization that hosts a service implementation is referred to as the *service provider*.

The elements of a service-oriented system are services, service consumers, and the SOA infrastructure that binds service consumers to services.

SOA Infrastructure

An SOA infrastructure is the set of technologies that bind service consumers to services through an agreed-upon communication model, such as one based on Web Services, message-oriented middleware (MOM), publish/subscribe, or Common Object Request Broker Architecture (CORBA). In addition, SOA infrastructures typically host infrastructure services that can be used by service providers and service consumers to perform common tasks or satisfy quality attribute requirements of the environment. Typical infrastructure services include security, discovery, and data transformation.

Service Consumers

Service consumers are the clients for the functionality provided by the services. Examples of service consumers are end-user applications, internal systems, external systems, and composite services. Consumers programmatically bind to services (i.e., there is a piece of code running on the consumer side that invokes a piece of code running on the provider side that corresponds to the service interface).

Three Basic Operations to Support Service-Oriented Systems

The SOA infrastructure enables the three basic operations to support service-oriented systems: service discovery, service composition, and service invocation.

Service Discovery

Service discovery is the mechanism by which service consumers become aware of available services and their capabilities. Service discovery starts with a one-time operation in which the service provider publishes its service in some form of service registry. The form of registry can range from a simple web page to a robust implementation with advanced query capabilities. The service registry is then queried at design time by the developers of service consumers for services with desired capabilities.

Even though there is much discussion about runtime discovery of services, the reality is that current technologies do not support runtime discovery.

Even though there is much discussion about runtime discovery of services, the reality is that current technologies do not support runtime discovery. The word *dynamic* is often used to describe the binding between service consumers and services. There are various degrees of dynamism. At the lower end of the spectrum is late binding of a proxy service to a specific service instance that depends on user context or load-balancing policies. At the higher end of the spectrum is fully dynamic binding in which service consumers are capable of querying service registries at runtime, selecting the "best" service from the list of returned services, and invoking the selected service—all at runtime, and without human intervention. Late binding is a common, out-of-the-box feature of many commercial and open-source SOA infrastructures. Fully dynamic binding, on the other hand, requires semantically described services that use an ontology that is shared between service consumers and service providers. Semantic Web Services represent an active area of research, as well as an unsolved problem that is not yet ready for large-scale deployment.

Service Composition

Service composition is the mechanism by which services are combined to fulfill a business or operational process. Service composition can be done fully within the service consumer, but there is also considerable support for service composition within SOA infrastructures, specifically in Business Process Modeling (BPM) infrastructure components. Typical BPM components enable a service consumer developer to graphically compose services available in a registry and then generate the appropriate code for the *orchestration*.

Service Invocation

Service invocation is the mechanism by which services are invoked by service consumers at runtime. There are two basic invocation patterns: point-to-point and mediated.

In the point-to-point invocation pattern, service consumers directly invoke services over a network. Point-to-point is most acceptable in environments that are small in number of services and consumers, homogenous in implementation technologies, and have low pace of change (business and technology).

In the mediated pattern, service consumers invoke services via a middleware component such as an Enterprise Service Bus (ESB). The mediated pattern is most acceptable in environments that are large in number of services and consumers, technologically diverse and rapidly changing.

Web Services

Web Services is one technology pattern for implementing service-oriented systems, as shown in Figure 2. Because it is the most common technology pattern, Web Services is often equated to SOA. Initially, Web Services were commonly implemented using the WS* stack. More recently, an alternative called REpresentational State Transfer (REST) has surfaced and is being widely adopted.

Because it is the most common technology pattern, Web Services is often equated to SOA. Initially, Web Services were commonly implemented using the WS* stack. In the last several years, an alternative called REST has surfaced and is being widely adopted.

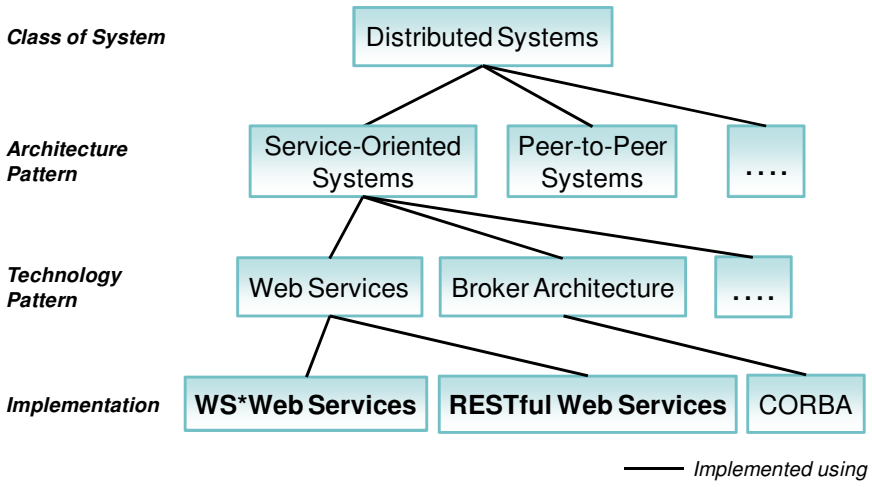

Figure 2: Web Services in the Context of Distributed Systems

WS* Web Services

In the simplest implementation of WS* Web Services

- Service interfaces are described using Web Services Description Language (WSDL).
- Data is transmitted using SOAP over HTTP.
- There is a way for service consumers to discover services and obtain information of how to bind to them, such as UDDI-based registries, database-based service registries, web pages, or wikis.

In a traditional request-response implementation of WS* Web Services, a service request is formed as an XML document according to the WSDL documentation of the service and bundled in a SOAP message that is transported via HTTP. The

service request is parsed and processed appropriately by the service provider, which then sends back a service response that is also an XML document formed according to the service's WSDL document and processed in the same way.

Even though the number changes frequently, there are currently approximately 100 WS* standards.

In addition to these base standards, there are many additional standards that deal with aspects such as service orchestration and choreography, as well as the support for cross-cutting quality attributes, as shown in Figure 3. Even though the number changes frequently, there are currently over 100 WS* standards. In Figure 3, standards names in green mean that the standard is fairly stable, yellow-orange means that it is becoming the standard, and red means that work on the standard has stopped but the standard is still used.

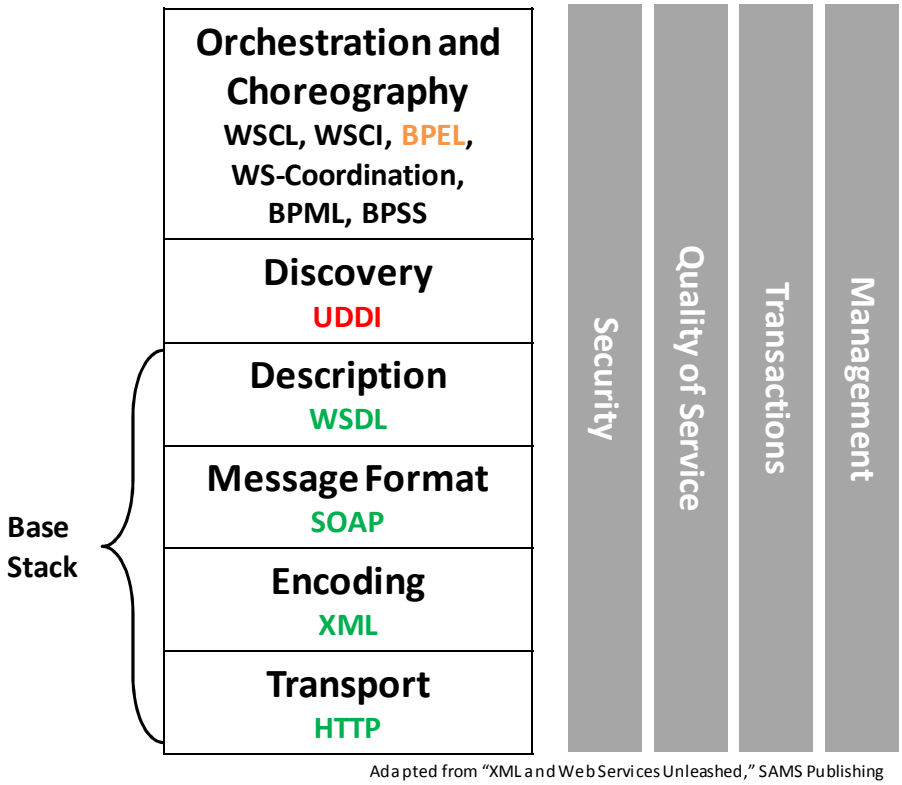

Figure 3: WS Protocol Stack*

A problem in WS* Services is that most of the standards related to the cross-cutting quality attributes are still emerging and even competing. This means that standards for implementing higher levels or cross-cutting aspects of the protocol stack require careful evaluation and selection.

Another aspect that is worth mentioning is that interoperability needs agreement on both syntax and semantics. WS* Web Services enable syntactic interoperability but do not guarantee semantic interoperability. Even though XML Schema

defines structure and data types and WSDL defines the service interfaces (operations, parameters, and return values), XML and WSDL do not define the meaning of data and WSDL does not define what a service does.

From an implementation perspective, what all this means is that systems that implement the WS* base stack are highly interoperable at the syntactic level, but this is not a guarantee for systems that implement higher levels or cross-cutting aspects of the protocol stack. Also, higher levels of interoperability such as semantic interoperability have to be built into the system through the use of data models or ontologies that describe the data that is being exchanged.

RESTful Web Services

REST provides a much simpler implementation of Web Services. In the REST implementation of Web Services, every entity that can be identified, named, or handled is considered a resource that is addressable by using a universal syntax (i.e., Universal Resource Identifier—URI). Consumers communicate with services using basic HTTP operations: GET, POST, PUT, and DELETE.

The main strength of RESTful Web Services is their simplicity: the request is a simple HTTP operation and the response is plain XML text embedded in an HTTP response. However, in REST there are no standards to support quality attributes, other than the basics (e.g., transport-level security using HTTPS). This means that REST may not be appropriate in environments with demanding quality attribute requirements.

WS* vs. REST

Web Services are commonly implemented using the WS* stack, but REST is starting to be widely adopted. The decision to implement WS* or REST will depend mainly on system quality attributes requirements. WS* implementation has greater support for security, availability, transaction management, etc. RESTful implementations, because of their simplicity, are more appropriate for read-only capabilities, typical of mashups, where there are minimal quality attribute requirements and concerns.

Enterprise Service Bus (ESB)

An Enterprise Service Bus (ESB) is a "middleware product that connects and mediates all communications and interactions between service consumers and services, usually based on standards."[2] Even though an ESB is not required to implement a service-oriented system, it is useful in large, heterogeneous service-oriented environments because it reduces the complexity of connecting services

[2] Definition from Wikipedia: http://en.wikipedia.org/wiki/Enterprise_service_bus

with their consumers by implementing the VETRO pattern,[3] as shown in Figure 4.

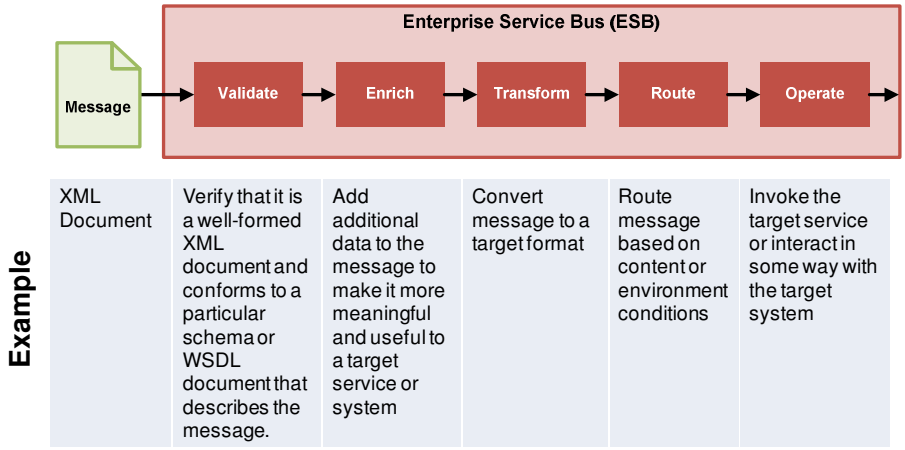

Figure 4: The VETRO Pattern

The implementation of the VETRO pattern is the way that the ESB manages aspects such as interface compatibility, service versioning, service routing, and data transformations. In essence, an ESB handles many of the potential incompatibilities between service consumers and services and provides elements to guarantee quality attribute requirements such as routing based on load balancing to promote performance and availability.

Another important component of an ESB is the service registry. As mentioned previously, the registry contains metadata about services that have been registered by their providers and are available for use. At a minimum, the registry contains the specification (or contract) for using the service (e.g., the WSDL document in the case of WS* Web Services). However, a registry can be configured to capture additional information that can be used in queries, such as

- Description
- Classification
- Usage history
- Test cases and test results
- Quality attribute metrics
- Additional documentation

A registry can also help keep track of service consumers. This is useful to understand required service levels, facilitate change impact analysis, and notify consumers of service changes.

---

[3] Source: Dave Chappell, *Enterprise Service Bus* (O'Reilly, 2004, ISBN 0-596-00675-6)

Summary

SOA is currently the best option available for systems integration and the leveraging of legacy systems. Technologies to implement service-oriented systems will certainly evolve to address emerging needs, but the concepts will remain. It is important to have a common SOA terminology and create a baseline for activities such as

- evaluating technologies in their context of use
- selecting technologies based on their relationship to business and operational goals
- understanding situations in which a service-oriented approach is appropriate and situations in which it is not
- analyzing service-oriented systems from the perspective of the service consumer, the service provider, and the SOA infrastructure
- understanding which parts of the system will come "out-of-the box" and which will have to be built

Resources

- Learn about SOA through SEI courses. For more information, visit http://www.sei.cmu.edu/go/soaofferings/index.cfm.
- Explore SEI SOA reports, webinars, podcasts, and presentations. For more information, visit http://www.sei.cmu.edu/library/abstracts/soa/.
- For information on SEI products and services, write to info@sei.cmu.edu.