

Service-oriented architecture

A **service-oriented architecture (SOA)** is an architectural pattern in computer software design in which application components provide services to other components via a communications protocol, typically over a network. The principles of service-orientation are independent of any vendor, product or technology.^[1]

A service is a self-contained unit of functionality, such as retrieving an online bank statement.^[2] By that definition, a service is an operation that may be discretely invoked. However, in the **Web Services Description Language (WSDL)**, a service is an interface definition that may list several discrete services/operations. And elsewhere, the term service is used for a component that is encapsulated behind an interface. This widespread ambiguity is reflected in what follows.

Services can be combined to provide the functionality of a large software application.^[3] SOA makes it easier for software components on computers connected over a network to cooperate. Every computer can run any number of services, and each service is built in a way that ensures that the service can exchange information with any other service in the network without human interaction and without the need to make changes to the underlying program itself.

1 Definitions

The **OASIS group**^[4] and the **Open Group**^[5] have both created formal definitions. OASIS defines SOA as:

A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

The Open Group's definition is:

Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation. Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services.

A service:

Is a logical representation of a repeatable business activity that has a

specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)

Is self-contained

May be composed of other services

Is a "black box" to consumers of the service

2 Overview

Services are **unassociated**, **loosely coupled** units of functionality that are self-contained. Each service implements at least one action, such as submitting an online application for an account, retrieving an online bank statement or modifying an online booking or airline ticket order. Within a SOA, services use defined protocols that describe how services pass and parse messages using description metadata, which in sufficient detail describes not only the characteristics of these services, but also the data that drives them. Programmers have made extensive use of XML in SOA to structure metadata that they wrap in a nearly exhaustive description-container. Analogously, the **Web Services Description Language (WSDL)** typically describes the services themselves, while **SOAP** (originally Simple Object Access Protocol) describes the communications protocols. SOA depends on data and services that are described by metadata that should meet the following two criteria:

1. The metadata should be provided in a form that software systems can use to configure dynamically by discovery and incorporation of defined services, and also to maintain coherence and integrity. For example, metadata could be used by other applications, like a catalogue, to perform auto discovery of services without modifying the functional contract of a service.
2. The metadata should be provided in a form that system designers can understand and manage with a reasonable expenditure of cost and effort.

The purpose of SOA is to allow users to combine fairly large chunks of functionality to form *ad hoc* applications built almost entirely from existing software services. The larger the chunks, the fewer the interfaces required to implement any given set of functionality; however, very large chunks of functionality may not prove sufficiently

granular for easy reuse. Each interface brings with it some amount of processing overhead, so there is a performance consideration in choosing the granularity of services.

SOA as an architecture relies on service-orientation as its fundamental design principle. If a service presents a simple interface that abstracts away its underlying complexity, then users can access independent services without knowledge of the service's platform implementation.^[6]

3 SOA framework

SOA-based solutions endeavour to enable business objectives while building an enterprise-quality system. SOA architecture is viewed as five horizontal layers:^[7]

1. Consumer Interface Layer – These are GUI for end users or apps accessing apps/service interfaces.
2. Business Process Layer – These are choreographed services representing business use-cases in terms of applications.
3. Services – Services are consolidated together for whole-enterprise in-service inventory.
4. Service Components – The components used to build the services, such as functional and technical libraries, technological interfaces etc.
5. Operational Systems – This layer contains the data models, enterprise data repository, technological platforms etc.

There are four cross-cutting vertical layers, each of which are applied to and supported by each of the following horizontal layers:

1. Integration Layer – starts with platform integration (protocols support), data integration, service integration, application integration, leading to enterprise application integration supporting B2B and B2C.
2. Quality of Service – Security, availability, performance etc. constitute the quality of service parameters which are configured based on required SLAs, OLAs.
3. Informational – provide business information.
4. Governance – IT strategy is governed to each horizontal layer to achieve required operating and capability model.

4 Design concept

SOA is based on the concept of a *service*. Depending on the service design approach taken, each SOA service is designed to perform one or more activities by implementing one or more service operations. As a result, each service is built as a discrete piece of code. This makes it possible to reuse the code in different ways throughout the application by changing only the way an individual service interoperates with other services that make up the application, versus making code changes to the service itself. SOA design principles are used during software development and integration.

SOA generally provides a way for consumers of services, such as web-based applications, to be aware of available SOA-based services. For example, several disparate departments within a company may develop and deploy SOA services in different implementation languages; their respective clients will benefit from a well-defined interface to access them.

SOA defines how to integrate widely disparate applications for a Web-based environment and uses multiple implementation platforms. Rather than defining an API, SOA defines the interface in terms of protocols and functionality. An *endpoint* is the entry point for such a SOA implementation.

Service-orientation requires *loose coupling* of services with operating systems and other technologies that underlie applications. SOA separates functions into distinct units, or services,^[8] which developers make accessible over a network in order to allow users to combine and reuse them in the production of applications. These services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format, or by coordinating an activity between two or more services.^[9]

For some, SOA can be seen as part of the continuum which ranges from the older concept of distributed computing^{[8][10]} and modular programming, through SOA, and on to current practices of mashups, SaaS, and cloud computing (which some see as the offspring of SOA).^[11]

5 Principles

There are no industry standards relating to the exact composition of a service-oriented architecture, although many industry sources have published their own principles. Some of these^{[12][13][14][15]} include the following:

- *Standardized service contract*: Services adhere to a communications agreement, as defined collectively by one or more service-description documents.
- *Service loose coupling*: Services maintain a relationship that minimizes dependencies and only requires

that they maintain an awareness of each other.

- *Service abstraction*: Beyond descriptions in the service contract, services hide logic from the outside world.
- *Service reusability*: Logic is divided into services with the intention of promoting reuse.
- *Service autonomy*: Services have control over the logic they encapsulate, from a Design-time and a run-time perspective.
- *Service statelessness*: Services minimize resource consumption by deferring the management of state information when necessary^[16]
- *Service discoverability*: Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.
- *Service composability*: Services are effective composition participants, regardless of the size and complexity of the composition.
- *Service granularity*: A design consideration to provide optimal scope and right granular level of the business functionality in a service operation.
- *Service normalization*: Services are decomposed or consolidated to a level of normal form to minimize redundancy. In some cases, services are denormalized for specific purposes, such as performance optimization, access, and aggregation.^[17]
- *Service optimization*: All else being equal, high-quality services are generally preferable to low-quality ones.
- *Service relevance*: Functionality is presented at a granularity recognized by the user as a meaningful service.
- *Service encapsulation*: Many services are consolidated for use under the SOA. Often such services were not planned to be under SOA.
- *Service location transparency*: This refers to the ability of a service consumer to invoke a service regardless of its actual location in the network. This also recognizes the discoverability property (one of the core principle of SOA) and the right of a consumer to access the service. Often, the idea of service virtualization also relates to location transparency. This is where the consumer simply calls a logical service while a suitable SOA-enabling runtime infrastructure component, commonly a service bus, maps this logical service call to a physical service.

5.1 Service architecture

This is the physical design of an individual service that encompasses all the resources used by a service. This would normally include databases, software components, legacy systems, *identity stores*, *XML schemas* and any backing stores, e.g. shared directories. It is also beneficial to include any *service agents* employed by the service, as any change in these service agents would affect the message processing capabilities of the service.

The (standardized service contract) design principle keeps service *contracts* independent from their implementation. The service contract needs to be documented to formalize the required processing resources by the individual service capabilities. Although it is beneficial to document details about the service architecture, the *service abstraction* design principle dictates that any internal details about the service are invisible to its consumers so that they do not develop any unstated *couplings*. The service architecture serves as a point of reference for evolving the service or gauging the impact of any change in the service.

5.2 Service composition architecture

One of the core characteristics of services developed using the *service-orientation* design paradigm is that they are *composition-centric*. Services with this characteristic can potentially address novel requirements by recomposing the same services in different configurations. Service composition architecture is itself a composition of the individual architectures of the participating services. In the light of the Service Abstraction principle, this type of architecture only documents the service contract and any published service-level agreement (SLA); internal details of each service are not included.

If a service composition is a part of another (parent) composition, the parent composition can also be referenced in the child service composition. The design of service composition also includes any alternate paths, such as error conditions, which may introduce new services into the current service composition.

Service composition is also a key technique in software integration, including enterprise software integration, business process composition and workflow composition.

5.3 Service inventory architecture

A service inventory is composed of services that automate business processes. It is important to account for the combined processing requirements of all services within the service inventory. Documenting the requirements of services, independently from the business processes that they automate, helps identify processing bottlenecks. The

service inventory architecture is documented from the service inventory blueprint, so that service candidates^[18] can be redesigned before their implementation.

5.4 Service-oriented enterprise architecture

This umbrella architecture incorporates service, composition, and inventory architectures, plus any enterprise-wide technological resources accessed by these architectures e.g. an ERP system. This can be further supplemented by including enterprise-wide standards that apply to the aforementioned architecture types. Any segments of the enterprise that are not service-oriented can also be documented in order to consider transformation requirements if a service needs to communicate with the business processes automated by such segments. SOA's main goal is to deliver agility to business.

6 Web services approach

Web services can implement a service-oriented architecture.^[19] They make functional building-blocks accessible over standard Internet protocols independent of platforms and programming languages. These services can represent either new applications or just wrappers around existing legacy systems to make them network-enabled.

Each SOA building block can play one or both of two roles:

1. *Service provider*: The service provider creates a web service and possibly publishes its interface and access information to the service registry. Each provider must decide which services to expose, how to make trade-offs between security and easy availability, how to price the services, or (if no charges apply) how/whether to exploit them for other value. The provider also has to decide what category the service should be listed in for a given broker service and what sort of trading partner agreements are required to use the service. It registers what services are available within it, and lists all the potential service recipients. The implementer of the broker then decides the scope of the broker. Public brokers are available through the Internet, while private brokers are only accessible to a limited audience, for example, users of a company intranet. Furthermore, the amount of the offered information has to be decided. Some brokers specialize in many listings. Others offer high levels of trust in the listed services. Some cover a broad landscape of services and others focus within an industry. Some brokers catalog other brokers. Depending on the business model, brokers can attempt to maximize

look-up requests, number of listings or accuracy of the listings. The **Universal Description Discovery and Integration (UDDI)** specification defines a way to publish and discover information about Web services. Other service broker technologies include (for example) **ebXML** (Electronic Business using eXtensible Markup Language) and those based on the **ISO/IEC 11179 Metadata Registry (MDR)** standard.

2. *Service consumer*: The service consumer or web service client locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its web services. Whichever service the service-consumers need, they have to take it into the brokers, bind it with respective service and then use it. They can access multiple services if the service provides multiple services.

7 Web service protocols

See also: [List of web service protocols](#)

Implementers commonly build SOAs using web services standards (for example, SOAP) that have gained broad industry acceptance after recommendation of Version 1.2 from the W3C^[20] (World Wide Web Consortium) in 2003. These standards (also referred to as *web service specifications*) also provide greater interoperability and some protection from lock-in to proprietary vendor software. One can, however, implement SOA using any service-based technology, such as Jini, CORBA or REST.

8 ESB in context of SOA

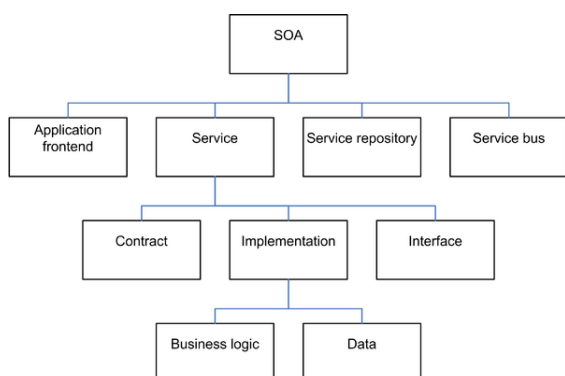
While knowing this it is obvious that even peer-to-peer applications are no direct linkages and require the middleware agent as well. Otherwise the endpoints need to program logic to wait for delayed server response, network errors, security and encryption and any kind of Quality of Service. The more endpoints exchange information mutually the more the need arises for a well-designed middleware. Such a middleware that provides standardized communication services in a multi-peer client server architecture is labelled Enterprise Service Bus (ESB). The services provided by an ESB exists in any communication, however often they are programmed individually; they comprise such services like routing, filtering, QoS, encryption, archiving, message queuing, logging, communication trace and others.

9 Other SOA concepts

Architectures can operate independently of specific technologies and can therefore be implemented using a wide range of technologies, including:

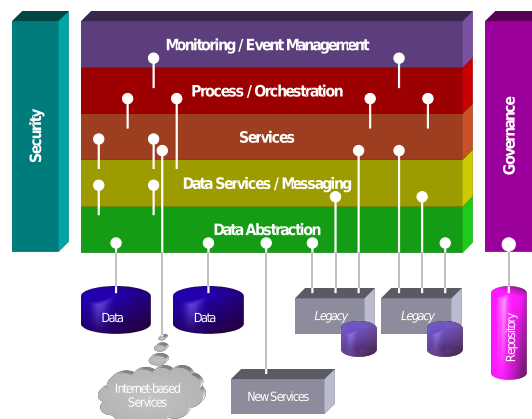
- SOAP, RPC
- REST
- DCOM
- CORBA
- OPC-UA
- Web services
- DDS
- Java RMI
- WCF (Microsoft's implementation of web services now forms a part of WCF)
- Apache Thrift
- SORCER

Implementations can use one or more of these protocols and, for example, might use a file-system mechanism to communicate data following a defined interface specification between processes conforming to the SOA concept. The key is independent services with defined interfaces that can be called to perform their tasks in a standard way, without a service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks.

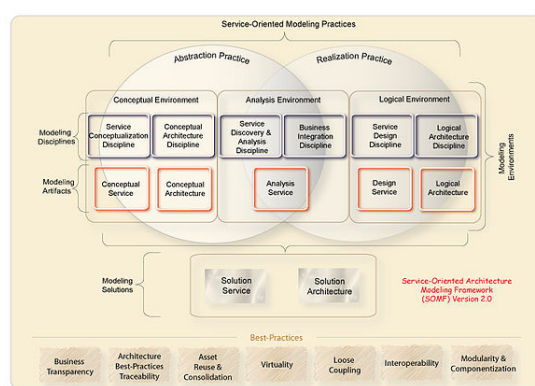


Elements of SOA, by Dirk Kraefzig, Karl Banke, and Dirk Slama^[21]

SOA enables the development of applications that are built by combining loosely coupled and interoperable services.^[22]



SOA meta-model, The Linthicum Group, 2007



Service-Oriented Modeling Framework (SOMF) Version 2.0

These services inter-operate based on a formal definition (or contract, e.g., WSDL) that is independent of the underlying platform and programming language. The interface definition **hides the implementation** of the language-specific service. SOA-based systems can therefore function independently of development technologies and platforms (such as Java, .NET, etc.). Services written in C# running on .NET platforms and services written in Java running on Java EE platforms, for example, can both be consumed by a common composite application (or client). Applications running on either platform can also consume services running on the other as web services that facilitate reuse. Managed environments can also wrap COBOL legacy systems and present them as software services.^[23]

High-level languages such as BPEL and specifications such as WS-CDL and WS-Coordination extend the service concept by providing a method of defining and supporting orchestration of fine-grained services into more coarse-grained business services, which architects can in turn incorporate into workflows and business processes implemented in composite applications or portals.^[24]

Service-oriented modeling^[8] is an SOA framework that identifies the various disciplines that guide SOA practi-

tioners to conceptualize, analyze, design, and architect their service-oriented assets. The **Service-oriented modeling framework (SOMF)** offers a modeling language and a work structure or “map” depicting the various components that contribute to a successful service-oriented modeling approach. It illustrates the major elements that identify the “what to do” aspects of a service development scheme. The model enables practitioners to craft a **project plan** and to identify the milestones of a service-oriented initiative. SOMF also provides a common modeling notation to address alignment between business and IT organizations.

10 Organizational benefits

Some **enterprise architects** believe that SOA can help businesses respond more quickly and more cost-effectively to changing market conditions.^[25] This style of *architecture* promotes reuse at the macro (service) level rather than micro (classes) level. It can also simplify interconnection to—and usage of—existing IT (legacy) assets.

With SOA, the idea is that an organization can look at a problem holistically. A business has more overall control. Theoretically there would not be a mass of developers using whatever tool sets might please them. But rather they would be coding to a standard that is set within the business. They can also develop enterprise-wide SOA that encapsulates a business-oriented infrastructure. SOA has also been illustrated as a highway system providing efficiency for car drivers. The point being that if everyone had a car, but there was no highway anywhere, things would be limited and disorganized, in any attempt to get anywhere quickly or efficiently. IBM Vice President of Web Services Michael Liebow says that SOA “builds highways”.^[26]

In some respects, SOA could be regarded as an architectural evolution rather than as a revolution. It captures many of the **best practices** of previous software architectures. In communications systems, for example, little development of solutions that use truly static bindings to talk to other equipment in the network has taken place. By formally embracing a SOA approach, such systems can position themselves to stress the importance of well-defined, highly inter-operable interfaces.^[27]

A service comprises a stand-alone unit of functionality available only via a formally defined interface. Services can be some kind of “nano-enterprises” that are easy to produce and improve. Also services can be “mega-corporations” constructed as the coordinated work of subordinate services.

A mature rollout of SOA effectively defines the API of an organization.

Reasons for treating the implementation of services as separate projects from larger projects include:

1. Separation promotes the concept to the business that services can be delivered quickly and independently from the larger and slower-moving projects common in the organization. The business starts understanding systems and simplified user interfaces calling on services. This advocates **agility**. That is to say, it fosters business innovations and speeds up time-to-market.^[28]
2. Separation promotes the decoupling of services from consuming projects. This encourages good design insofar as the service is designed without knowing who its consumers are.
3. Documentation and test artifacts of the service are not embedded within the detail of the larger project. This is important when the service needs to be reused later.

An indirect benefit of SOA involves dramatically simplified testing. Services are autonomous, stateless, with fully documented interfaces, and separate from the cross-cutting concerns of the implementation.

If an organization possesses appropriately defined test data, then a corresponding stub is built that reacts to the test data when a service is being built. A full set of regression tests, scripts, data, and responses is also captured for the service. The service can be tested as a 'black box' using existing stubs corresponding to the services it calls. Test environments can be constructed where the primitive and out-of-scope services are stubs, while the remainder of the mesh is test deployments of full services. As each interface is fully documented with its own full set of regression test documentation, it becomes simple to identify problems in test services. Testing evolves to merely validate that the test service operates according to its documentation, and finds gaps in documentation and test cases of all services within the environment. Managing the data state of **idempotent** services is the only complexity.

Examples may prove useful to aid in documenting a service to the level where it becomes useful. The documentation of some APIs within the Java Community Process provide good examples. As these are exhaustive, staff would typically use only important subsets. The 'os-sjsa.pdf' file within JSR-89 exemplifies such a file.^[29]

11 Challenges

One obvious and common challenge faced involves managing *services metadata*.^[30] SOA-based environments can include many services that exchange messages to perform tasks. Depending on the design, a single application may generate millions of messages. Managing and providing information on how services interact can become complex. This becomes even more complicated when these services are delivered by different organizations within the company or even different companies

(partners, suppliers, etc.). This creates huge trust issues across teams; hence SOA Governance comes into the picture.

Another challenge involves the *lack of testing* in SOA space. There are no sophisticated tools that provide testability of all headless services (including message and database services along with web services) in a typical architecture. Lack of horizontal trust requires that both producers and consumers test services on a continuous basis. It is important to invest in a testing framework (build it or buy it) that would provide the visibility required to find the culprit in the architecture. **Business agility** requires SOA services to be controlled by the business goals and directives as defined in the **business Motivation Model (BMM)**.^[31]

Interoperability becomes an important aspect of SOA implementations. The WS-I organization has developed basic profile (BP) and basic security profile (BSP) to enforce compatibility.^[32] WS-I has designed testing tools to help assess whether web services conform to WS-I profile guidelines. Additionally, another charter has been established to work on the Reliable Secure Profile.

Significant *vendor hype* surrounds SOA, which can create exaggerated expectations. Product stacks continue to evolve as early adopters test the development and runtime products with real-world problems. SOA does not guarantee reduced IT costs, improved systems agility or shorter time to market. Successful SOA implementations may realize some or all of these benefits depending on the quality and relevance of the system architecture and design.^{[33][34]}

12 Criticisms

Some criticisms of SOA depend on conflating SOA with **Web services**.^[35] In the absence of native or binary forms of remote procedure call (RPC), applications could run more slowly and require more processing power, increasing costs. Most implementations do incur these overheads, but SOA can be implemented using technologies (for example, **Java Business Integration (JBI)**, **Windows Communication Foundation (WCF)** and **data distribution service (DDS)**) that do not depend on remote procedure calls or translation through XML. At the same time, emerging open-source XML parsing technologies (such as **VTD-XML**) and various XML-compatible binary formats promise to significantly improve SOA performance. Services implemented using JSON instead of XML do not suffer from this performance concern.^{[36][37][38]}

Stateful services require both the consumer and the provider to share the same consumer-specific context, which is either included in or referenced by messages exchanged between the provider and the consumer. This constraint has the drawback that it could reduce the overall scalability of the service provider if the service-

provider needs to retain the shared context for each consumer. It also increases the coupling between a service provider and a consumer and makes switching service providers more difficult.^[39] Ultimately, some critics feel that SOA services are still too constrained by applications they represent.^[40]

Another concern relates to the ongoing evolution of **WS-*** standards and products (e. g., transaction, security), and SOA can thus introduce new risks unless properly managed and estimated with additional budget and contingency for additional **proof-of-concept** work.

The next step in the design process covers the definition of a service delivery platform (SDP) and its implementation. In the SDP design phase one defines the business information models, identity management, products, content, devices, and the end-user service characteristics, as well as how agile the system is so that it can deal with the evolution of the business and its customers.

13 SOA Manifesto

In October 2009, at the *2nd International SOA Symposium*, a mixed group of 17 independent SOA practitioners and vendors, the “SOA Manifesto Working Group,” announced the publication of the SOA Manifesto.^[41] The SOA Manifesto is a set of objectives and guiding principles that aim to provide a clear understanding and vision of SOA and service-orientation. Its purpose is rescuing the SOA concept from an excessive use of the term by the vendor community and “a seemingly endless proliferation of misinformation and confusion.”^[42]

The manifesto provides a broad definition of SOA, the values it represents for the signatories and some guiding principles. The manifesto prioritizes:

- Business value over technical strategy
- Strategic goals over project-specific benefits
- Intrinsic interoperability over custom integration
- Shared services over specific-purpose implementations
- Flexibility over optimization
- Evolutionary refinement over pursuit of initial perfection

As of September 2010, the SOA Manifesto had been signed by more than 700 signatories and had been translated to nine languages.

14 Extensions

14.1 Web 2.0

Tim O'Reilly coined the term “Web 2.0” to describe a perceived, quickly growing set of web-based applications.^[43] A topic that has experienced extensive coverage involves the relationship between Web 2.0 and Service-Oriented Architectures (SOAs).

SOA is the philosophy of encapsulating application logic in services with a uniformly defined interface and making these publicly available via discovery mechanisms. The notion of complexity-hiding and reuse, but also the concept of loosely coupling services has inspired researchers to elaborate on similarities between the two philosophies, SOA and Web 2.0, and their respective applications. Some argue Web 2.0 and SOA have significantly different elements and thus can not be regarded “parallel philosophies”, whereas others consider the two concepts as complementary and regard Web 2.0 as the global SOA.^[44]

The philosophies of Web 2.0 and SOA serve different user needs and thus expose differences with respect to the design and also the technologies used in real-world applications. However, as of 2008, use-cases demonstrated the potential of combining technologies and principles of both Web 2.0 and SOA.^[44]

In an “Internet of Services”, all people, machines, and goods will have access via the network infrastructure of tomorrow. The Internet will thus offer services for all areas of life and business, such as virtual insurance, online banking and music, and so on. Those services will require a complex services infrastructure including service-delivery platforms bringing together demand and supply. Building blocks for the Internet of Services include SOA, Web 2.0 and semantics on the technology side; as well as novel business models, and approaches to systematic and community-based innovation.^[45]

Even though Oracle indicates that Gartner is coining a new term, Gartner analysts indicate that they call this *advanced SOA* and refer to it as “SOA 2.0”.^[46] Most of the major middleware vendors (e. g., Red Hat, webMethods, TIBCO Software, IBM, Sun Microsystems, and Oracle) have had some form of SOA 2.0 attributes for years.

14.2 Internet of Things

As the idea of SOA is extended to large numbers of devices, we see the emergence of the **Internet of Things**. An approach to control and manage all the flows of information through such devices connecting them as services is called **BPM Everywhere**.^[47]

14.3 Microservices

Microservices are a modern interpretation of service-oriented architectures used to build distributed software



systems. Services in a microservice architecture^[48] are processes that communicate with each other over the network in order to fulfill a goal. These services use technology agnostic protocols,^[49] which aid in encapsulating choice of language and frameworks, making their choice a concern internal to the service. Microservices architectural style is a first realisation of SOA that has happened after the introduction of DevOps and this is becoming the standard for building continuously deployed systems.^[50]

15 See also

- Business-oriented architecture
- Component business model
- Enterprise service bus
- Open ESB
- Service layer
- Service-oriented architecture implementation framework
- Service (systems architecture)
- SOA governance
- SOALIB
- Web-oriented architecture

16 References

- [1] Chapter 1: Service Oriented Architecture (SOA). Msdn.microsoft.com. Retrieved on May 30, 2014.
- [2] “What Is SOA?”. opengroup. Retrieved 2013-08-19.
- [3] Velte, Anthony T. (2010). *Cloud Computing: A Practical Approach*. McGraw Hill. ISBN 978-0-07-162694-1.
- [4] SOA Reference Model definition
- [5] “Service Oriented Architecture : What Is SOA?”. opengroup.
- [6] Kishore Channabasavaiah, Kerrie Holley and Edward Tuggle, Jr. (December 16, 2003) Migrating to a service-oriented architecture, *IBM DeveloperWorks*.
- [7] “SOA Reference Architecture Technical Standard : Basic Concepts”. opengroup. Retrieved October 10, 2014.
- [8] Michael Bell (2008). “Introduction to Service-Oriented Modeling”. *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley & Sons. p. 3. ISBN 978-0-470-14111-3.
- [9] Michael Bell (2010). *SOA Modeling Patterns for Service-Oriented Discovery and Analysis*. Wiley & Sons. p. 390. ISBN 978-0-470-48197-4.

- [10] Thomas Erl (June 2005). *About the Principles*. Service-orientation.org
- [11] "Application Platform Strategies Blog: SOA is Dead; Long Live Services". Apsblog.burtongroup.com. January 5, 2009. Retrieved August 13, 2012.
- [12] Yvonne Balzer Improve your SOA project plans, *IBM*, July 16, 2004
- [13] Microsoft Windows Communication Foundation team (2012). "Principles of Service Oriented Design". *msdn.microsoft.com*. Retrieved September 3, 2012.
- [14] Principles by Thomas Erl of SOA Systems Inc. eight specific service-orientation principles
- [15] M. Hadi Valipour; Bavar AmirZafari; Kh. Niki Maleki; Negin Daneshpour (2009). "A brief survey of software architecture concepts and service oriented architecture". *2009 2nd IEEE International Conference on Computer Science and Information Technology*. pp. 34–38. doi:10.1109/ICCSIT.2009.5235004. ISBN 978-1-4244-4519-6.
- [16] Services Oriented Architecture (SOA) – Jargon Buster. Lansa.com. Retrieved on May 30, 2014.
- [17] Tony Shan (2004). "Building a service-oriented e Banking platform". *IEEE International Conference on Services Computing, 2004. (SCC 2004). Proceedings. 2004*. pp. 237–244. doi:10.1109/SCC.2004.1358011. ISBN 0-7695-2225-4.2004
- [18] "Service Candidate". ServiceOrientation.com. Retrieved October 17, 2014.
- [19] E. Oliveros (2012). "Web Service Specifications Relevant for Service Oriented Infrastructures". *Achieving Real-Time in Distributed Computing: From Grids to Clouds*. IGI Global. pp. 174–198. doi:10.4018/978-1-60960-827-9.ch010.
- [20] "SOAP Version 1.2  (W3C )" (in Japanese). W3.org. Retrieved August 13, 2012.
- [21] *Enterprise SOA*. Prentice Hall, 2005
- [22] Jorge Cardoso; Amit P. Sheth (2006). "Foreword". *Semantic Web Services, Processes and Applications*. SEMANTIC WEB AND BEYOND: Computing for Human Experience. Springer. xxi. ISBN 978-0-387-30239-3. The corresponding architectural style is called "service-oriented architecture": fundamentally, it describes how service consumers and service providers can be decoupled via discovery mechanisms resulting in loosely coupled systems. Implementing a service-oriented architecture means to deal with heterogeneity and interoperability concerns.
- [23] Haruhiro, Okishima (2006). "Case Study of System Architectures that use COBOL Assets" (PDF). *Fujitsu Scientific and Technical Journal*. **42** (3): 414–424. Retrieved March 27, 2006.
- [24] Kyriazis, Dimosthenis; Tserpes, Konstantinos; Menychtas, Andreas; Sarantidis, Ioannis; Varvarigou, Theodora (2008). "Service selection and workflow mapping for Grids: an approach exploiting quality-of-service information". *Concurrency and Computation: Practice and Experience*. **21** (6): 739–766. doi:10.1002/cpe.1343.
- [25] Christopher Koch A New Blueprint For The Enterprise, *CIO Magazine*, March 1, 2005
- [26] Elizabeth Millard (January 2005). "Building a Better Process". *Computer User*. Page 20.
- [27] Norbert Bieberstein, Sanjay Bose, Marc Fiammante (2005) *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap*, IBM Press books. ISBN 0133762904
- [28] Brayan Zimmerli (November 11, 2009) Business Benefits of SOA, *University of Applied Science of Northwestern Switzerland, School of Business*
- [29] JSR-000089 OSS Service Activation API Specification 1.0 Final Release. sun.com
- [30] Sikka, Vishal (2005). "Data and Metadata Management in Service Oriented Architectures: Some Open Challenges". *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM. pp. 849–850. doi:10.1145/1066157.1066264.
- [31] "From The Business Motivation Model (BMM) To Service Oriented Architecture (SOA)". Jot.fm. Retrieved June 15, 2013.
- [32] Basic Profile Version 1.0. ws-i.org. April 16, 2004
- [33] Is There Real Business Value Behind the Hype of SOA?, *Computerworld*, June 19, 2006.
- [34] See also: WS-MetadataExchange OWL-S
- [35] Joe McKendrick. "Bray: SOA too complex; 'just vendor BS'". ZDNet.
- [36] Jimmy Zhang (February 20, 2008) "Index XML Documents with VTD-XML". *XML Journal*.
- [37] Jimmy Zhang (August 5, 2008) "i-Technology Viewpoint: The Performance Woe of Binary XML". *Microservices Journal*.
- [38] Jimmy Zhang (January 9, 2008) "Manipulate XML Content the Ximple Way". *devx.com*.
- [39] "The Reason SOA Isn't Delivering Sustainable Software". *jpmorgenthal.com*. June 19, 2009. Retrieved June 27, 2009.
- [40] "SOA services still too constrained by applications they represent". *znet.com*. June 27, 2009. Retrieved June 27, 2009.
- [41] SOA Manifesto Official Website Date Accessed: October 2, 2010.
- [42] About the SOA Manifesto

- [43] “What Is Web 2.0”. Tim O’Reilly. September 30, 2005. Retrieved June 10, 2008.
- [44] Christoph Schroth & Till Janner (2007). “Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services”. *IT Professional* 9 (2007), Nr. 3, pp. 36–41, IEEE Computer Society. Retrieved February 23, 2008.
- [45] Rainer Ruggaber (2007). “Internet of Services—A SAP Research Vision” (PDF). IEEE Computer Society. Retrieved February 23, 2008.
- [46] Yefim Natis & Roy Schulte Advanced SOA for Advanced Enterprise Projects, *Gartner*, July 13, 2006
- [47] Swenson, Keith & Palmer, Nathaniel. “BPM Everywhere - Internet of Things, Process of Everything”. Future Strategies, Incorporated (22 April 2015). ISBN 978-0986321412.
- [48] “Microservices: yesterday, today, and tomorrow” (PDF). Retrieved 6 July 2016.
- [49] Martin Fowler. “Microservices”.
- [50] Balalaie, A.; Heydarnoori, A.; Jamshidi, P. (2016-05-01). “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture”. *IEEE Software*. **33** (3): 42–52. doi:10.1109/MS.2016.64. ISSN 0740-7459.

17 External links

- A comparison of SOA standards carried out for Ministry of Defence (United Kingdom) in 2010
- SOA in the real world – Microsoft Developer network
- SOA reference architecture from IBM
- SOA Practitioners Guide Part 2: SOA Reference Architecture
- SOA Practitioners Guide Part 3: Introduction to Services Lifecycle
- SOA for Existing Applications - A Case Study

18 Text and image sources, contributors, and licenses

18.1 Text

- Service-oriented architecture** *Source:* https://en.wikipedia.org/wiki/Service-oriented_architecture?oldid=740205708 *Contributors:* Shd-enwiki, Christian List, Frecklefoot, Edward, Michael Hardy, Repton, Kku, MartinSpamer, Haakon, Ronz, Arthur Frayn, Julesd, Stefan-S, MasterDirk, Dwo, Chprasad-enwiki, Fuzheado, Nickg, Rednblu, Greenrd, Pedant17, Maximus Rex, Sboehringer, Bevo, Nnh, Gentgeen, Rfc1394, Texture, Feigling, Caknuck, Bkell, Pengo, Dave6, Jonabbey, Varlaam, Khalid hassani, Pne, Matt Darby, Wmanhan, Neilc, Stevietheman, Gadfium, Metlin, Beland, OwenBlacker, Usrnmeh8er, Axelangeli, Lipton-enwiki, Ziroby, Kustere, Rfl, Breedlov, Discospinder, William Pietri, Rich Farmbrough, Rhobite, Avriette, ArnoldReinhold, ESKog, S.K., JoeSmack, Marx Gomes, Kyz, Shanes, Felagund, Simon South, Nigelj, Iain Cheyne, Polluks, Giraffedata, Cem Basman-enwiki, BlueNovember, Mdd, Iolar-enwiki, Kraupu, Pinar, Walter Görlitz, Dethron, Diego Moya, WTGDMan1986, M@, Warpsmith, Tayal01, Velella, Bugg, Adoble, SteveLetwin, H2g2bob, Saxifrage, Zntrip, Stuartyeates, Weyes, Angr, Jacobolus, Timosa, Cnb, Stefan Ivanovich, Varco, Mattmcc, GregorB, MauriceKA, Lastorset, ThomasOwens, Rogerd, Erebus555, Phantast, MZMcBride, Oblivious, Scorpiuss, FayssalF, Kmorozov, Penedo, RobyWayne, Chobot, Sharkface217, Gwernol, YurikBot, Whoisjohngalt, RussBot, Arado, Bovineone, SamJohnston, Alexliu-enwiki, SAE1962, Mccready, Brandon, Jpbowen, Tony1, Everyguy, Sneakymoose, LeonardWalstad, LandNav, Zzuuzz, WebWombat, Hadipedia, Tvarnoe-enwiki, Julube, GraemeL, TBadger, Aarsanjani, LeonardoRob0t, Fram, Backache, JLaTondre, DEng, Mbruggen, Halhelms, Rohitcool, Tom Morris, Binuraghavan, Riverawynter, That Guy, From That Show!, DocendoDiscimus, SmackBot, Reedy, McGeddon, Dwl, Blueshark-enwiki, Basil.bourque, Elwood j blues, Renesis, Rajah9, Netsql, Ordinant, Roma emu, Gilliam, Senfo, Ohnoitsjamie, Aamironline, Mdhmdh31, Anwar saadat, Chris the speller, Dlafont, Jjalexand, Thumperward, Philoserf, MalafayaBot, Sadads, Wykis, BBCWatcher, DHN-bot-enwiki, A. B., Mogman1, Peter Campbell, Jeff DLB, Atr0x, Ashishvaid, Cybercobra, Hslayer, Enarcuus, Dreadstar, Iskatel-enwiki, Warren, SeanAhern, Luís Felipe Braga, RayGates, Zbalai, Mrego, Moejorris, Kuru, Spir, Ebertelsen, Soumyasch, JorisvS, Aleenfl, Martin Wagenleiter, Dfass, Nutcracker, Joe.jackson, Hulmem, Jstrachan, Kompere, Beetstra, George The Dragon, Ehheh, Evmako, Afrab null, Zorabi, Akademy, Olimpiu.pop, ShakingSpirit, Hu12, Meitar, BananaFiend, JHP, Jamehealy, IvanLanin, Vocaro, Paul Foxworthy, Igoldste, Dp462090, AGK, Linkspamremover, Tawkerbot2, JeffAdkins, CmdrObot, Libin v, Raysonho, Kensall, Nunquam Dormio, Anil Kumar Kartham, Wsibob, Cybernetic, IanDBailey, Bbarthel, Billhunt, Slp1, Gogo Dodo, ST47, Solidpoint, Odie5533, Tawkerbot4, Torc2, DumbBOT, Dblanchard, Kckid, ErnstRohlicek, Ambitus, Assentt, IvoTotev, Corycasanave, Thijs'bot, Shuumass, Hervegirod, SReddy, HappyInGeneral, Sphodes, Z10x, Mschneid@agentos.net, RichardVeryard, Nick Number, Bob.gourley@comcast.net, Nobar, Peashy, AntiVandalBot, RobotG, Rohit Sood, Nickull, Randybardwell, Fayenatic london, Spencer, Ozgod, Wayiran, Fbahr, Barek, Yocto42, MER-C, Hasalaka, Michig, Chiya07, BrotherE, Technologyvoices, Kirrages, Daveh1, Shigidon, Magioladitis, VoABot II, MiguelMunoz, Transcendence, HouKid, Digitalfunda, Akmani, Jen2000, Tedickey, Maard, Schastain, Snowded, Straxus, Think4yourself, Kristen pucket, Wagnermr14, Hamiltonstone, Wwmbes, Allstarecho, David Eppstein, The Rapper, Cander0000, Muzikman67, ErikTownsend, Curtbeckmann, MartinBot, Sciagent, Davidjcmorris, Axlq, Lmxspice, Ahmadar, Zzglenn, Exostor, Fpbear, TroIII, NetManage, Pharaoh of the Wizards, Mange01, Fioranoweb, Samiam70002, Helon, JVersteeg, Salliesatt, Robert Illes, Monster1218, Marioct, Betswiki, Grshiplett, Tonyshan, Midnight Madness, Arms & Hearts, SJP, Kmittal, Yogishpai, Doug4, Christopher norton, Cometstyles, Fkroon, Jzupan, Davidslinthicum, Sbv, Fogartis, HighKing, Sempertinal, Stwomack, Tomerfiliba, Malik fairose, Psheld, Enderminh, Erpgenie, Vrac, Bramschoenmakers, Hominidx, Goflow6206, Digerateur, Dataprit, Parker007, A4bot, Stevers1, SynAsha, Gabhala, Tiago simoes, Oxfordwang, Lradrama, Q Chris, Graham Berrisford, Maxim, Bansipatel, Hajoworldwide, Jvlock, Jaideco, Fasih, Badja, Paladin1979, Rajeevgn, Kbrose, Colm.mcmullan, SieBot, Jzhang2007, LarsHolmberg, Yadoo86, YonaBot, Kgoarany, Y2ksw, Flyer22 Reborn, CIOGuru, JCLately, Jojalozzo, CutOffTies, EnOreg, Smet, Svick, Ncw0617, Odedtd45, Youngfu, HighFlyer12, ManOnPipes, SanderEvers, Escape Orbit, Jvlock527, Dancinggoat, Stevepolyak, ImageRemovalBot, Slwatmough, Sathya sh, Martarius, Cmagururaj, Sfan00 IMG, Spa015, Techwonder, GarbagEcol, ClueBot, Radharajgopal, Brandguru, Gnanaguru, Nklata, MRqtH2, Mmichaelbell, Prov519, The Thing That Should Not Be, H.E. Hall, Alksentrs, Malharbarai, Sbono, Drphallus, Gunaseelapandian, Ignorance is strength, SamTheButcher, Vatsakatta, StigBot, Ericseunans, ITBusinessEdge, Rose Booth, Carusus, Ddelash, ShachiDube, Lacomoeidia, NuclearWarfare, Mkuhbock, Swtechwr, ChrisKalt, Razorflame, Bruno rodin, El bot de la dieta, John at Lancelotlinc, Aprock, Bjdehut, Djmackenzie, Mymallandnews, Defragyourmind, Bpmsoa, GeoffMacartney, Expertjohn, DumZiBoT, Spinner4321, Abrobitroy, Milais, XLinkBot, Xagronaut, Heather.taylor, OWV, AngelaMartin2008, DamsonDragon, Wintaborn, AnitaRogel, ZooFari, Cesar.obach-enwiki, Richard.McGuire88, Addbot, Qiuzhuang.Lian, Sean R Fox, Betterusername, Non-dropframe, Maria C Mosak, DanielGellis, Raypereda, AroundLain80Days, Ddhanasekharan, Kobakhet, MrOllie, LaaknorBot, Jmilgram, Empathy321, Williamglasby, Ashwineek, Manishpk, Lightbot, OIEnglish, Richard R White, Ravisriv, Jarble, Blablablob, Yobot, Themfromspace, Bunnyhop11, TaBOT-zerem, Billing geek, Danperryy, Ricky123tony, Mojei, Dactyllic, Lelguea, AncientArk, AnomieBOT, Azambataro, Aaronaberg, HelloSOAWorld, Galoubet, Kingpin13, Sz-ibot, ItchyDE-enwiki, Materialscientist, Markcowan, JamesLWilliams2010, Citation bot, Haleyga, Capricorn42, Boaworm, Kiran.kaipa, Yetasoli, Alvin Seville, Soalib, Mangst, Aligilman, Prari, FrescoBot, Riventree, Mark Renier, Ibspublishing, Sidna, Sae1962, Feravoon-enwiki, BusinessMashups, HamburgerRadio, Citation bot 1, Nav141174, Winterst, I dream of horses, Momergil, JLRdperson, Alinabalint, Gedimin, MHPSM, DavidInCambridge, E.s.cohenlevy, MertWiki, Vinaysingla, Taylorrx, Soa101, LogAntiLog, Orangesodakid, Hanay, Darshana.jayasinghe, Vrenator, Allen4names, Crysb, Randimae, Rjabate, Starfishjones, Nottrobin, Beantowngal, Soaguy, Obankston, Lozw, Projectmbt, Bangsanegara, Jann.poppinga, Fiftytwo thirty, Quaysea, John of Reading, Dewritech, Faolin42, Sam Tomato, Biskup2010, Dcirovic, Jiri Pavelka, Thecheesykid, Bollyjeff, Traxs7, GZ-Bot, Ramachandra20, ISresearcher, BioPupil, Kirby2010, Yveschaix, Sunrisesunset12345, ClueBot NG, Ptrb, Roubo, Xeroboxer, Sudhanshuss, Satellizer, Luis Mailhos, Wambunet, Anshuman.dwivedi, Madhavan93, Carl presscott, George392, Helpful Pixie Bot, Wbm1058, GuySh, Taibah U, BG19bot, Wasbeer, Hvh22, Compfreak7, AdventurousSquirrel, Pkka02, BattyBot, Softwareqa, DucerGraphic, Mtriana, Cj211, Czafan, Jots and graphs, Baringmo, Mwsobol, Me, Myself, and I are Here, Gabby Merger, Lemnaminor, Biogeographist, Marinac93, Lasith011, GreggRock, Maura Driscoll, Comp.arch, Dnader90, Wtsai36, K0zka, Nishsvn, Jpmunz, Beaver creekful, Gorohoroh, Rabbidil, Amenyochts, Monkbot, Theenterprisearchitect, Aerosteak, Juanchristy, Haloedscape, Viam Ferream, Tsfto01, Yinongchen, KasparBot, Fill17buy100, Brent3600, YadvavCom, Omniscient21, Firebrace, Irene Brand, Ybenzine, Rsinha2, Orgedoneosy and Anonymous: 984

18.2 Images

- File:Ambox_important.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/b/b4/Ambox_important.svg *License:* Public domain *Contributors:* Own work, based off of Image:Ambox scales.svg *Original artist:* Dsmurat (talk · contribs)
- File:Commons-logo.svg** *Source:* <https://upload.wikimedia.org/wikipedia/en/4/4a/Commons-logo.svg> *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?

- **File:En-Service-oriented_architecture.ogg** *Source:* https://upload.wikimedia.org/wikipedia/commons/0/09/En-Service-oriented_architecture.ogg *License:* CC BY-SA 3.0 *Contributors:*
- Derivative of Service-oriented architecture *Original artist:* **Speaker:** Mangst
Authors of the article
- **File:Folder_Hexagonal_Icon.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/4/48/Folder_Hexagonal_Icon.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:SOA_Elements.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d4/SOA_Elements.png *License:* CC BY 2.5 *Contributors:* Dirk Krafzig, Karl Banke, and Dirk Slama. Enterprise SOA. Prentice Hall, 2005 *Original artist:* Florian Lindner (designer)
- **File:SOA_Metamodel.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/0/06/SOA_Metamodel.svg *License:* GFDL *Contributors:* self-made, based on SOA Meta Model.jpg by David S. Linthicum *Original artist:* Loïc Corbasson, created with en:OOo Draw (ODG source file available on request)
- **File:SOMF_V_2.0.jpg** *Source:* https://upload.wikimedia.org/wikipedia/en/7/77/SOMF_V_2.0.jpg *License:* PD *Contributors:* ? *Original artist:* ?
- **File:Sound-icon.svg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/4/47/Sound-icon.svg> *License:* LGPL *Contributors:* Derivative work from Silsor's versio *Original artist:* Crystal SVG icon set

18.3 Content license

- Creative Commons Attribution-Share Alike 3.0