WIKIPEDIA

Service-oriented architecture

Service-oriented architecture (**SOA**) is an architectural style that supports service orientation.[1] By consequence, it is as well applied in the field of software design where services are provided to the other components by application components, through a communication protocol over a network. A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online. SOA is also intended to be independent of vendors, products and technologies.[2]

Service orientation is a way of thinking in terms of services and service-based development and the outcomes of services.[1]

A service has four properties according to one of many definitions of SOA:[3]

1. It logically represents a repeatable business activity with a specified outcome.
2. It is self-contained.
3. It is a black box for its consumers, meaning the consumer does not have to be aware of the service's inner workings.
4. It may be composed of other services.[4]

Different services can be used in conjunction as a service mesh to provide the functionality of a large software application,[5] a principle SOA shares with modular programming. Service-oriented architecture integrates distributed, separately maintained and deployed software components. It is enabled by technologies and standards that facilitate components' communication and cooperation over a network, especially over an IP network.

SOA is related to the idea of an application programming interface (API), an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software. An API can be thought of as the service, and the SOA the architecture that allows the service to operate.

Contents

Overview

In SOA, services use protocols that describe how they pass and parse messages using description metadata. This metadata describes both the functional characteristics of the service and quality-of-service characteristics. Service-oriented architecture aims to allow users to combine large chunks of functionality to form applications which are built purely from existing services and combining them in an ad hoc manner. A service presents a simple interface to the requester that abstracts away the underlying complexity acting as a black box. Further users can also access these independent services without any knowledge of their internal implementation.[6]

Defining concepts

The related buzzword service-orientation promotes is *loose coupling* between services. SOA separates functions into distinct units, or services,[7] 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.[8]

A manifesto was published for service-oriented architecture in October, 2009. This came up with six core values which are listed as follows:[9]

1. **Business value** is given more importance than technical strategy.
2. **Strategic goals** are given more importance than project-specific benefits.
3. **Intrinsic interoperability** is given more importance than custom integration.
4. **Shared services** are given more importance than specific-purpose implementations.
5. **Flexibility** is given more importance than optimization.
6. **Evolutionary refinement** is given more importance than pursuit of initial perfection.

SOA can be seen as part of the continuum which ranges from the older concept of distributed computing[7][10] and modular programming, through SOA, and on to practices of mashups, SaaS, and cloud computing (which some see as the offspring of SOA).[11]

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 standard communications agreement, as defined collectively by one or more service-description documents within a given set of services.

Service reference autonomy (an aspect of loose coupling)
: The relationship between services is minimized to the level that they are only aware of their existence.

Service location transparency (an aspect of loose coupling)
: Services can be called from anywhere within the network that it is located no matter where it is present.

Service longevity
: Services should be designed to be long lived. Where possible services should avoid forcing consumers to change if they do not require new features, if you call a service today you should be able to call the same service tomorrow.

Service abstraction
: The services act as black boxes, that is their inner logic is hidden from the consumers.

Service autonomy
: Services are independent and control the functionality they encapsulate, from a Design-time and a run-time perspective.

Service statelessness
: Services are stateless, that is either return the requested value or give an exception hence minimizing resource use.

Service granularity
: A principle to ensure services have an adequate size and scope. The functionality provided by the service to the user must be relevant.

Service normalization
: Services are decomposed or consolidated (normalized) to minimize redundancy. In some, this may not be done. These are the cases where performance optimization, access, and aggregation are required.[16]

Service composability
: Services can be used to compose other services.

Service discovery
: Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.

Service reusability
: Logic is divided into various services, to promote reuse of code.

Service encapsulation
: Many services which were not initially planned under SOA, may get encapsulated or become a part of SOA.

Patterns

Each SOA building block can play any of the three roles:

Service provider
: It creates a web service and provides its information to the service registry. Each provider debates upon a lot of hows and whys like which service to expose, which to give more importance: security or easy availability, what price to offer the service for and many more. The provider also has to decide what category the service should be listed in for a given broker service[17] and what sort of trading partner agreements are required to use the service.

Service broker, service registry or service repository
: Its main functionality is to make the information regarding the web service available to any potential requester. Whoever implements the broker decides the scope of the broker. Public brokers are available anywhere and everywhere but private brokers are only available to a limited amount of public. UDDI was an early, no longer actively supported attempt to provide Web services discovery.

Service requester/consumer

It 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.

The service consumer–provider relationship is governed by a standardized service contract,[18] which has a business part, a functional part and a technical part.

Service composition patterns have two broad, high-level architectural styles: choreography and orchestration. Lower level enterprise integration patterns that are not bound to a particular architectural style continue to be relevant and eligible in SOA design.[19][20][21]

Implementation approaches

Service-oriented architecture can be implemented with web services or Microservices.[22] This is done to make the functional building-blocks accessible over standard Internet protocols that are 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.[23]

Implementers commonly build SOAs using web services standards. One example is SOAP, which has gained broad industry acceptance after recommendation of Version 1.2 from the W3C[24] (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, also implement SOA using any other service-based technology, such as Jini, CORBA, REST, or gRPC.

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

- Web services based on WSDL and SOAP
- Messaging, e.g., with ActiveMQ, JMS, RabbitMQ
- RESTful HTTP, with Representational state transfer (REST) constituting its own constraints-based architectural style
- OPC-UA
- WCF (Microsoft's implementation of Web services, forming a part of WCF)
- Apache Thrift
- gRPC
- 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. SOA enables the development of applications that are built by combining loosely coupled and interoperable services.

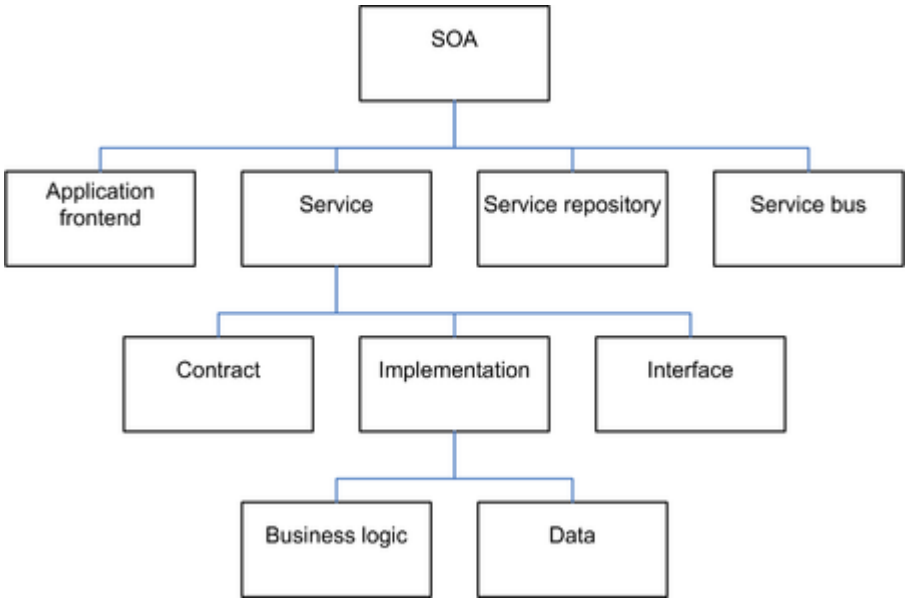
These services inter-operate based on a formal definition (or contract, e.g., WSDL) that is independent of the underlying platform and programming language. The interface definition hides the implementation of the language-specific service. SOA-based systems can therefore function independently of

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.[25]

High-level programming 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.

Service-oriented modeling is an SOA framework that identifies the various disciplines that guide SOA practitioners 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.

Organizational benefits



Elements of SOA, by Dirk Krafzig, Karl Banke, and Dirk Slama[26]

Some enterprise architects believe that SOA can help businesses respond more quickly and more cost-effectively to changing market conditions.[27] 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.

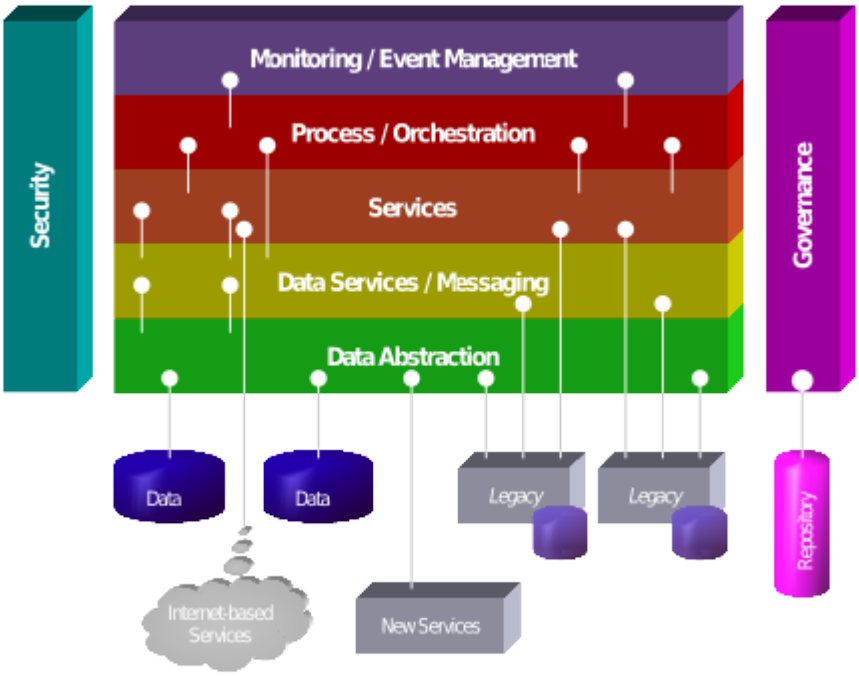
With SOA, the idea is that an organization can look at a problem holistically. A business has more overall control. Theoretically there would not be a mass of developers using whatever tool sets might please them. But rather they would be coding to a standard that is set within the business. They can 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".[28]

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 embracing a SOA approach, such systems can position themselves to stress

the importance of well-defined, highly inter-operable interfaces. Other predecessors of SOA include Component-based software engineering and Object-Oriented Analysis and Design (OOAD) of remote objects, for instance, in CORBA.

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.

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



SOA meta-model, The Linthicum Group, 2007

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.[29]
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.

SOA promises to simplify testing indirectly. 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 'ossjsa.pdf' file within JSR-89 exemplifies such a file.[30]

Criticism

SOA has been conflated with Web services;[31] however, Web services are only one option to implement the patterns that comprise the SOA style. 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 or JSON. 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.[32][33][34]

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.[35] Ultimately, some critics feel that SOA services are still too constrained by applications they represent.[36]

A primary challenge faced by service-oriented architecture is managing of metadata. Environments based on SOA include many services which communicate among each other to perform tasks. Due to the fact that the design may involve multiple services working in conjunction, an Application may generate millions of messages. Further services may belong to different organizations or even competing firms creating a huge trust issue. Thus SOA governance comes into the scheme of things.[37]

Another major problem faced by SOA is the lack of a uniform testing framework. There are no tools that provide the required features for testing these services in a service-oriented architecture. The major causes of difficulty are:[38]

- Heterogeneity and complexity of solution.
- Huge set of testing combinations due to integration of autonomous services.
- Inclusion of services from different and competing vendors.
- Platform is continuously changing due to availability of new features and services.

Extensions and variants

Event-driven architectures

Application programming interfaces

Application programming interfaces (APIs) are the frameworks through which developers can interact with a web application.

Web 2.0

Tim O'Reilly coined the term "Web 2.0" to describe a perceived, quickly growing set of web-based applications.[39] A topic that has experienced extensive coverage involves the relationship between Web 2.0 and service-oriented architectures.

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.[40]

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.[40]

Microservices

Microservices are a modern interpretation of service-oriented architectures used to build distributed software systems. Services in a microservice architecture[41] are processes that communicate with each other over the network in order to fulfill a goal. These services use technology agnostic protocols,[42] which aid in encapsulating choice of language and frameworks, making their choice a concern internal to the service. Microservices are a new realisation and implementation approach to SOA, which have become popular since 2014 (and after the introduction of DevOps), and which also emphasize continuous deployment and other agile practices.[43]

There is no single commonly agreed definition of microservices. The following characteristics and principles can be found in the literature:

- fine-grained interfaces (to independently deployable services),
- business-driven development (e.g. domain-driven design),
- IDEAL cloud application architectures,
- polyglot programming and persistence,
- lightweight container deployment,
- decentralized continuous delivery, and
- DevOps with holistic service monitoring.

Service-oriented architectures for interactive applications

Interactive applications requiring real-time response times, for example low-latency interactive 3d applications, are using specific service oriented architectures addressing the specific needs of such kind of applications. These include for example low-latency optimized distributed computation and communication as well as resource and instance management.[44][45][46]

See also

- Application programming interface
- Loose coupling
- OASIS SOA Reference Model
- Service granularity principle
- SOA governance

- Software architecture
- Service-oriented communications (SOC)
- Service-oriented development of applications
- Service-oriented distributed applications

References

1. "SOA Source Book - What Is SOA?" (https://collaboration.opengroup.org/projects/soa-book/pages.php?action=show&ggid=1314). *collaboration.opengroup.org*. Retrieved March 30, 2021.
2. "Chapter 1: Service Oriented Architecture (SOA)" (https://web.archive.org/web/20170707052149/https://msdn.microsoft.com/en-us/library/bb833022.aspx). *msdn.microsoft.com*. Archived from the original (https://msdn.microsoft.com/en-us/library/bb833022.aspx) on July 7, 2017. Retrieved September 21, 2016.
3. "Service-Oriented Architecture Standards - The Open Group" (https://publications.opengroup.org/standards/soa). *www.opengroup.org*.
4. "What Is SOA?" (https://web.archive.org/web/20160819141303/http://opengroup.org/soa/source-book/soa/soa.htm). *www.opengroup.org*. Archived from the original (http://www.opengroup.org/soa/source-book/soa/soa.htm) on August 19, 2016. Retrieved September 21, 2016.
5. Velte, Anthony T. (2010). *Cloud Computing: A Practical Approach*. McGraw Hill. ISBN 978-0-07-162694-1.
6. "Migrating to a service-oriented architecture, Part 1" (https://web.archive.org/web/20081209120916/http://www-128.ibm.com/developerworks/library/ws-migratesoa/). December 9, 2008. Archived from the original on December 9, 2008. Retrieved September 21, 2016.
7. Michael Bell (2008). "Introduction to Service-Oriented Modeling". *Service-Oriented Modeling: Service Analysis, Design, and Architecture* (https://archive.org/details/serviceorientedm00bell). Wiley & Sons. p. 3 (https://archive.org/details/serviceorientedm00bell/page/n23). ISBN 978-0-470-14111-3.
8. Michael Bell (2010). *SOA Modeling Patterns for Service-Oriented Discovery and Analysis* (https://archive.org/details/soamodelingpatte00bell). Wiley & Sons. p. 390 (https://archive.org/details/soamodelingpatte00bell/page/n413). ISBN 978-0-470-48197-4.
9. "SOA Manifesto" (http://www.soa-manifesto.org/). *www.soa-manifesto.org*. Retrieved September 21, 2016.
10. Thomas Erl (June 2005). *About the Principles*. Serviceorientation.org
11. "Application Platform Strategies Blog: SOA is Dead; Long Live Services" (https://web.archive.org/web/20090115205704/http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html). Apsblog.burtongroup.com. January 5, 2009. Archived from the original (http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html) on January 15, 2009. Retrieved August 13, 2012.
12. Yvonne Balzer Improve your SOA project plans (http://www-128.ibm.com/developerworks/webservices/library/ws-improvesoa/), *IBM*, July 16, 2004
13. Microsoft Windows Communication Foundation team (2012). "Principles of Service Oriented Design" (http://msdn.microsoft.com/en-us/library/bb972954.aspx). *msdn.microsoft.com*. Retrieved September 3, 2012.
14. Principles by Thomas Erl of SOA Systems Inc. eight specific service-orientation principles (http://soaprinciples.com)
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 (https://doi.org/10.1109%2FICCSIT.2009.5235004). ISBN 978-1-4244-4519-6. S2CID 14980694 (https://api.semanticscholar.org/CorpusID:14980694).

16. 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 (https://doi.org/10.1109%2FSCC.2004.1358011). ISBN 978-0-7695-2225-8. S2CID 13156128 (https://api.semanticscholar.org/CorpusID:13156128).2004
17. Duan, Yucong; Narendra, Nanjangud; Du, Wencai; Wang, Yongzhi; Zhou, Nianjun (2014). "Exploring Cloud Service Brokering from an Interface Perspective". *2014 IEEE International Conference on Web Services*. IEEE. pp. 329–336. doi:10.1109/ICWS.2014.55 (https://doi.org/10.1109%2FICWS.2014.55). ISBN 978-1-4799-5054-6. S2CID 17957063 (https://api.semanticscholar.org/CorpusID:17957063).
18. Duan, Yucong (2012). "A Survey on Service Contract". *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE. pp. 805–810. doi:10.1109/SNPD.2012.22 (https://doi.org/10.1109%2FSNPD.2012.22). ISBN 978-1-4673-2120-4. S2CID 1837914 (https://api.semanticscholar.org/CorpusID:1837914).
19. Olaf Zimmermann, Cesare Pautasso, Gregor Hohpe, Bobby Woolf (2016). "A Decade of Enterprise Integration Patterns". *IEEE Software*. **33** (1): 13–19. doi:10.1109/MS.2016.11 (https://doi.org/10.1109%2FMS.2016.11).
20. Rotem-Gal-Oz, Arnon (2012). *SOA Patterns*. Manning Publications. ISBN 978-1933988269.
21. Julisch, Klaus; Suter, Christophe; Woitalla, Thomas; Zimmermann, Olaf (2011). "Compliance by design – Bridging the chasm between auditors and IT architects" (http://soadecisions.org/download/ComplianceByDesign-AAM.pdf) (PDF). *Computers & Security*. **30** (6–7): 410–426. CiteSeerX 10.1.1.390.3652 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.390.3652). doi:10.1016/j.cose.2011.03.005 (https://doi.org/10.1016%2Fj.cose.2011.03.005).
22. Brandner, M., Craes, M., Oellermann, F., Zimmermann, O., Web Services-Oriented Architecture in Production in the Finance Industry, Informatik-Spektrum 02/2004, Springer-Verlag, 2004
23. "www.ibm.com" (http://www.ibm.com/support/knowledgecenter/en/SSEQTP_6.1.0/com.ibm.websphere.base.iseries.doc/info/iseries/ae/cwbs_soawbs.html). Retrieved September 10, 2016.
24. "SOAP Version 1.2 の公開について (W3C 勧告)" (http://www.w3.org/2003/06/soap12-pressrelease) (in Japanese). W3.org. Retrieved August 13, 2012.
25. Okishima, Haruhiru (2006). ". "Case Study of System Architecture that use COBOL assets" " (http://www.fujitsu.com/global/documents/about/resources/publications/fstj/archives/vol42-3/paper18.pdf) (PDF).
26. *Enterprise SOA*. Prentice Hall, 2005
27. Christopher Koch A New Blueprint For The Enterprise (http://www.cio.com.au/index.php/id;1350140708), *CIO Magazine*, March 1, 2005
28. Elizabeth Millard (January 2005). "Building a Better Process". *Computer User*. Page 20.
29. Brayan Zimmerli (November 11, 2009) Business Benefits of SOA (https://web.archive.org/web/20101105063545/http://www.brayan.com/projects/BenefitsOfSOA/default.htm), *University of Applied Science of Northwestern Switzerland, School of Business*
30. JSR-000089 OSS Service Activation API Specification 1.0 Final Release (https://web.archive.org/web/20110726070810/https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=7854-oss_service_activation-1.0-fr-spec-oth-JSpec%40CDS-CDS_Developer). sun.com
31. Joe McKendrick. "Bray: SOA too complex; 'just vendor BS' " (https://www.zdnet.com/blog/service-oriented/bray-soa-too-complex-just-vendor-bs/597). ZDNet.
32. Jimmy Zhang (February 20, 2008) "Index XML Documents with VTD-XML" (http://xml.sys-con.com/read/453082.htm) Archived (https://web.archive.org/web/20080704164141/http://xml.sys-con.com/read/453082.htm) July 4, 2008, at the Wayback Machine. *XML Journal*.
33. Jimmy Zhang (August 5, 2008) "i-Technology Viewpoint: The Performance Woe of Binary XML" (http://soa.sys-con.com/read/250512.htm). *Microservices Journal*.

34. Jimmy Zhang (January 9, 2008) "Manipulate XML Content the Ximple Way" (http://www.devx.com/xml/Article/36379). *devx.com*.
35. "The Reason SOA Isn't Delivering Sustainable Software" (http://www.jpmorgenthal.com/morgenthal/?p=31). jpmorgenthal.com. June 19, 2009. Retrieved June 27, 2009.
36. "SOA services still too constrained by applications they represent" (https://www.zdnet.com/article/soa-services-still-too-constrained-by-applications-they-represent/). zdnet.com. June 27, 2009. Retrieved June 27, 2009.
37. "Governance Layer" (https://www.opengroup.org/soa/source-book/soa_refarch/governance.htm). *www.opengroup.org*. Retrieved September 22, 2016.
38. "How to Efficiently Test Service Oriented Architecture | WSO2 Inc" (http://wso2.com/library/articles/2014/04/how-to-efficiently-test-service-oriented-architecture/). *wso2.com*. Retrieved September 22, 2016.
39. "What Is Web 2.0" (http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html). Tim O'Reilly. September 30, 2005. Retrieved June 10, 2008.
40. Christoph Schroth; Till Janner (2007). "Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services" (http://www.alexandria.unisg.ch/Publikationen/37270). *IT Professional*. **9** (3): 36–41. doi:10.1109/MITP.2007.60 (https://doi.org/10.1109%2FMITP.2007.60). S2CID 2859262 (https://api.semanticscholar.org/CorpusID:2859262). Retrieved February 23, 2008.
41. Dragoni, Nicola; Giallorenzo, Saverio; Alberto Lluch Lafuente; Mazzara, Manuel; Montesi, Fabrizio; Mustafin, Ruslan; Safina, Larisa (2016). "Microservices: yesterday, today, and tomorrow". arXiv:1606.04036v1 (https://arxiv.org/abs/1606.04036v1) [cs.SE (https://arxiv.org/archive/cs.SE)].
42. James Lewis and Martin Fowler. "Microservices" (http://martinfowler.com/articles/microservices.html).
43. Balalaie, A.; Heydarnoori, A.; Jamshidi, P. (May 1, 2016). "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture" (http://spiral.imperial.ac.uk/bitstream/10044/1/40557/8/SO_SWSI-2015-10-0149.R1_Balalaie.pdf) (PDF). *IEEE Software*. **33** (3): 42–52. doi:10.1109/MS.2016.64 (https://doi.org/10.1109%2FMS.2016.64). hdl:10044/1/40557 (https://hdl.handle.net/10044%2F1%2F40557). ISSN 0740-7459 (https://www.worldcat.org/issn/0740-7459). S2CID 18802650 (https://api.semanticscholar.org/CorpusID:18802650).
44. Frank Glinka; Allaithy Raed (2009). "A Service-Oriented Interface for Highly Interactive Distributed Applications" (https://link.springer.com/chapter/10.1007/978-3-642-14122-5_31). *European Conference on Parallel Processing*. Lecture Notes in Computer Science. **6043**: 266–277. doi:10.1007/978-3-642-14122-5_31 (https://doi.org/10.1007%2F978-3-642-14122-5_31). ISBN 978-3-642-14121-8. Retrieved February 9, 2021.
45. Dieter Hildebrandt; Jan Klimke (2011). "Service-oriented interactive 3D visualization of massive 3D city models on thin clients" (https://dl.acm.org/doi/10.1145/1999320.1999326). *COM.Geo '11: Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications*: 1. doi:10.1145/1999320.1999326 (https://doi.org/10.1145%2F1999320.1999326). ISBN 9781450306812. S2CID 53246415 (https://api.semanticscholar.org/CorpusID:53246415). Retrieved February 9, 2021.
46. Mahy Aly; Michael Franke (2016). "Service Oriented Interactive Media (SOIM) Engines Enabled by Optimized Resource Sharing" (https://ieeexplore.ieee.org/document/7473030). *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*: 231–237. doi:10.1109/SOSE.2016.47 (https://doi.org/10.1109%2FSOSE.2016.47). hdl:1854/LU-7215326 (https://hdl.handle.net/1854%2FLU-7215326). ISBN 978-1-5090-2253-3. S2CID 9511734 (https://api.semanticscholar.org/CorpusID:9511734). Retrieved February 9, 2021.
