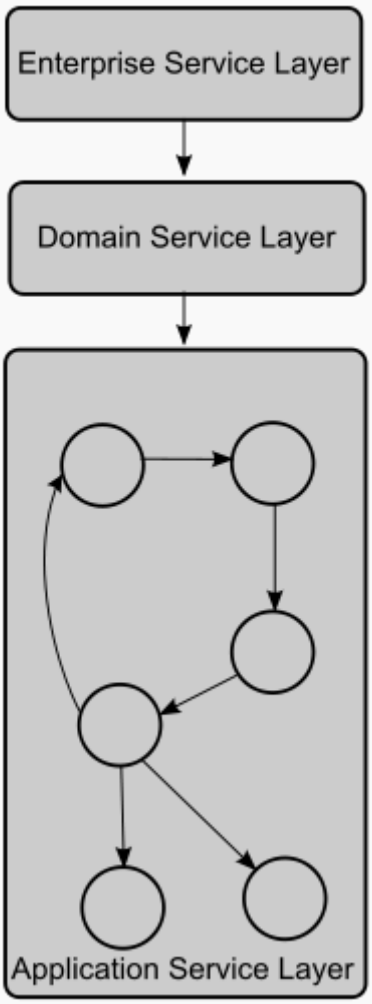
Service-oriented architecture

From Wikipedia, the free encyclopedia



Layer interaction in service-oriented Architecture

Service-oriented architecture (**SOA**) is a flexible set of design principles used during the phases of systems development and integration in computing. A system based on a SOA will package functionality as a suite of interoperable *services* that can be used within multiple separate systems from several business domains.

SOA also 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 understood, well defined interface to access them. XML is commonly used for interfacing with SOA services, though this is not required.

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 underlies applications. SOA separates functions into distinct units, or services,[1] 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.[2]

One can consider SOA a continuum, as opposed to distributed computing or modular programming.

Contents

Description

Overview

The SOA implementations rely on a mesh of software services. Services comprise unassociated, loosely coupled units of functionality that have no calls to each other embedded in them. Each service implements one action, such as filling out an online application for an account, or viewing an online bank statement, or placing an online booking or airline ticket order. Rather than services embedding calls to each other in their source code, they use defined protocols that describe how services pass and parse messages using description metadata.

SOA developers associate individual SOA objects by using orchestration. In the process of orchestration the developer associates software functionality (the services) in a non-hierarchical arrangement using a software tool that contains a complete list of all available services, their characteristics, and the means to build an application utilizing these sources.

Underlying and enabling all of this requires metadata in sufficient detail to describe not only the characteristics of these services, but also the data that drives them. Programmers have made extensive use of XMLin SOA to structure data that they wrap in a nearly exhaustive description-container. Analogously, the Web Services Description Language (WSDL) typically describes the services themselves, while the SOAPprotocol describes the communications protocols. Whether these description languages are the best possible for the job, and whether they will become/remain the favorites in the future, remain open questions. As of 2008 SOA depends on data and services that are described by metadata that should meet the following two criteria:

1. The metadata should come 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 autodiscovery of services without modifying the functional contract of a service.

2. The metadata should come in a form that system designers can understand and manage with a reasonable expenditure of cost and effort.

SOA aims to allow users to string together fairly large chunks of functionality to form *ad hoc* applications that are built almost entirely from existing software services. The larger the chunks, the fewer the interface points 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. The great promise of SOA suggests that the marginal cost of creating the n-th application is low, as all of the software required already exists to satisfy the requirements of other applications. Ideally, one requires only orchestration to produce a new application.

For this to operate, no interactions must exist between the chunks specified or within the chunks themselves. Instead, humans specify the interaction of services (all of them unassociated peers) in a relatively *ad hoc* way with the intent driven by newly emergent requirements. Thus the need for services as much larger units of functionality than traditional functions or classes, lest the sheer complexity of thousands of such granular objects overwhelm the application designer. Programmers develop the services themselves using traditional languages like Java, C, C++, C#, Visual Basic, COBOL, or PHP.

SOA services feature loose coupling, in contrast to the functions that a linker binds together to form an executable, to a dynamically linked library or to an assembly. SOA services also run in "safe" wrappers (such as Java or .NET) and in other programming languages that manage memory allocation and reclamation, allow *ad hoc* and late binding, and provide some degree of indeterminate data typing.

As of 2008, increasing numbers of third-party software companies offer software services for a fee. In the future, SOA systems may consist of such third-party services combined with others created in-house. This has the potential to spread costs over many customers and customer uses, and promotes standardization both in and across industries. In particular, the travel industry now has a well-defined and documented set of both services and data, sufficient to allow any reasonably competent software engineer to create travel-agency software using entirely off-the-shelf software services.[3] Other industries, such as the finance industry, have also started making significant progress in this direction.

SOA as an architecture relies on service-orientation as its fundamental design principle.[4] If a service presents a simple interface that abstracts away its underlying complexity, users can access independent services without knowledge of the service's platform implementation.[5]

Requirements

In order to efficiently use a SOA, the architecture must meet the following requirements:

- Interoperability among different systems and programming languages that provides the basis for integration between applications on different platforms through a communication protocol. One example of such communication depends on the concept of messages. Using messages across defined message channels decreases the complexity of the end application, thereby allowing the developer of the application to focus on true application functionality instead of the intricate needs of a communication protocol.
- Desire to create a federation of resources. Establish and maintain data flow to a Federated database system. This allows new functionality developed to reference a common business format for each data element.

Principles

The following **guiding principles** define the ground rules for development, maintenance, and usage of the SOA[6]:

- reuse, granularity, modularity, composability, componentization and interoperability.
- standards-compliance (both common and industry-specific).
- services identification and categorization, provisioning and delivery, and monitoring and tracking.

The first publicly published research of service-orientation from an industry perspective was provided by Thomas Erl of SOA Systems Inc. who defined eight specific service-orientation principlescommon to all primary SOA platforms. These principles were published in “Service-Oriented Architecture: Concepts, Technology, and Design”, on the www.soaprinciples.com research site, and in the September 2005 edition of the Web Services Journal (see Service-orientation).

- **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.
- **Service Statelessness** - Services minimize resource consumption by deferring the management of state information when necessary
- **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.

Some authors also include the following principles:

- **Service Optimization** – All else 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** – It refers 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 that the right of a consumer to access the service. Often, the idea of service virtualization, 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, also relates to location transparency.

The following references provide additional considerations for defining a SOA implementation:

- SOA Reference Architectureprovides a working design of an enterprise-wide SOA implementation with detailed architecture diagrams, component descriptions, detailed requirements, design patterns, opinions about standards, patterns on regulation compliance, standards templates etc.[7]
- Life cycle management SOA Practitioners Guide Part 3: Introduction to Services Lifecycleintroduces the services lifecycle and provides a detailed process for services management through the service lifecycle, from inception to retirement or repurposing of the services. It also contains an appendix that includes organization and governance best-practices, templates, comments on key SOA standards, and recommended links for more information.

In addition, one might take the following factors into account when defining a SOA implementation:

- Efficient use of system resources
- Service maturity and performance
- EAI (Enterprise Application Integration)

Web services approach

Web services can implement a service-oriented architecture. Web services 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, then bind it with respective service and then use it. They can access multiple services if the service provides multiple services.

SOA and Web service protocols

Implementors commonly build SOAs using web services standards (for example, SOAP) that have gained broad industry acceptance. 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.

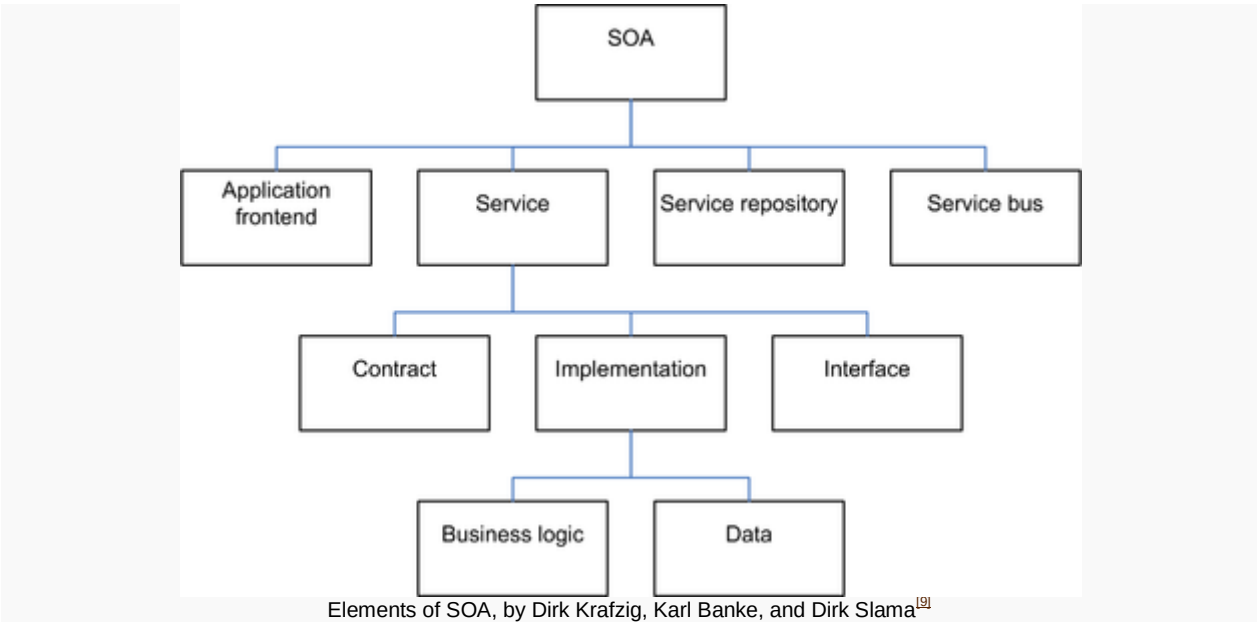
Other SOA concepts

Architectures can operate independently of specific technologies.[8] Designers can implement SOA using a wide range of technologies, including:

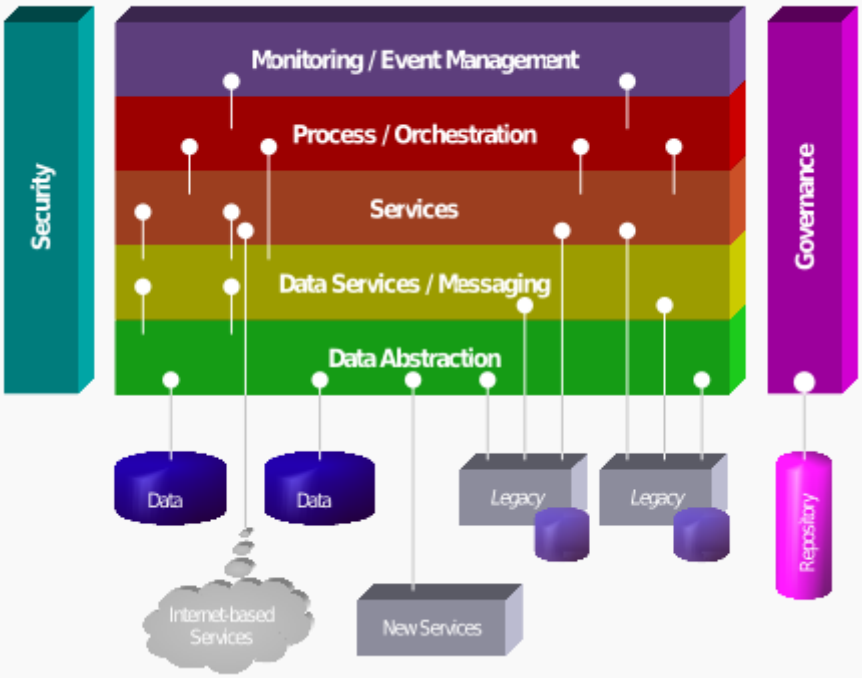
- SOAP, RPC
- REST
- DCOM
- CORBA
- Web Services
- DDS
- WCF (Microsoft's implementation of Web Services now forms a part of WCF)

Implementations can use one or more of these protocols and, for example, might use a file-system mechanism to communicate data conforming to 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.

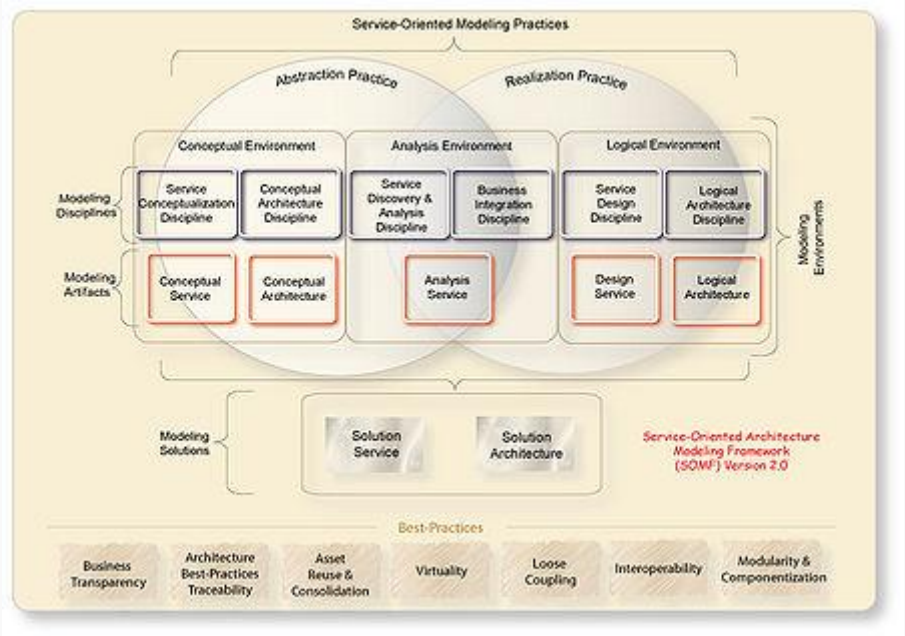
Many implementers of SOA have begun to adopt an evolution of SOA concepts into a more advanced[*citation needed*] architecture called SOA 2.0.



Elements of SOA, by Dirk Krafzig, Karl Banke, and Dirk Slama[9]



SOA meta-model, The Linthicum Group, 2007



Service-Oriented Modeling Framework (SOMF) Version 2.0

SOA enables the development of applications that are built by combining loosely coupled and interoperable services.[10]

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. This has extended the useful life of many core legacy systems indefinitely, no matter what language they originally used.

SOA can support integration and consolidation activities within complex enterprise systems, but SOA does not specify or provide a methodology or framework for documenting capabilities or services.

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 orportals[citation needed].

As of 2008 researchers have started investigating the use of Service Component Architecture (SCA) to implement SOA.

Service-oriented modeling [1] 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.

SOMF addresses the following principles:

- business traceability
- architectural best-practices traceability
- technological traceability
- SOA value proposition
- software assets reuse
- SOA integration strategies
- technological abstraction and generalization
- architectural components abstraction

SOA definitions

Commentators have provided multiple definitions of SOA. The OASIS group[11] and the Open Group[12] have both created formal definitions.

OASIS defines SOA as the following:

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.

Programmatic service contract

A service contract needs[citation needed] to have the following components:

- Header
  - Name – Name of the service. This should indicate in general terms what the service does, not just its definition
  - Version – The version of this service contract
  - Owner – The person/team in charge of the service
  - Responsibility assignment (RACI)
    - Responsible – The role/person/team responsible for the deliverables of this contract/service. All versions of the contract
    - Accountable – Ultimate Decision Maker in terms of this contract/service
    - Consulted – Whom one must consult before action is taken on this contract/service. This is two-way communication. These people have an impact on the decision or the execution of that decision.
    - Informed – Who must be informed that a decision or action is being taken. This is a one-way communication. These people are impacted by the decision or execution of that decision, but have no control over the action.
  - Type – This is the type of the service: to help distinguish the layer in which it resides. Different implementations will have different service types. Examples of service types include:
    - Presentation
    - Process

        - Business
        - Data
        - Integration
- Functional
    - Functional Requirement (from Requirements Document) – Indicates the functionality in specific bulleted items — what exactly this service accomplishes. The language should encourage test cases to prove the functionality is accomplished.
    - Service Operations – Methods, actions etc. Must be defined in terms of what part of the functionality it provides.
    - Invocation – Indicates how to invoke the service. This includes the URL, interface, etc. There may be multiple invocation paths for the same service. One may have the same functionality for an internal and some external clients, each with different invocation means and interfaces. Examples:
        - SOAP
        - REST
        - Events Triggers
- Non-Functional
    - Security Constraints – Defines who can execute this service in terms of roles or individual partners etc. and which invocation mechanism they can invoke.
    - Quality of Service – Determines the allowable failure rate
    - Transactional – Is this capable of acting as part of a larger transaction and if so, how do we control that?
    - Service Level Agreement – Determines the amount of latency the service is allowed to have to perform its actions
    - Semantics – Dictates or defines the meaning of terms used in the description and interfaces of the service
    - Process – Describes the process, if any, of the contracted service

SOA and network management architecture

As of 2008 the principles of SOA are being applied to the field of network management. Examples of service-oriented network management architectures include TS 188 001 *NGN Management OSS Architecture* from ETSI, and M.3060 *Principles for the Management Of Next Generation Networks* recommendation from the ITU-T.

Tools for managing SOA infrastructure include:

- HP Software & Solutions
- HyPerformix IPS Performance Optimizer
- IBM Tivoli Framework
- Red Hat JBoss Operations Network

Discussion

Benefits

Some enterprise architects believe that SOA can help businesses respond more quickly and cost-effectively to changing market-conditions.[13] 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.

In some respects, one can regard SOA 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 has taken place of solutions that use truly static bindings to talk to other equipment in the network. By formally embracing a SOA approach, such systems can position themselves to stress the importance of well-defined, highly inter-operable interfaces.[14]

Some have questioned whether SOA simply revives concepts like modular programming (1970s), event-oriented design (1980s) or interface/component-based design (1990s)[*citation needed*]. SOA promotes the goal of separating users (consumers) from the service implementations. Services can therefore be run on various distributed platforms and be accessed across networks. This can also maximize reuse of services[*citation needed*].

SOA realizes its business and IT benefits by utilizing an analysis and design methodology when creating services. This methodology ensures that services remain consistent with the architectural vision and roadmap, and that they adhere to principles of service-orientation. Arguments supporting the business and management aspects from SOA are outlined in various publications.[15]

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 sub-ordinate services.

Services generally adhere to the following principles of service-orientation:[16]

- abstraction
- autonomy
- composability
- discoverability
- formal contract
- loose coupling
- reusability
- statelessness

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.[17]
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. The industry has never been exposed to this circumstance before.[*citation needed*]

If an organization possesses appropriate 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 are 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.[18]

Challenges in adopting SOA

One obvious and common challenge faced involves managing **services metadata**[*citation needed*]. 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, and 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. SOA's main goal is to deliver *Agility to Businesses*[*citation needed*]. Therefore it is important to invest in a testing framework (build or buy) 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).[19]

Another challenge relates to providing **appropriate levels of security**. Security models built into an application may no longer suffice when an application exposes its capabilities as services that can be used by other applications. That is, application-managed

security is not the right model for securing services. A number of new technologies and standards have started to emerge and provide more appropriate models for security in SOA.

As SOA and the WS-* specifications practitioners expand, update and refine their output, they encounter a shortage of skilled people to work on SOA-based systems, including the integration of services and construction of services infrastructure.

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.[20] 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 faster 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.[21][22]

Internal IT delivery organizations routinely initiate SOA efforts, and some of these improperly introduce concepts to the business[*citation needed*] so it remains misunderstood. The adoption starts meeting IT delivery needs instead of those of the business, resulting in an organization with, for example, superlative laptop provisioning services, instead of one that can quickly respond to market opportunities. Businessleadership also becomes convinced that the organization is executing well on SOA.

One of the most important benefits of SOA is its ease of reuse. Therefore accountability and funding models must ultimately evolve within the organization.[*citation needed*] A business unit needs to be encouraged to create services that other units will use. Conversely, units must be encouraged to reuse services. This requires a few new governance components:

- Each business unit creating services must have an appropriate support structure in place to deliver on its service-level obligations, and to support enhancing existing services strictly for the benefit of others. This is typically quite foreign to business leaders.
- Each business unit consuming services accepts the apparent risk of reusing services outside their own control, with the attendant external project dependencies, etc.
- An innovative funding model is needed as incentive to drive these behaviors above.[*citation needed*] Business units normally pay the IT Organization to assist during projects, and then to operate the environment. Corporate incentives should discount these costs to service providers, and create internal revenue-streams from consuming business units to the service provider.[*citation needed*] These streams should be less than the costs of a consumer simply building it the old-fashioned way. This is where SOA deployments can benefit from the SaaS monetization architecture.[23]

Criticisms of SOA

Some criticisms[24] of SOA depend on conflating SOA with Web services. For example, some critics claim SOA results in the addition of XML layers, introducing XML parsing and composition. In the absence of native or binary forms of Remote Procedure Call (RPC), applications could run slower 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) 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.[25][26][27]

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.[28] Ultimately, some critics feel that SOA services are still too constrained by applications they represent.[29]

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.

Some critics regard SOA as merely an obvious evolution of currently well-deployed architectures (open interfaces, etc.).

IT system designs sometimes overlook the desirability of modifying systems readily. Many systems, including SOA-based systems, hard-code the operations, goods and services of the organization, thus restricting their online service and business agility in the global marketplace.[*citation needed*]

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.

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.[30] 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". [1]

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 translated to nine languages.

Roadmap

The Roadmap to SOA involves:

- Understanding the industry specific end state for the first iteration
- Gather the current state of Business process maintenance, Applications, Integrations, Security, Master Data and Governance
- Reducing the SOA readiness gap between various entities

The various entities would be in one of the SOA readiness states.

Business Process Maintenance

- Ad-hoc: Processes are defined as and when needed
- Documented: Business steps are documented in disparate documents
- Structured: Process management tools used albeit disconnected
- Services: Processes designed as services through standardized process management tool

Integration

- Point to Point: Applications directly interface with required parties
- Loosely Coupled: Some applications use an intermediary that abstracts the interface
- Static Services: Services available but bound to other Static Applications
- Dynamic Binding: Static Services bound in a dynamic way through Pub-Sub
- Dynamic Services: Services that can dynamically sense when they are required and respond appropriately.

Applications

- Standalone: Applications requiring large interface efforts
- Service Wrapped: Application(s) wrapped as a service
- Exposed API's: The Applications have service based API's
- Exposed Services: This makes the applications reusable
- Dynamic Discovery: Applications can be discovered by other applications

Security

- Application level Authentication and Authorization: Every application is accessed in its own Silo
- Loose Authentication control: Some applications are on LDAP
- Tight directory controlled authentication: SSO enabled across most applications
- Centrally controlled Identity and Access Management

Master Data

- Application Controlled: All applications control their master data
- Partial sharing: Some applications share master data but not all masterdata
- Centralized Master Data Maintenance: Most master data is shared and maintained through some MDM implementation

Extensions

SOA, Web 2.0, services over the messenger, and mashups

Web 2.0, a perceived "second generation" of web activity, primarily features the ability of visitors to contribute information for collaboration and sharing. Web 2.0 applications often use REST-ful web services and commonly feature AJAX based user interfaces, utilizing web syndication, blogs, and wikis. While there are no set standards for Web 2.0, it is characterized by building on the existing Web server architecture and using services. Web 2.0 can therefore be regarded as displaying some SOA characteristics.[31][32][33]

Some commentators also regard mashups as Web 2.0 applications. The term "Business Mashups" has been coined to describe web applications that combine content from more than one source into an integrated user experience that shares many of the characteristics of service-oriented business applications (SOBAs). SOBAs are applications composed of services in a declarative manner. There is ongoing debate about "the collision of Web 2.0, mashups, and SOA," with some stating that Web 2.0 applications are a realization of SOA composite and business applications.[34]

Web 2.0

Tim O'Reilly coined the term "Web 2.0" to describe a perceived, quickly-growing set of web-based applications.[35] A topic that has experienced extensive coverage involves the relationship between Web 2.0 and Service-Oriented Architectures (SOAs). SOA is considered as 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.[32]

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.[32]

In an "Internet of Services", all people, machines, and goods will have access via the network infrastructure of tomorrow.[*citation needed*] 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.[36]

Even though Oracle indicates[*citation needed*] that Gartner is coining a new term, Gartner analysts indicate that they call this **advanced SOA** and refer to it as "SOA 2.0".[37] Most of the major middlewarevendors (e. g., Red Hat, webMethods, TIBCO Software, IBM, Sun Microsystems, and Oracle) have had some form of SOA 2.0 attributes for years.[*citation needed*]

Digital Nervous System

SOA implementations have been described as representing a piece of the larger vision known as the Digital Nervous System [38][39] or the Zero Latency Enterprise.[40]

Lean SOA

A methodology to developing SOA employing Lean Software Development principles.[41]

See also

- Amazon Web Services
- Business-Agile Enterprise
- Business-driven development
- Business Intelligence 2.0 (BI 2.0)
- Cloud Computing
- Communications-enabled application (CEA)
- Comparison of business integration software
- Component business model
- Enterprise application integration
- Enterprise Integration Patterns
- Enterprise Messaging System
- Enterprise service bus
- EMML
- Event-driven programming
- Event-driven SOA
- Integration Objects
- Java Business Integration
- Microsoft Connected Services Framework
- Open ESB
- Oracle SOA Suite
- Platform as a Service
- Representational Sta
- Resource oriented ar
- SAP Enterprise Servi
- Search oriented archi
- Semantic service orie
- Service component a
- Service layer
- Service-oriented anal
- Service-oriented arch
- Service-oriented mod
- SOA Governance
- SOA Security
- SOALIB
- Software AG webMet
- Software as a service
- Sun Java CAPS
- Web Oriented Archite

External links

- *InfoWorld* magazine provides a 3-minute videoexplaining SOA and how it works
- Recommendations for beginning an SOA initiativeby Ian Robinson (video)
- The pros and cons of using SOAP/WSDL/WS-* and REST for SOAby Mark Little (video)
- Technology choices and business ramifications around implementing an SOA, an interview with Ian Robinson (video)
- Service-oriented architecture's 6 burning questions

Further reading

- Hündling , J.; Weske, M.: Web Services: Foundation and Composition, in: Electronic Markets, 13, 2, 2003, pp. 108–119.

References

1. ^ [a] [b] Bell, Michael (2008). "Introduction to Service-Oriented Modeling". *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley & Sons. pp. 3. ISBN 978-0-470-14111-3.
2. ^ Bell_, Michael (2010). *SOA Modeling Patterns for Service-Oriented Discovery and Analysis*. Wiley & Sons. pp. 390. ISBN 978-0470481974.
3. ^ http://www.opentravel.org/
4. ^ An alternative view, particularly after initial deployments, denies that SOAs properly ought to dictate physical implementation, so the formal definition should not include "network". High-performance SOAs may not be viable, especially if deployed to distributed nodes on a network. Separate nodes for every (or most) services could become prohibitively expensive.
5. ^ Channabasavaiah, Holley and Tuggle, Migrating to a service-oriented architecture, *IBM DeveloperWorks*, 16 December 2003.
6. ^ Yvonne Balzer Improve your SOA project plans, *IBM*, 16 July 2004
7. ^ SOA Practitioners Guide Part 2: SOA Reference Architecture
8. ^ Erl, Thomas. Serviceorientation.org – About the Principles, 2005-2006
9. ^ *Enterprise SOA*. Prentice Hall, 2005
10. ^ Cardoso, Jorge; Sheth, Amit P. (2006). "Foreword". *Semantic Web Services, Processes and Applications*. SEMANTIC WEB AND BEYOND: Computing for Human Experience. Foreword by Frank Leymann. 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."

11. ^ SOA Reference Model definition
12. ^ SOA – Documents – Document details
13. ^ Christopher Koch A New Blueprint For The Enterprise, *CIO Magazine*, March 1, 2005
14. ^ Bieberstein et al., Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap (The developerWorks Series) (Hardcover), IBM Press books, 2005, 978-0131870024
15. ^ Martin van den berg et al. SOA for Profit, A Manager's Guide to Success with Service-Oriented Architecture (Hardcover), 978-9075414141
16. ^ M. Hadi Valipour, Bavar AmirZafari, Kh. Niki Maleki, Negin Daneshpour, A Brief Survey of Software Architecture Concepts and Service Oriented Architecture, in Proceedings of 2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT'09, pp 34-38, Aug 2009, China.
17. ^ Brayan Zimmerli Business Benefits of SOA, *University of Applied Science of Northwestern Switzerland, School of Business*, 11 November 2009
18. ^ 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@CDS-CDS_Developer — JSR-89 Spec download
19. ^ From the Business Motivation Model(BMM) to SOA, Journal Of Object Technology – November/December 2008
20. ^ WS-I Basic Profile
21. ^ Is There Real Business Value Behind the Hype of SOA?, *Computerworld*, June 19, 2006.
22. ^ See also: WS-MetadataExchange OWL-S
23. ^ The Overlapping Worlds of SaaS and SOA
24. ^ Tim Bray, XML co-founder – http://blogs.zdnet.com/service-oriented/?p=597
25. ^ Index XML documents with VTD-XML
26. ^ The Performance Woe of Binary XML
27. ^ Manipulate XML Content the Ximple Way
28. ^ "The Reason SOA Isn't Delivering Sustainable Software". jpmorgenthal.com. 2009-06-19. Retrieved 2009-06-27.
29. ^ "SOA services still too constrained by applications they represent". zdnet.com. 2009-06-27. Retrieved 2009-06-27.
30. ^ SOA Manifesto Official WebsiteDate Accessed: 02 October 2010.
31. ^ Dion Hinchcliffe Is Web 2.0 The Global SOA?, *SOA Web Services Journal*, 28 October 2005
32. ^ [a] [b] [c] Schroth, Christoph ; Janner, Till; (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 2008-02-23.
33. ^ Hoyer, Volker ; Stanoesvka-Slabeva, Katarina; Janner, Till; Schroth, Christoph; (2008). *Enterprise Mashups: Design Principles towards the Long Tail of User Need*. Proceedings of the 2008 IEEE International Conference on Services Computing (SCC 2008). Retrieved 2008-07-08.
34. ^ Jason Bloomberg Mashups and SOBAs: Which is the Tail and Which is the Dog?, *Zapthink*
35. ^ "What Is Web 2.0". Tim O'Reilly. 2005-09-30. Retrieved 2008-06-10.
36. ^ Ruggaber, Rainer; (2007). *Internet of Services—A SAP Research Vision*. IEEE Computer Society. Retrieved 2008-02-23.
37. ^ Yefim Natis & Roy Schulte Advanced SOA for Advanced Enterprise Projects, *Gartner*, July 13, 2006
38. ^ "From Web to Boarding Area: Delta's SOA is Ready". Retrieved 2009-05-02.
39. ^ "The Value of An Enterprise Architecture". Retrieved 2009-05-02.
40. ^ "Moving Toward the Zero Latency Enterprise". Retrieved 2009-05-02.
41. ^ "Lean Development Applied to SOA". Retrieved 2010-08-16.