WHAT IS SOA? SERVICE-ORIENTED ARCHITECTURE EXPLAINED

Service oriented architecture (SOA) refers to the software architecture design paradigm that allows software components to behave as separate, autonomous, loosely coupled network-accessible units.

The use of SOA is on the rise. Let's take a look at how SOA works—and why businesses are adopting it.

How service oriented architecture works

In SOA, software components function as their own loosely coupled units. These units provide services or data using a network protocol, making them independent of vendors or proprietary technology systems.

These services can be independent, repeatable, and self-contained tasks of a global system functionality—consider them the building blocks of a large consumer service where each feature is composed of multiple small services that can be developed, managed, modified and replaced independently of other components (and services).

(Compare SOA to microservice architecture.)

Service Oriented Architecture allows the flexibility to treat every component independent of the global service that requires those components. This approach solves some of the key challenges facing large enterprise IT systems and has driven the growth and popularity of the SOA design

paradigm.

Most of the drivers are shared across earlier design philosophies like object-oriented programming and component-based engineering, such as:

- Multiple use
- Non-context-specific
- Composable
- Encapsulated
- Components independent deployment and versioning

Before we discuss why it's important to adopt an SOA approach for software and systems design, it's important to understand its characteristics and driving factors. What makes SOA valuable to organizations operating large complex and distributed IT environments?

What is loose coupling?

Let's start with the term loose coupling.

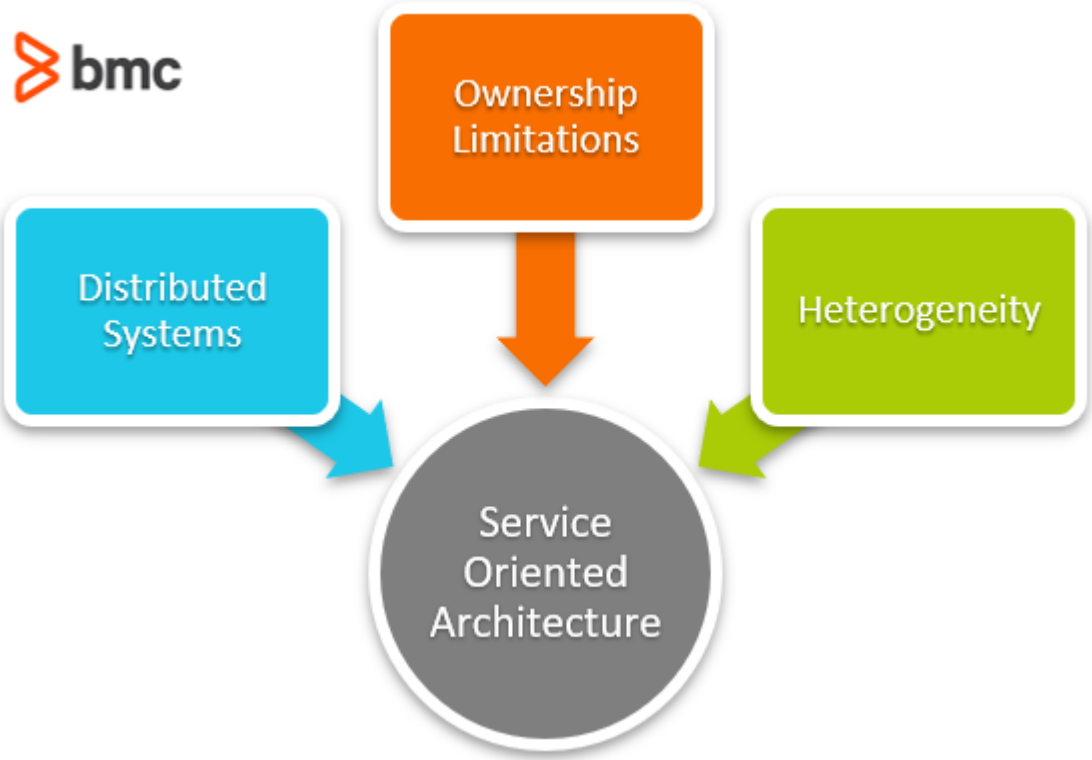
The term "loose coupling" refers to the client of a service, and its ability to remain independent of the service that it requires. The most important part of this concept is that the client, which in itself can be a service, can communicate with the service even if they are not closely related.

This facilitated communication is achieved through the implementation of a specified interface that is able to perform the necessary actions to allow for the transmission of data.

A common example of this increased ability to communicate without service constraints involves coding languages used by these services. There is an array of different languages from which software platforms are created and not all of these languages can interact fluently, without encountering communication issues. By using an SOA, it is not necessary for the client to understand the language that is being used by the service, but instead, it relies on a structured interface that is able to process the transmission between the service and the client.

Drivers of service oriented architecture

The more prevalent factors driving interest and growth of SOA capabilities in the modern software engineering landscape include:



Distributed systems

Modern enterprise IT solutions are built on multiple layers of technology that evolve constantly. New components are integrated and legacy systems are updated, infrastructure resources are provisioned and scaled to meet variable and unpredictable demand.

When the underlying services are loosely coupled, they can be located and communicate with each other through an interface, such as an API, over the network using standardized protocols of the OSI model. These protocols are supported by open source and proprietary technology vendors alike.

Designing the software architecture specifically with the openness and standardization to interact with a variety of services is a necessary imperative for large-scale distributed networks.

Ownership limitations

Business organizations subscribe to cloud-based services for the convenience of provisioning hardware resources *without* doing any of the heavy lifting. Cloud solutions are operated and managed by third-party vendors while customers access the service through a Web interface.

These customers must also ensure that the cloud service interacts with their existing systems and with their data assets without technical limitations such as:

- Integration
- Performance
- Standardization issues

Cloud vendors, on the other hand, can only offer limited control and visibility into the hardware components that power their cloud services.

The conflict in ownership domains of these underlying components is a driving factor for services in distributed systems to interact with each other, without requiring ownership and control over those

components. Customers of cloud services cannot modify the behavior of the cloud infrastructure.

Similarly, it's important for vendors to modify small system components, without necessarily modifying the system-wide functionality.

Heterogeneity

Large, distributed, and complex systems inherently lack harmony. Many systems are developed in different times—some evolve and replace, some are maintained as legacy systems. Old programming languages and platforms do not maintain the same high level of support or popularity over the course of their entire lifecycle.

The Service Oriented Architecture supports this behavior—allowing organizations to adopt agile design methodologies. Heterogeneity of the entire architecture itself is not the goal of SOA, but it ensures practices such as:

- Vendor diversity
- Agnostic platforms
- Programming languages

When the diverse services are interoperable, organizations can avoid vendor lock-in and establish independent services that can be leveraged without having to modify or control the underlying components and services.

There is no doubt that as web application technologies continue to evolve, more businesses will utilize the power of SOA. By switching to a standardized communication protocol, engineers will be able to create software applications without having to worry about the languages on which platforms are built, and can instead rely upon the interoperability that the SOA structure creates.

Finally, SOA can help ensure that applications can be easily scaled, while at the same time decreasing the costs that are often encountered when developing business service solutions.

Related reading

- BMC DevOps Blog
- The Role of Microservices in DevOps
- 15 Best Practices for Building a Microservices Architecture
- What Is a Container Pipeline?
- The Software Development Lifecycle (SDLC): An Introduction
- The State of DevOps Today: A Report Roundup