Service-Oriented Architecture

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 product, vendor or technology.

SOA just makes it easier for software components over various networks to work with each other.

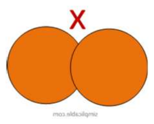
Web services which are built as per the SOA architecture tend to make web service more independent. The web services themselves can exchange data with each other and because of the underlying principles on which they are created, they don't need any sort of human interaction and also don't need any code modifications. It ensures that the web services on a network can interact with each other seamlessly.

SOA Principles

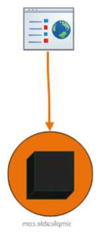
SOA is based on some key principles which are mentioned below

1. **Standardized Service Contract** - Services adhere to a service description. A service must have some sort of description which describes what the service is about. This makes it easier for client applications to understand what the service does.

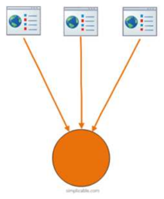
2. **Loose Coupling** – Less dependency on each other. This is one of the main characteristics of web services which just states that there should be as less dependency as possible between the web services and the client invoking the web service. So if the service functionality changes at any point in time, it should not break the client application or stop it from working.



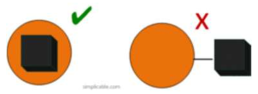
3. **Service Abstraction** - Services hide the logic they encapsulate from the outside world. The service should not expose how it executes its functionality; it should just tell the client application on what it does and not on how it does it.



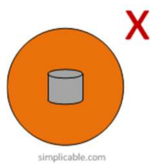
4. **Service Reusability** - Logic is divided into services with the intent of maximizing reuse. In any development company re-usability is a big topic because obviously one wouldn't want to spend time and effort building the same code again and again across multiple applications which require them. Hence, once the code for a web service is written it should have the ability work with various application types.



5. **Service Autonomy** - Services should have control over the logic they encapsulate. The service knows everything on what functionality it offers and hence should also have complete control over the code it contains.

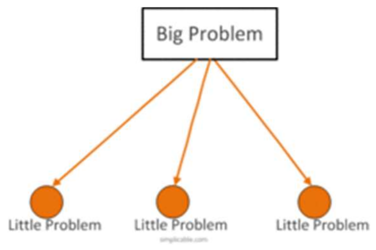


6. **Service Statelessness** - Ideally, services should be stateless. This means that services should not withhold information from one state to the other. This would need to be done from either the client application. An example can be an order placed on a shopping site. Now you can have a web service which gives you the price of a particular item. But if the items are added to a shopping cart and the web page navigates to the page where you do the payment, the responsibility of the price of the item to be transferred to the payment page should not be done by the web service. Instead, it needs to be done by the web application.

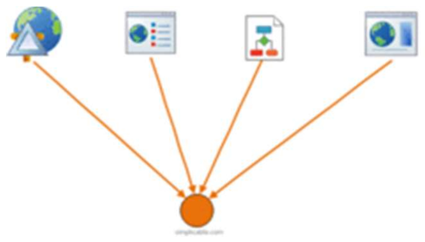


7. **Service Discoverability** - Services can be discovered (usually in a service registry). We have already seen this in the concept of the UDDI (*Universal Description, Discovery and Integration*), which performs a registry which can hold information about the web service.

8. **Service Composability** - Services break big problems into little problems. One should never embed all functionality of an application into one single service but instead, break the service down into modules each with a separate business functionality.



9. **Service Interoperability** - Services should use standards that allow diverse subscribers to use the service. In web services, standards as XML and communication over HTTP is used to ensure it conforms to this principle.



Characteristics

The service-orientation design principles help in distinguishing a service-oriented solution[14] from a traditional object-oriented solution by promoting distinct design characteristics. The presence of these characteristics in a service-oriented solution greatly improves the chances of realizing the aforementioned goals and benefits. Thomas Erl has identified four service-orientation characteristics as follows:

- Vendor-neutral
- Business-driven
- Enterprise-centric
- Composition-centric

A vendor-neutral service-oriented solution helps to evolve the underlying technology architecture in response to ever changing business requirements. By not being dependent on a particular vendor, any aging infrastructure could be replaced by more efficient technologies without the need for redesigning the whole solution from scratch. This also helps in creating a heterogeneous technology environment where particular business automation requirements are fulfilled by specific technologies.

Service Oriented Architecture

Within a SOA, the development of solution logic is driven by the needs of the business and is designed in a manner that focuses on the long-term requirements of the business. As a result, the technology architecture is more aligned with the business needs.

Unlike traditional silo-based application development, a SOA takes into account the requirements of either the whole of the enterprise or at least some considerable part of it. As a result, the developed services are interoperable and reusable across the different segments of the enterprise.

A service-oriented solution enables to deal with new and changing requirements, within a reduced amount of time, by making use of existing services. The services are designed in a manner so that they can be recomposed i.e. become a part of different solutions.

WCF Bindings

Bindings specify the communication mechanism to use when talking to an endpoint and indicate how to connect to an endpoint. A binding contains the following elements:

- The protocol stack determines the security, reliability, and context flow settings to use for messages that are sent to the endpoint.
- The transport determines the underlying transport protocol to use when sending messages to the endpoint, for example, TCP or HTTP.
- The encoding determines the wire encoding to use for messages that are sent to the endpoint. For example, text/XML, binary, or Message Transmission Optimization Mechanism (MTOM).

The following bindings ship with WCF:

Binding	**Configuration Element**	**Description**
BasicHttpBinding	<basicHttpBinding>	A binding that is suitable for communicating with WS-Basic Profile-conformant Web services, for example, ASP.NET Web services (ASMX)-based services. This binding uses HTTP as the transport and text/XML as the default message encoding.
WSHttpBinding	<wsHttpBinding>	A secure and interoperable binding that is suitable for non-duplex service contracts.
WSDualHttpBinding	<wsDualHttpBinding>	A secure and interoperable binding that is suitable for duplex service contracts or communication through SOAP intermediaries.
WSFederationHttpBinding	<wsFederationHttpBinding>	A secure and interoperable binding that supports the WS-Federation protocol, which enables organizations that are in a federation to efficiently authenticate and authorize users.
NetHttpBinding	<netHttpBinding>	A binding designed for consuming HTTP or WebSocket services that uses binary encoding by default.
NetHttpsBinding	<netHttpsBinding>	A secure binding designed for consuming HTTP or WebSocket services that uses binary encoding by default.
NetTcpBinding	<netTcpBinding>	A secure and optimized binding suitable for cross-machine communication between WCF applications.

NetNamedPipeBinding	<netNamedPipeBinding>	A secure, reliable, optimized binding that is suitable for on-machine communication between WCF applications.
NetMsmqBinding	<netMsmqBinding>	A queued binding that is suitable for cross-machine communication between WCF applications.
NetPeerTcpBinding	<netPeerTcpBinding>	A binding that enables secure, multiple machine communication.
MsmqIntegrationBinding	<msmqIntegrationBinding>	A binding that is suitable for cross-machine communication between a WCF application and existing Message Queuing applications.
BasicHttpContextBinding	<basicHttpContextBinding>	A binding suitable for communicating with WS-Basic Profile conformant Web services that enables HTTP cookies to be used to exchange context.
NetTcpContextBinding	<netTcpContextBinding>	A secure and optimized binding suitable for cross-machine communication between WCF applications that enables SOAP headers to be used to exchange context.
WebHttpBinding	<webHttpBinding>	A binding used to configure endpoints for WCF Web services that are exposed through HTTP requests instead of SOAP messages.
WSHttpContextBinding	<wsHttpContextBinding>	A secure and interoperable binding suitable for non-duplex service contracts that enables SOAP headers to be used to exchange context.
UdpBinding	<udpBinding>	A binding to use when sending a burst of simple messages to a large number of clients simultaneously.