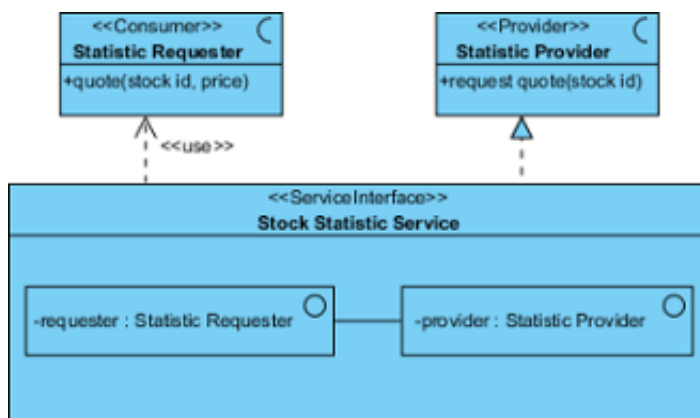

How to draw a Service Interface Diagram in SoaML

[SoaML](#) supports both a contract and an interface-based approach to SOA. They differ in the way services are specified.

The interface-based approach involves the use of simple interfaces and service interface. Simple interface focuses mainly on one-way service delivery that requires no protocol between parties. Service interface allows for bi-directional services. Provider and consumer work together to complete a service.

Service interface diagram is a type of SoaML diagram specialized for the definition and specification of both simple interface and service interface.






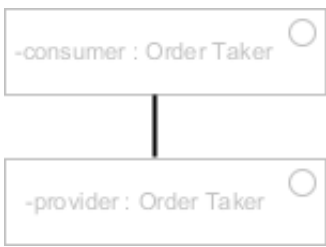
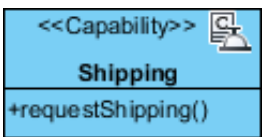
Creating service interface diagram


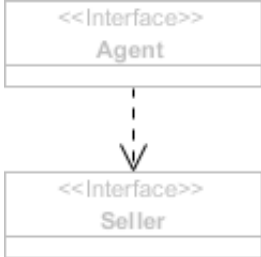
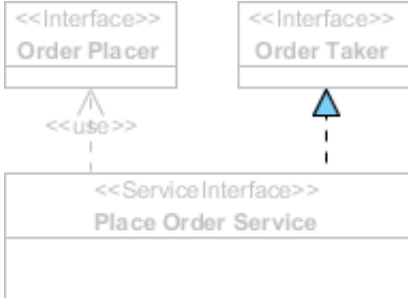
1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Service Interface Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

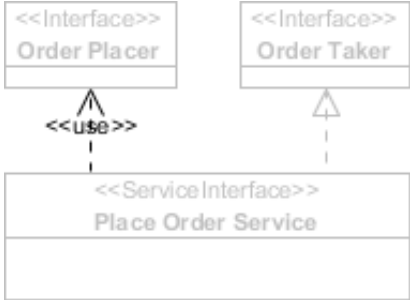
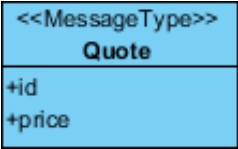
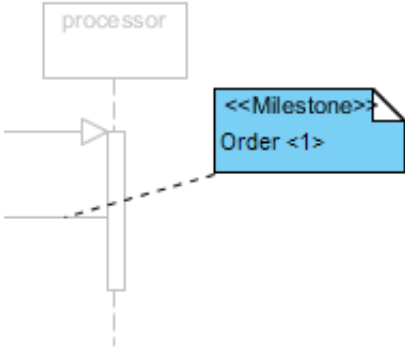
Notations

The description of notations is either extracted or derived from the OMG SoaML Specification v1.0.1.

Name	Representation	Description
Service Interface		A ServiceInterface defines the interface and responsibilities of a participant to provide or consume a service. It is used as the type of a Service or Request Port. A ServiceInterface is the means for specifying how a participant is to interact to

Name	Representation	Description
Interface	 <pre> classDiagram class StockQuote { <<Interface>> } </pre>	<p>provide or consume a Service. A ServiceInterface may include specific protocols, commands, and information exchange by which actions are initiated and the result of the real world effects are made available as specified through the functionality portion of a service. A ServiceInterface may address the concepts associated with ownership, ownership domains, actions communicated between legal peers, trust, business transactions, authority, delegation, etc.</p> <p>Simple interfaces define one-way services that do not require a protocol. Such services may be defined with only a single UML interface and then provided on a "Service" port and consumed on a "Request" port.</p>
Role	 <pre> classDiagram class OrderTaker { -consumer : Order Taker } </pre>	<p>A ServiceInterface is a UML Class. It defines specific roles for each participant plays in the service interaction. These roles have a name and an interface type. The interface of the provider (which must be the type of one of the parts in the class) is realized (provided) by the ServiceInterface class. The interface of the consumer (if any) must be used by the class.</p>
Connector	 <pre> classDiagram class OrderTaker { -consumer : Order Taker } class OrderTaker { -provider : Order Taker } OrderTaker -- OrderTaker </pre>	<p>Connect roles in a service interface.</p>
Capability	 <pre> classDiagram class Shipping { <<Capability>> +requestShipping() } </pre>	<p>A Capability models the ability to act and produce an outcome that achieves a result that may provide a service specified by a ServiceContract</p>

Name	Representation	Description
		<p>or ServiceInterface irrespective of the Participant that might provide that service. A ServiceContract alone, has no dependencies or expectation of how the capability is realized – thereby separating the concerns of 'what' vs. "how." The Capability may specify dependencies or internal process to detail how that capability is provided including dependencies on other Capabilities. Capabilities are shown in context using a service dependencies diagram.</p>
Expose	 <pre> classDiagram class Orders["<<ServiceInterface>>\nOrders"] class Marketing["<<Capability>>\nMarketing"] Orders -.-> Marketing : <<Expose>> </pre>	<p>The Expose dependency provides the ability to indicate what Capabilities that are required by or are provided by a participant should be exposed through a Service Interface.</p>
Dependency	 <pre> classDiagram class Agent["<<Interface>>\nAgent"] class Seller["<<Interface>>\nSeller"] Agent -.-> Seller </pre>	<p>The Provider may also have a uses dependency on the consumer interface, representing the fact that the provider may call the consumer as part of a bi-directional interaction. These are also known as "callbacks" in many technologies.</p>
Realization	 <pre> classDiagram class OrderPlacer["<<Interface>>\nOrder Placer"] class OrderTaker["<<Interface>>\nOrder Taker"] class PlaceOrderService["<<ServiceInterface>>\nPlace Order Service"] OrderPlacer .. > PlaceOrderService : <<use>> OrderTaker .. > PlaceOrderService </pre>	<p>A ServiceInterface specifies the receptions and operations it receives through InterfaceRealizations. A ServiceInterface can realize any number of Interface. Some platform specific models may restrict the number of realized interfaces to at most one.</p>
Usage		<p>A ServiceInterface specifies its required needs through Usage dependences to Interfaces.</p>

Name	Representation	Description
		
Message Type		<p>A MessageType is a kind of value object that represents information exchanged between participant requests and services. This information consists of data passed into, and/or returned from, the invocation of an operation or event signal defined in a service interface. A MessageType is in the domain or service-specific content and does not include header or other implementation or protocol-specific information.</p>
Milestone		<p>A Milestone depicts progress by defining a signal that is sent to an abstract observer. The signal contains an integer value that intuitively represents the amount of progress that has been achieved when passing a point attached to this Milestone.</p>