WHAT IS: JAVA
By Matthew Tyson, Java Developer, JavaWorld
MAR 12, 2020 10:26 AM PDT

About
Everything you need to know about Java programming tools and APIs, with code and examples.

FEATURE

What is service-oriented architecture?

Learn the key characteristics of SOA and SOAP-based web services

Service-oriented architecture (SOA) emerged in the early part of this century as an evolution of distributed computing. Before SOA, *services* were understood as the end result of the application development process. In SOA, the application itself is composed of services. Services can be delivered individually or combined as components in a larger, composite service.

Services interact over the wire using a protocol such as REST or SOAP (Simple Object Access Protocol). Services are *loosely coupled*, meaning the service interface is independent of the underlying implementation. Developers or system integrators can compose one or more services into an application without necessarily knowing how each service is implemented.

This article is an overview of Java SOA and the key characteristics of a service-oriented architecture implemented using SOAP-based web services. I'll also briefly compare SOA and microservices and discuss the difference between RESTful and SOAP-based web services in Java.

SOA and web services

SOA and web services are frequently conflated, but they are not the same thing. SOA is an architecture that allows developers to combine multiple application services into a larger, composite service. SOA can be implemented using SOAP-based web services or REST APIs, or sometimes a combination of both. It's important to understand that in SOA, a *service* is any remotely available resource that can respond to requests. A *web service* is implemented using specific protocols.

Why service-oriented architecture?

SOA addresses three common enterprise challenges:

- Respond quickly to business changes.
- Leverage existing infrastructure investments.
- Support new channels of interaction with customers, partners, and suppliers.

Enterprise infrastructure is heterogeneous across operating systems, applications, system software, and application infrastructure. As a result, many enterprise systems are comprised of complex and inconsistent applications delivering a range of interdependent services. Existing applications running current business processes are critical, so starting from scratch or modifying them is a delicate proposition. But enterprises must be able to modify and expand technical infrastructure to meet business demands.

SOA and microservices

Microservices is an architectural style evolved from SOA. Both are distributed architectures and both offer a decoupled paradigm. Whereas service-oriented architecure is heavier on infrastructure, microservices offers a more flexible, lightweight development style. There are pros and cons to both, and both are widely used. Generally speaking, SOA is more frequently implemented or maintained by established enterprises that have the resources to support more formality. Microservices often appeals to new or growing organizations that require agility.

As compared to a monolithic architecture, SOA's loosely coupled nature makes it relatively seamless to plug in new services or upgrade existing services for new business requirements. It also provides the option to make services consumable across different channels, and to expose legacy applications as services, thereby safeguarding infrastructure investments.

Because they are loosely coupled, SOA components can be changed with minimal impact to other components. Components can also be added to the architecture in a standardized fashion, and they can be scaled to to address load.

[Learn Java from beginning concepts to advanced design patterns in this comprehensive 12-part course!]

As an example, consider how an enterprise might use a set of existing applications to create a new, composite supply-chain application. While the existing applications are heterogeneous and distributed across various systems, their functionality is exposed and accessed using standard interfaces.

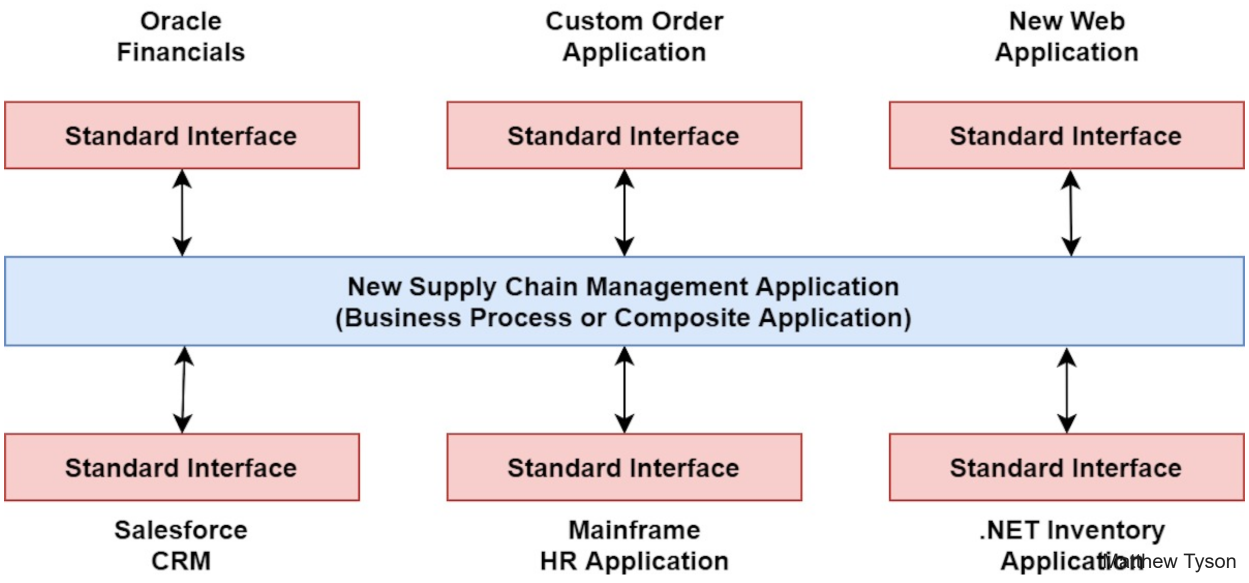

Figure 1. Supply-chain application in a service-oriented architecture

Key characteristics of SOA

SOA can be as simple as a single component consuming services provided by another component or as sophisticated as a range of components interacting via an enterprise service bus such as MuleSoft's ESB. No matter what the scale, the key to a successful SOA implementation is to use as little complexity as possible to achieve your aims. Your first and last question should always be: *Does this design satisfy our business requirements?*

Regardless of scale or complexity, the pattern of a service-oriented architecture is more or less the same:

- Service providers expose endpoints and describe the available actions at each endpoint.
- Service consumers issues requests and consume responses.
- Service providers generate messages to handle requests.

Implementing service-oriented architecture

To implement SOA you start with the basic service architecture, then provide the infrastructure, meaning protocols and other tools that enable communication and interoperability. Figure 2 shows a diagram of a typical service architecture.

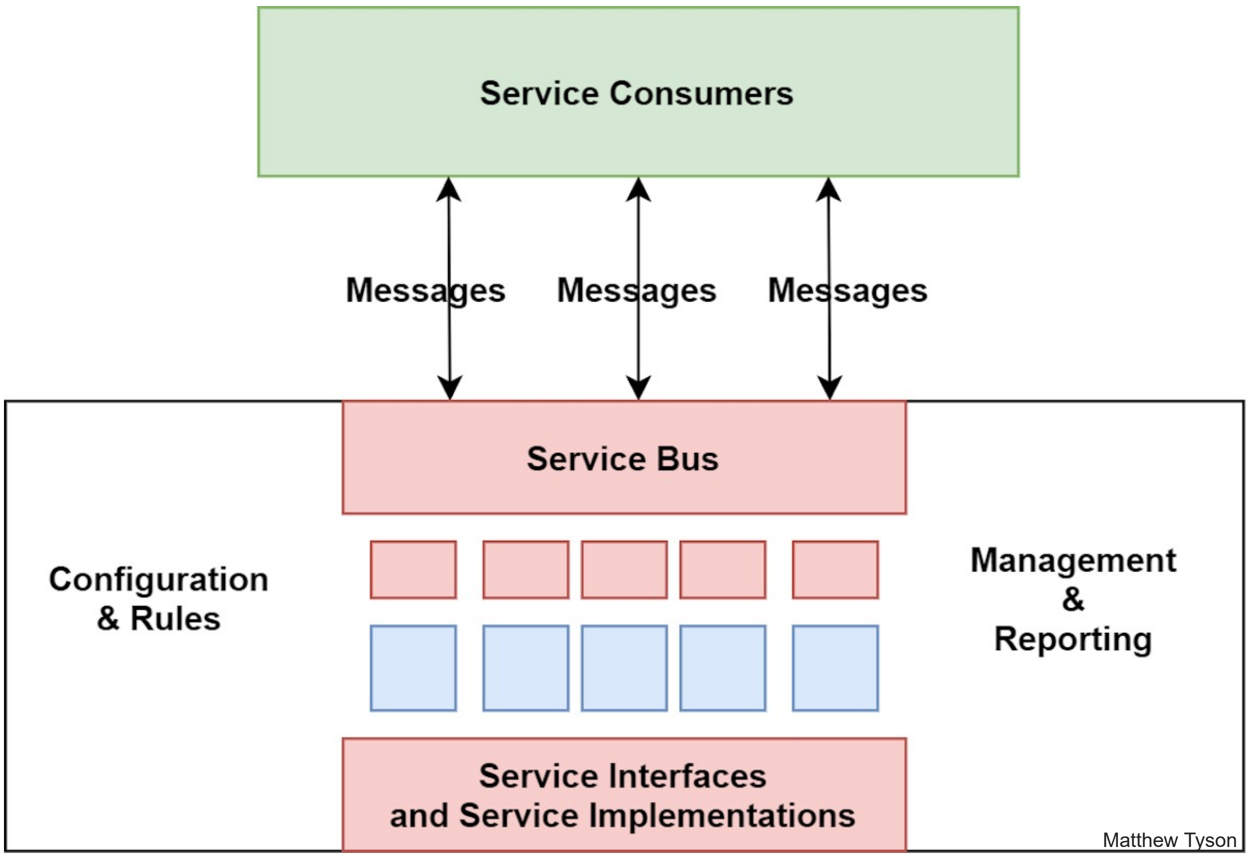

Figure 2. A sample service architecture

In this diagram, three consumers invoke services by sending messages to an enterprise service bus, which transforms and routes the messages to an appropriate service implementation. A *business rules engine* incorporates business rules in a service or across services. A *service management layer* manages activities like auditing, billing, and logging.

Components in this architecture are loosely coupled, so they can be switched out or updated with relatively minimal impact to the application as a whole. This gives the enterprise flexibility to add or update business processes as needed. For the most part, changes to individual services should not greatly affect other services.

SOAP vs RESTful web services

It is possible to adopt the SOA style and implement it with REST, for instance using the JAX-RS API or Spring Boot Actuator, but that discussion is out of scope for this article. See "SOAP vs REST vs JSON" for a useful comparison of SOAP vs RESTful web services. There is also some overlap between RESTful web services and the more lightweight style associated with microservices.

SOAP-based web services

Web services implemented using SOAP are still more rigid than a RESTful web services or microservices implementation, but far more flexible than the early days of SOA. Here we'll just look at the high-level protocols required for SOAP-based web services.

SOAP, WSDL, and XSD

SOAP, WSDL, and XSD are the fundamental infrastructure of a SOAP-based web service implementation. WSDL is used to describe the service, and SOAP is the transport layer for sending messages between service consumers and providers. Services communicate with messages formally defined using XML Schema (XSD). You can think of WSDL as the service's interface (loosely analogous to a Java interface). The implementation is done in Java classes, and communication across the network happens via SOAP. Functionally, a consumer would look for a service, get the WSDL for that service, then invoke the service using SOAP.

Web service security

The WS-I Basic Profile 2.0 specification addresses message security. This specification focuses on credential exchange, message integrity, and message confidentiality.

Web service discovery

Once the cornerstone of web service discovery, UDDI (Universal Description, Definition and Integration) has faded into history. Today it's common to expose a SOAP-based web service the way you would any other service, via an endpoint URL. As an example, you could use the JAX-WS Service Endpoint Interface and its `@WebService` and `@WebMethod` annotations.

Building and deploying web services

Java developers have several options for building and deploying SOAP-based web services, including Apache Axis2 and Spring-WS; however, the Java standard is JAX-WS, the Java API for XML Web Services. The core idea behind JAX-WS is to create Java classes and annotate them to create the required artifacts. Under the hood, JAX-WS uses several Java packages, including JAXB, a general purpose library for binding Java classes to XML.

JAX-WS hides the underlying complexity and protocols from the developer, thus streamlining the process of defining and deploying Java-based SOAP services. Modern Java IDEs like Eclipse include full support for developing JAX-WS web services. The JAX-WS specification has also been selected for ongoing development in Jakarta EE.

Conclusion

Service-oriented architecture implemented with SOAP-based web services requires more rigid and formal service definitions than RESTful web services or microservices. However, some larger organizations continue to favor the more formal style enforced by SOAP. Many large-scale legacy systems are also built on SOAP, and some B2B and internal systems choose SOAP-based web services for their more formally defined API contracts. Whether you are developing or maintaining a large-scale enterprise system, understanding the SOA pattern and being able to evaluate your options for implementing it will serve you well in your programming career.