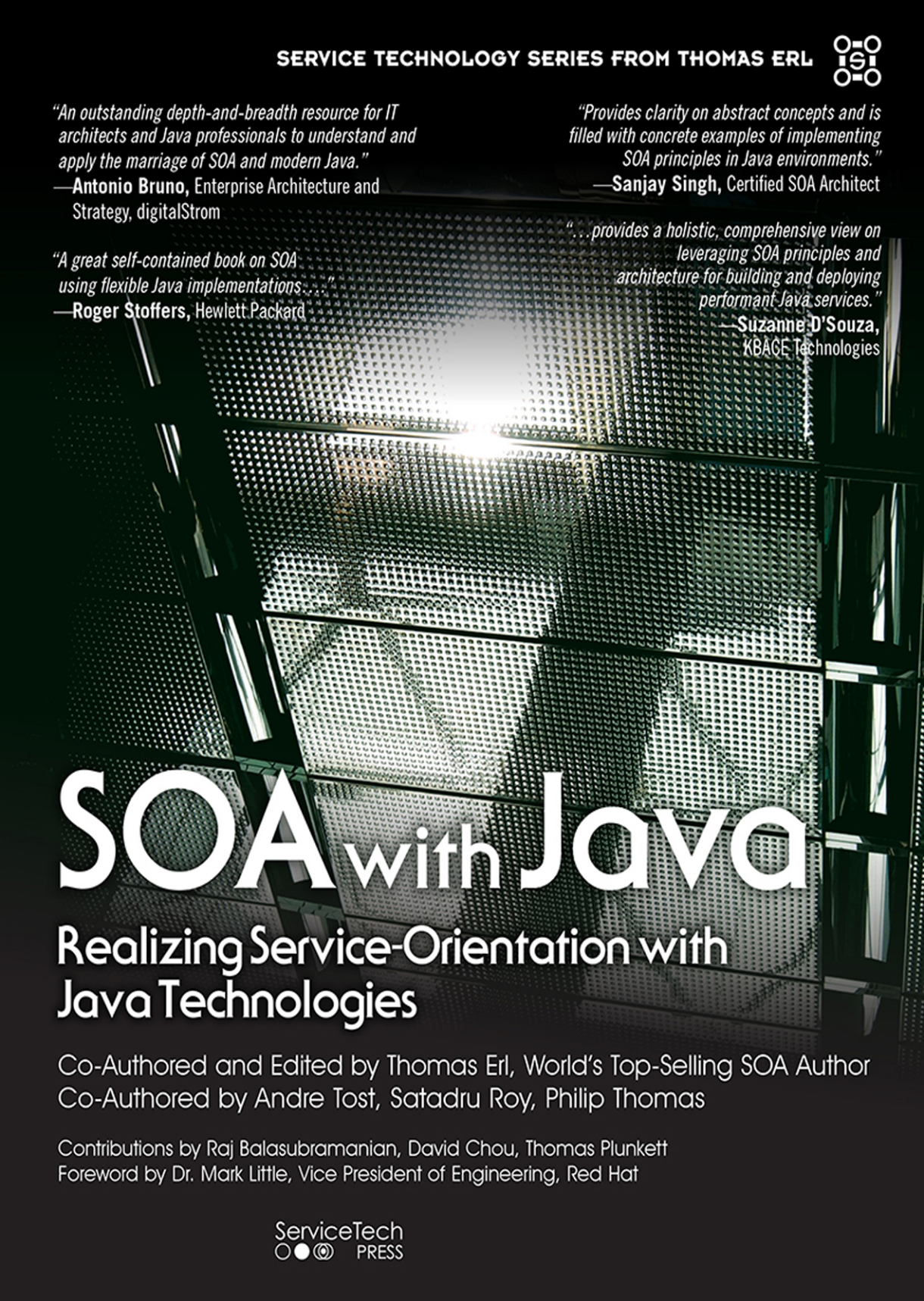
SERVICE TECHNOLOGY SERIES FROM THOMAS ERL



*"An outstanding depth-and-breadth resource for IT architects and Java professionals to understand and apply the marriage of SOA and modern Java."*
—**Antonio Bruno,** Enterprise Architecture and Strategy, digitalStrom

*"A great self-contained book on SOA using flexible Java implementations…."*
—**Roger Stoffers,** Hewlett Packard

*"Provides clarity on abstract concepts and is filled with concrete examples of implementing SOA principles in Java environments."*
—**Sanjay Singh,** Certified SOA Architect

*"…provides a holistic, comprehensive view on leveraging SOA principles and architecture for building and deploying performant Java services."*
—**Suzanne D'Souza,** KBACE Technologies

# SOA with Java

## Realizing Service-Orientation with Java Technologies

Co-Authored and Edited by Thomas Erl, World's Top-Selling SOA Author
Co-Authored by Andre Tost, Satadru Roy, Philip Thomas

Contributions by Raj Balasubramanian, David Chou, Thomas Plunkett
Foreword by Dr. Mark Little, Vice President of Engineering, Red Hat

ServiceTech PRESS

# SOA with Java

*Realizing Service-Orientation with Java Technologies*

Thomas Erl, Andre Tost,
Satadru Roy, and Philip Thomas


PRENTICE HALL
UPPER SADDLE RIVER, NJ • BOSTON • INDIANAPOLIS • SAN FRANCISCO
NEW YORK • TORONTO • MONTREAL • LONDON • MUNICH • PARIS • MADRID
CAPE TOWN • SYDNEY • TOKYO • SINGAPORE • MEXICO CITY

## 7.2 Standardized Service Contract

A foundational criterion in service-orientation is that a service have a well-defined and standardized contract. When building Web services with SOAP and WS-*, portable machine-readable service contracts are mandatory between different platforms as WSDL documents and associated XML schema artifacts describing the service data model. The finite set of widely used HTTP verbs for REST services form an implicit service contract. However, describing the entity representations for capture in a portable and machine-independent format is the same as SOAP and WS-*.

For REST services, capturing and communicating various aspects of resources can be necessary, such as the set of resources, relationships between resources, HTTP verbs allowed on resources, and supported resource representation formats. Standards, such as WADL, can be used to satisfy the mandatory requirements. Having a standards-based service contract exist separate from the service logic, with service data entities described in a platform-neutral and technology-neutral format, constitutes a service by common definition. Even the self-describing contract of HTTP verbs for a REST service establishes a standards-based service contract. Recall the standards used for service contracts, such as WSDL/WADL and XML Schema, from Chapter 5.

### Top-Down vs. Bottom-Up

Ensuring that services are business-aligned and not strictly IT-driven is necessary when identifying services for a service portfolio in an SOA. Services are derived from a decomposition of a company's core business processes and a collection of key business entities. For a top-down approach, services are identified and interfaces are designed by creating appropriate schema artifacts to model either the operating data and WSDL-based service contracts, or model REST resources and resource methods. The completed service interface is implemented in code.

However, enterprises can have irreplaceable mission-critical applications in place. Therefore, another aspect of finding services is assessing existing applications and components to be refactored as services for a bottom-up approach. This includes creating standard service contracts, such as WSDL definitions or REST resource models, for the existing components.

Tooling provides support for both approaches in a Java world. For SOAP-based Web services, tools play a more prominent role than in Java-based REST services. JAX-WS defines the wsimport tool, which takes an existing WSDL definition as input to generate Java skeletons. These skeletons can be used as the starting point for implementing the

actual service logic. Similarly, the wsgen tool generates WSDL from existing Java code. The mapping between WSDL/XML schema and Java is an important function associated with the wsimport tool.

Machine-readable contracts are also necessary for REST services. JAX-RS, if WADL is not used, starts with a resource model to implement the resources in Java. Consider the contract as a logical collection of the resource model, with the supported resource methods, resource representations, and any hyperlinks embedded in the representations allowing navigability between resources. If WADL is used, tools like wadl2java can generate code artifacts. Initiatives exist to help generate WADL from annotated JAX-RS classes for a bottom-up approach, although these recent developments can have limited usefulness.

Some SOA projects will employ both a bottom-up and a top-down approach to identify and design services and service contracts, which often results in a meet-in-the-middle approach. Service definitions and Java interfaces are tuned and adjusted until a good match is found.

Sometimes an XML schema definition developed as part of the service design cannot map well into Java code. Conversely, existing Java code may not easily map into an XML schema. Java code that does not precisely map to a service interface designed as part of a top-down approach can exist. In this case, the Service Façade pattern can be applied to insert a thin service wrapper to satisfy the service interface and adapt incoming and outgoing data to the format supported by the existing Java code.

### Mapping Between Java and WSDL

WSDL is the dominant method of expressing the contract of a Java component. While typically related to Web services, the language can also be utilized for other types of services. Formalization and standardization of the relationship between Java and WSDL has made this possible, such as the work completed on the JAX-RPC standard.

The JAX-RPC standard initially defined basic tasks, such as "a service portType is mapped to a Java interface" and "an operation is mapped to a method." However, formalizing allows the definitions described in the JAX-RPC standard to define how an existing Java component (a class or interface) can generate a WSDL definition, and vice versa. Consequently, most contemporary Java IDEs support generating one from the other without requiring any manual work.

JAX-WS, the successor standard for JAX-RPC, builds on top of its predecessor's definitions and delegates all the issues of mapping between Java and XML to the JAXB specification (as discussed in Chapter 6). These sections serve to highlight some of the issues raised when creating standard service contracts from Java or creating Java skeletons from existing service contracts. The majority of the details explained in the next section apply specifically to Web services.

### Wrapped Document/Literal Contracts

The WSDL standard identifies a variety of styles for transmitting information between a service consumer and service. Most of the styles are specific for the chosen message and network protocol, and specified in a section of the WSDL definition called the binding. A common binding found in a WSDL definition uses SOAP as the message protocol and HTTP as the network transport. Assume that SOAP/HTTP is the protocol used for the services presented as examples.

The portType is a binding-neutral part of a service definition in WSDL that describes the messages that travel in and out of a service. Reusable across multiple protocols, the portType is not bound to the use of a Web service. Any service, even if invoked locally, can be described by a WSDL portType, which allows service interfaces to be defined in a language-neutral fashion regardless of whether the service logic will be implemented in Java or another language.

As discussed in Chapter 5, the WSDL binding information defines the message format and protocol details for Web services. For SOAP-based bindings, two key attributes known as the encoding style and the invocation style determine how messages are encoded and how services are invoked.

The wrapped document/literal style supported by default in all Java environments for services dictates that an exchange should be literal. Literal means that no encoding happens in the message, so the payload of the message is a literal instantiation of the schema descriptions in the `<types>` element of the WSDL. The invocation style is document. Document means that the runtime environment should generate a direct copy of the input and output messages as defined in the portType and not just an arbitrary part of the message. Wrapped means that the payload of the message includes a wrapper element with the same name as the operation invoked.

In order to understand how the WSDL standard relates to Java, let's review Example 7.11 to expose the following class as a service and create a standardized contract.

```
package pack;
import javax.jws.*;
@WebService
public class Echo {
  public String echo(String msg) {
    return msg;
  }
}
```

**Example 7.11**

Using the wrapped document/literal style implements a wrapper element called `"echo"` after the `echo()` method in the public class. Echo is included in the XML schema associated with this service. An excerpt in the resulting schema is provided in Example 7.12.

```
...
  <xs:element name="echo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  </xs:element>
...
```

**Example 7.12**

Wrapping a message in one additional element named after the operation is prevalent and the default in any commonly used tool. Note that naming the global element after the operation is common practice and not required by the specification.

### Implicit and Explicit Headers

Transferring information as part of the `<Header>` portion of the SOAP message, to be added to the WSDL definition, is another important part of the binding information for SOAP. This section discusses how to bind the information with explicit, implicit, or no headers.

#### *Explicit Headers*

Header data is part of the messages referenced in the portType of the service, which is often called an explicit header. The header definition in the SOAP binding refers to a message part either included in the input message or the output message of an operation.

In Example 7.13, assume an Echo service takes a `string` as input and returns that `string` as the response. A `timestamp` must also be added into the header of the SOAP request message, indicating the time at which the request was sent.

```
<definitions targetNamespace="http://pack/" name="EchoService"
  xmlns:tns="http://pack/" xmlns:xs="http://www.w3.org/2001/
  XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xs:schema targetNamespace="http://pack/">
      <xs:element name="echo">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="arg0" type="xs:string" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="echoResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="return" type="xs:string" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="timestamp" type="xs:dateTime"/>
    </xs:schema>
  </types>
  <message name="echo">
    <part name="parameters" element="tns:echo"/>
    <part name="timestamp" element="tns:timestamp"/>
  </message>
  <message name="echoResponse">
    <part name="parameters" element="tns:echoResponse"/>
  </message>
  <portType name="Echo">
    <operation name="echo">
      <input message="tns:echo"/>
      <output message="tns:echoResponse"/>
    </operation>
  </portType>
  <binding name="EchoPortBinding" type="tns:Echo">
    <soap:binding transport="http://schemas.xmlsoap.org/ soap/http"
      style="document"/>
    <operation name="echo">
      <soap:operation soapAction=""/>
      <input>
        <soap:body parts="parameters" use="literal"/>
```

```
        <soap:header message="tns:echo" part="timestamp"
          use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
...
</definitions>
```

**Example 7.13**
The Echo service WSDL definition with an explicit header contains a timestamp.

Example 7.13 contains an extract of the respective WSDL definition for an Echo service that shows:

- an additional element in the schema, called `"timestamp"` of type `xs:dateTime`
- an additional part in the input message definition, which refers to the `timestamp` element
- an additional definition for the header in the SOAP binding, which indicates that the `timestamp` element should be carried in the SOAP header of the request message

The header binding shown in Example 7.14 refers to a part also included in the portType of the service, the input message, and the Java service interface generated by the JAX-WS wsimport tool. Note that the import statements are left out.

```
@WebService(name = "Echo", targetNamespace = "http://pack/")
@SOAPBinding(parameterStyle = ParameterStyle.BARE)
public interface Echo {
  @WebMethod
  @WebResult(name = "echoResponse", targetNamespace = "http://pack/",
    partName = "parameters")
  public EchoResponse echo(
    @WebParam(name = "echo", targetNamespace = "http://pack/",
      partName = "parameters")
    Echo_Type parameters,
      @WebParam(name = "timestamp", targetNamespace = "http://pack/",
      header = true, partName = "timestamp")
    XMLGregorianCalendar timestamp);
}
```

**Example 7.14**