Web Services with SAP Part 1 – Concepts

Applies to:

SAP Web Application Server (WAS) 6.40, 7.0 and CE (7.1).

Summary

The article presented here will take you through some basic concepts of Web Services, their architecture, and ideal WSDL structure. It also gives the description of Web Service Administration methods and background of Web Service Security.

Author: Alka Panday

Company: Larsen & Toubro Infotech Ltd.

Created on: 08 December 2008

Author Bio

Alka Panday works at Larsen & Toubro Infotech Ltd., as a SAP Netweaver Consultant. She started with Web Dynpro for Java, and now covers J2EE (EJBs), Web Services, Web Dynpro (ABAP), SOA and Web Services, with special interests in PI.

Table of Contents

Concept

A 'Web service' as defined by the W3C is "a software system designed to support interoperable machine-to-machine interaction over a network". Web Services, generally, are services / functions that execute on a remote system, when accessed / invoked over a network like Internet.

A Web Service, once created, is referred to using its WSDL (Web Service Definition Language). A Web Service WSDL is an XML like definition of the Web Service, containing the methods and their parameter descriptions, result definition and the defined exceptions that occur with Web Service execution.

Web Service Standards

Web Services is an open integration technology. Being open integration, it is based on the generally accepted standards of XML (Extensible Markup Language), WSDL and UDDI.

The Web Service operations on SAP WAS are driven by the following basic standards:

- WS Interoperability: WS-I BP 1.0
- XML: SOAP 1.1 w/ Attachments, WSDL 1.1, UDDI v2, Web Services Inspection Language (WSIL)
- Security: WS-Security (1.0), SSL / TLS
- Java AP: JAXP 1.2, JAXM 1.1, SAAJ 1.1, JAX-RPC 1.0

Web Service Architecture

The architecture of Web Services can be well derived from its definition.

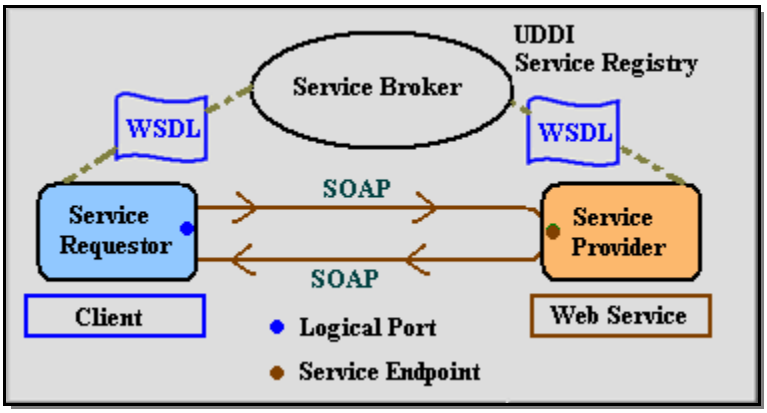

Fig 1: Web Service Architecture

The main entities in any kind of Web Service communication are the provider of the web service and its requestor / consumer.

Service Requestor is an entity which will request for the service and consume its execution result and, Service Provider is an entity that will execute the service / function upon receiving the invocation request. As shown in the above figure, this communication between the requestor and provider follows the SOAP standards, and the message body contains the XML data.

A Service Broker, like UDDI (Universal, Description, Discovery and Integration), can be used to store the Web Service addresses. It acts as a directory like Yellow Pages for Web Services, storing their name, WSDL addresses and details like web service interfaces. Service Requestors, in turn, can access this information to establish their Web Service clients and avail the functionality provided by the service.

Service Endpoint is a runtime representation of a service definition. It includes the runtime configuration settings of the service definition and is available on the provider system at a unique location called service endpoint URL. From the point of view of a Web service client, a service endpoint is the location at which that client can access the service definition with particular runtime settings. Logical Port, defined at the Service Requestor i.e. Client, contains a reference to the service endpoint it wants to access and has the runtime configuration settings, such as a user name and password, with which it can access the service endpoint.

Web Service Definition Language (WSDL)

Before going ahead with the procedures of creating and consuming Web Service with SAP WAS, it is of great importance to understand the information possessed by the different sections of the WSDL.

The various sections of an ideal WSDL are described below with an example of adding 2 integer numbers Web Service. For the whole WSDL refer to the Related Content section.

wsdl:types : Hierarchical tag with the details of all the message types giving out their parameter names, occurrences, and basic schema definition types.

```
- <wsdl:types>
 - <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
targetNamespace="http://sap.com/test/calc/">
      <xs:element name="addNum" type="tns:addNum" />
      <xs:element name="addNumResponse" type="tns:addNumResponse" />
    - <xs:complexType name="addNum">
     - <xs:sequence>
         <xs:element name="num1" type="xs:int" />
         <xs:element name="num2" type="xs:int" />
       </xs:sequence>
      </xs:complexType>
    - <xs:complexType name="addNumResponse">
     - <xs:sequence>
         <xs:element name="return" type="xs:int" />
       </xs:sequence>
      </xs:complexType>
  </xs:schema>
 </wsdl:types>
```

Thus, from the above snippet, there are two complex types, **addNum** and **addNumResponse**, elaborated with the names of the parameters as **num1** & **num2** for input and **return** for output, all of integer types.

wsdl:message : Specifies the messages that are exposed by the Web Service. It gives out the names of the message, part name and the subsequent element name that is defined in the types tag. Per operation that is included in the WSDL there can be one (asynchronous), two (synchronous) or three (synchronous with fault) messages.

```
- <wsdl:message name="addNumIn">
      <wsdl:part name="parameters" element="tns:addNum" />
  </wsdl:message>
- <wsdl:message name="addNumOut">
      <wsdl:part name="addNumResponse" element="tns:addNumResponse" />
  </wsdl:message>
```

With Add Numbers Web Service, there is one synchronous operation, and hence two messages for request and response respectively.

wsdl:portType : Name of the Web service logical port, and all the operation names and their input, output and fault message names and types.

```
- <wsdl:portType name="CalculatorInt">
   - <wsdl:operation name="addNum" parameterOrder="parameters">
       <wsdl:input message="tns:addNumIn" />
       <wsdl:output message="tns:addNumOut" />
     </wsdl:operation>
  </wsdl:portType>
```

As we can see, the port type name is **CalculatorInt**, single operation **addNum** with input and output messages is defined.

wsdl:bindings : Name of the binding, define all the operations in the Web Service and their input, output and fault parameter types.

```
- <wsdl:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
name="CalculatorIntBinding" type="tns:CalculatorInt">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
 - <wsdl:operation name="addNum">
     <soap:operation soapAction="" />
   - <wsdl:input>
        <soap:body parts="parameters" use="literal" />
     </wsdl:input>
  -  <wsdl:output>
        <soap:body use="literal" />
     </wsdl:output>
   </wsdl:operation>
  </wsdl:binding>
```

As shown above, the details regarding the parameters of the operation **addNum** are specified.

wsdl:service : Gives the name of the Web service and its callable URL for the clients

```
- <wsdl:service name="CalcService">
   - <wsdl:port name="CalcBeanPort" binding="tns:CalculatorIntBinding">
        <soap:address xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
          location="http://hostserver:50000/CalcService/CalcBean" />
     </wsdl:port>
  </wsdl:service>
```

The name of the Web service is **CalcService**, and the URL of the Web service is specified in the soap:address tag's location attribute.

Web Services Security

Talking about security for any Web Service there are two points of discussion, Authentication and Transport Guarantee. Web Service authentication deals with the validation of the user account with the Web Service Endpoint / Provider, while transport guarantee also termed ad Connection Security ensures security of the message delivery till the endpoint.

Web Service security is also defined by the security mechanism provided at Transport level and Message level. At transport level, security can be ensured by means of mechanisms used on the Internet. HTTPS sets up an encrypted connection between the client and the server where every single message is sent via an encrypted channel. This has disadvantages when there are many messages or when Web Service is not confidential and as always when SOAP interaction is not point – to – point. At message level, an encryption and signature concept with fine granularity is possible. Here, not the transport canal but the message itself is protected.

The security configurations that are possible with Web Services can be generally described as:

- Simple SOAP: The transport binding is provided over HTTP and the communication is stateless. No user security is supported. This can be used when the Web Service is not confidential and the Web Service communication is over Internet.
- Basic Auth SOAP: The transport binding and communication her too are over HTTP and stateless respectively. The authentication is Basic HTTP, where a valid User Id and Password are needed to access the Web Service. Using this method, the Web Service operations can also be protected.
- Secure SOAP: The communication with this security configuration is also stateless by default. The transport binding is SOAP over HTTPS. The authentication for Web Service is carried out using Digital Certificates. Authorization for Web Service methods can be configured. Additional configurations to ensure transport guarantee (encryption for messages) are provided. For SAP WAS, the X.509 client certificates are used and the message encryption is handled using Secure Socket Layer (SSL) protocol.

SAP NetWeaver Administrator – SOA Management

SAP's NetWeaver Administration application (NWA) is a one point access to all the administrative operations that are needed to be conducted by the Web AS Administrator for smooth functioning. The application is only available with SAP Web AS 7.0 and higher. The SOA Management for the NetWeaver Administrator is subdivided into 4 tasks viz., Technical Configuration, Business Administration, Logs and Traces and Monitoring. The mostly used administration options available under these tasks are briefly described here:

Destination Template Management

This service allows the administrator to configure at one go one or more Web service clients running on the same consumer system to consume Web services provided by another system.

Mass Configuration and Profile Management

Mass Configuration is used to group a set of Web Services with identical runtime system in a configuration scenario and apply settings to one or more of these Web Services by assigning them with configuration profiles. These configuration profiles can be designed using Profile Management.

Web Service Administration

Administrator can configure individual Web Services and Web Service clients by applying runtime settings to them using this option. The administrator can also view the information of all individual Web Services and clients, as well as review the logs and traces for the calls to and from Web Service clients.

Service Registry Configuration and Publication Restrictions

The Service Registry configuration option allows the administrator to configure the UDDI and Service Registry. Publication Restrictions is used to publish the Web Services to the Service registry.

Web Services Logging and Tracing

This option is used to view and analyze the logs and traces of calls made to Web services and from Web service clients deployed on the system. Administrator can set up filters for easy analysis.

SOAMANGER

The SOAMANAGER transaction available with ECC 6.0 and Web AS 7.0 and above is used for Administration of the Web Services of the ABAP stack. Previous to the mentioned versions, transactions WSCONFIG and WSADMIN were used in conjunction for Web Service administrative tasks.

The initialization of SOAMANAGER transaction results into start of the Web Browser with SOA management task. The functionalities available with SOAMANAGER are equivalent in all respects with the SOA Management Tab of the NWA.

Related Content

Web Service Standards and Open Integration: An Overview

Utilizing Web Services with Java and SAP Web Application Server 6.40

First Look at WSDL 2.0

Calculator Web Service

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://lti.com/test/calc/" targetNamespace="http://lti.com/test/calc/">
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
targetNamespace="http://lti.com/test/calc/">
      <xs:element name="addNum" type="tns:addNum"/>
      <xs:element name="addNumResponse" type="tns:addNumResponse"/>
      <xs:complexType name="addNum">
        <xs:sequence>
          <xs:element name="num1" type="xs:int"/>
          <xs:element name="num2" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="addNumResponse">
        <xs:sequence>
          <xs:element name="return" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="addNumIn">
    <wsdl:part name="parameters" element="tns:addNum"/>
  </wsdl:message>
  <wsdl:message name="addNumOut">
    <wsdl:part name="addNumResponse" element="tns:addNumResponse"/>
  </wsdl:message>
  <wsdl:portType name="CalculatorInt">
    <wsdl:operation name="addNum" parameterOrder="parameters">
      <wsdl:input message="tns:addNumIn"/>
      <wsdl:output message="tns:addNumOut"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
name="CalculatorIntBinding" type="tns:CalculatorInt">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="addNum">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body parts="parameters" use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CalcService">
    <wsdl:port name="CalcBeanPort" binding="tns:CalculatorIntBinding">
      <soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
location="http://sappi71:50000/CalcService/CalcBean"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.