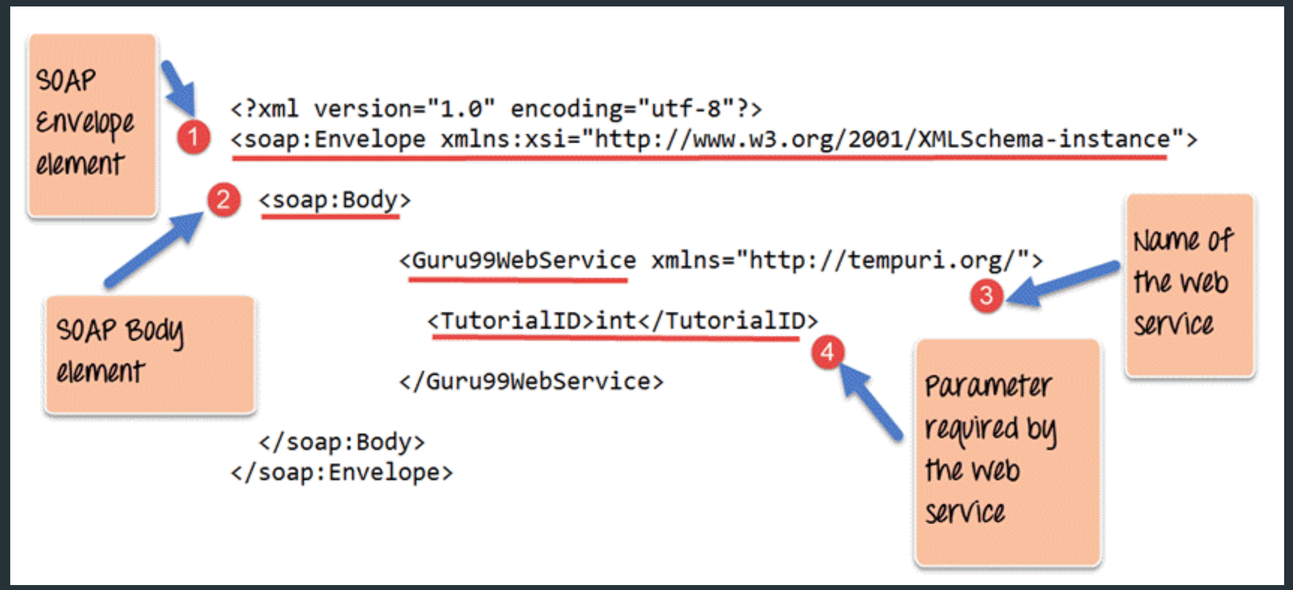
Lecture #2
SOAP

Spring 2024

Introduction to SOAP Web Services

- Stands for Simple Object Access Protocol
- A messaging protocol used to exchange structured data over the internet
- Use XML as their message format

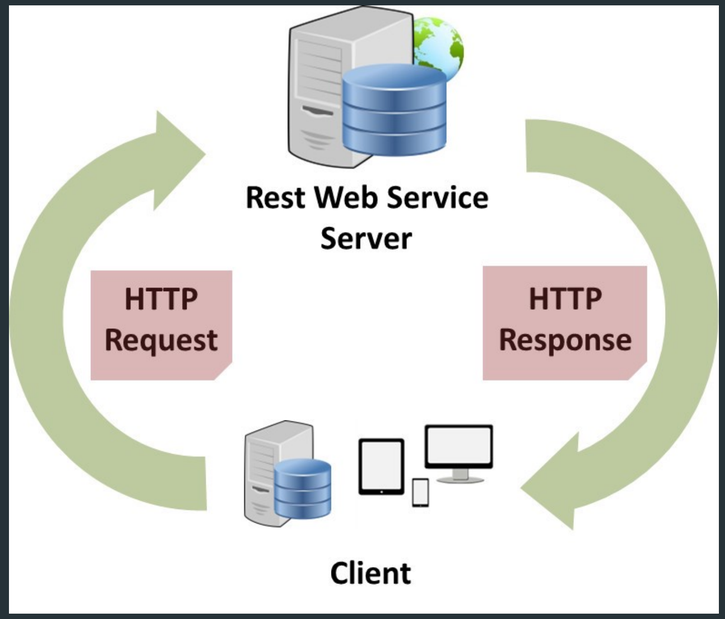


Brief History of SOAP Web Services

- SOAP was first introduced by Microsoft in 1998 as a protocol for exchanging structured data over the internet
- SOAP 1.1 was published as a W3C recommendation in 2000
- SOAP 1.2 was published as a W3C recommendation in 2003

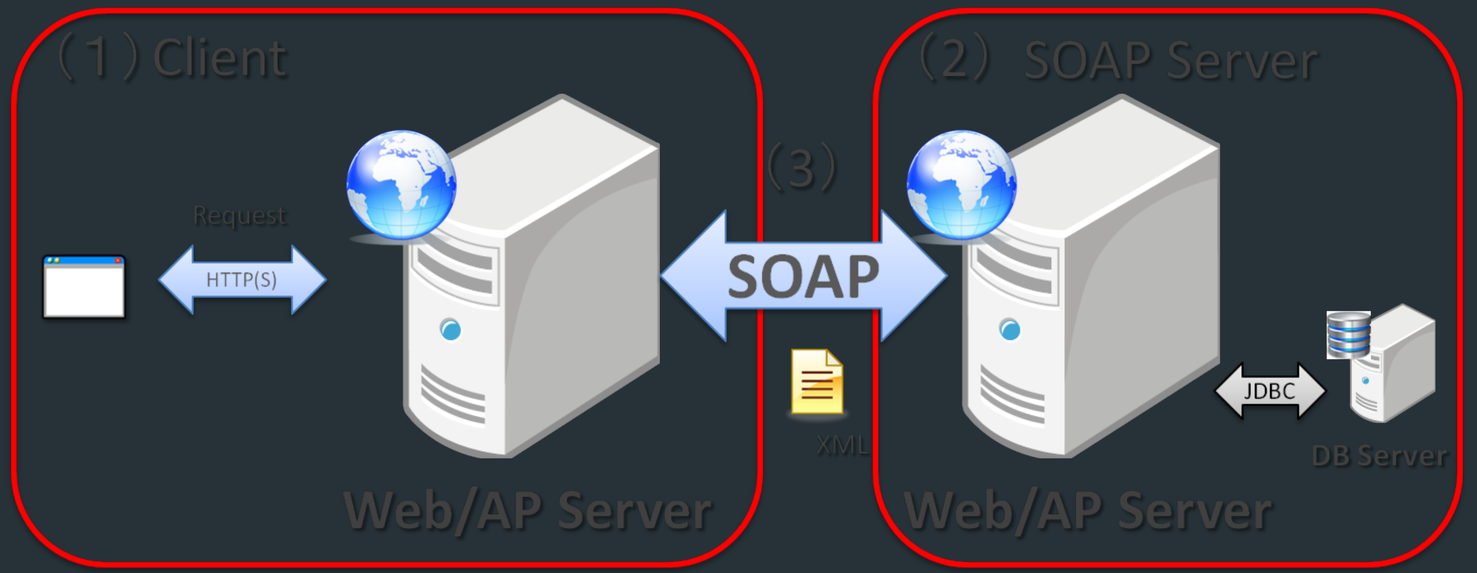
Evolution of SOAP Web Services

- With the introduction of RESTful web services in mid-2000s, SOAP faced competition
- SOAP 1.2 introduced more flexibility and support for wider range of transport protocols
- SOAP 1.2 became widely adopted by enterprises for integration of their systems



Current State of SOAP Web Services

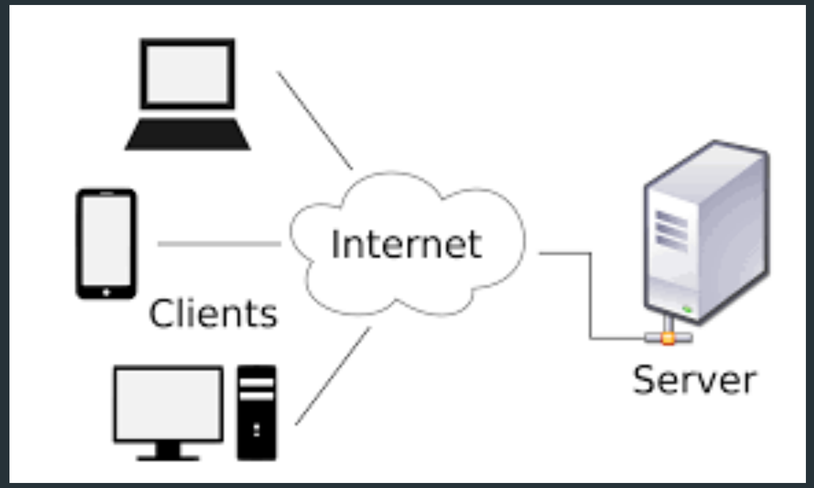
- Continue to be used for enterprise-level integrations and large-scale applications
- SOAP 1.2 still remains the current version of the SOAP protocol
- Many programming languages and platforms provide built-in support for SOAP web services



Understanding the Web Services Architecture

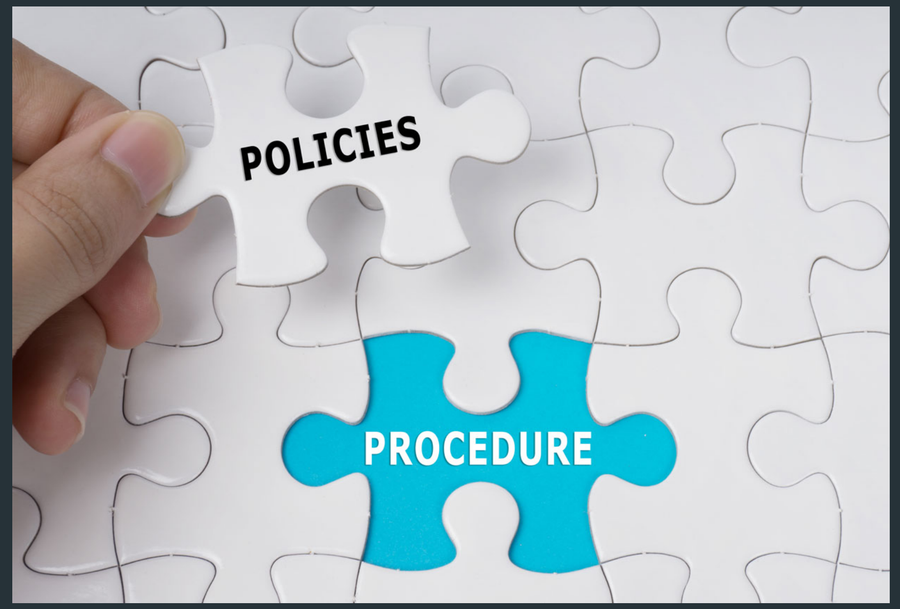
Client-Server Model

- Web services architecture is based on the client-server model for distributed computing
- The client is the software component that consumes the services provided by the server
- The server is the software component that provides the services to the client



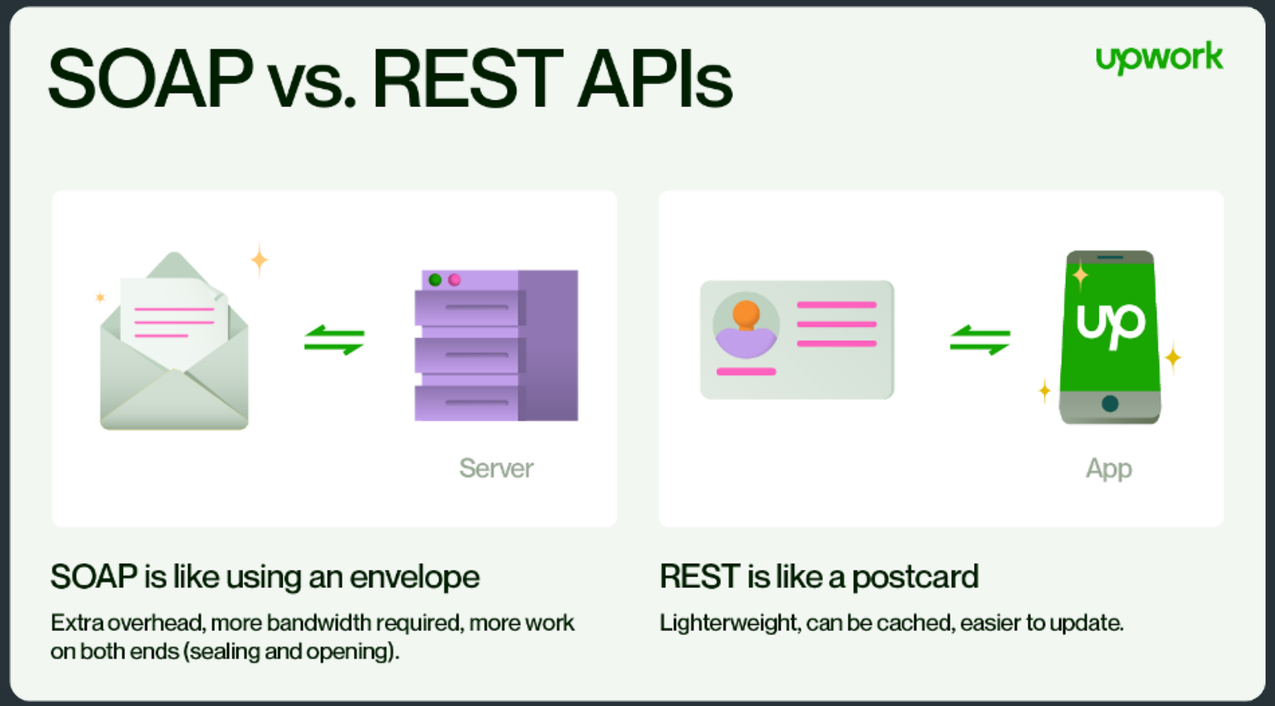
Standardized Protocols

- Web services architecture uses standardized protocols to ensure interoperability between different software components
- HTTP is the standard protocol for web communication
- XML and SOAP are commonly used for data exchange in web services



SOAP vs RESTful

- Web services can be classified as either SOAP or RESTful
- SOAP web services use the SOAP protocol for message transmission
- RESTful web services use the REST architecture and typically use the HTTP protocol



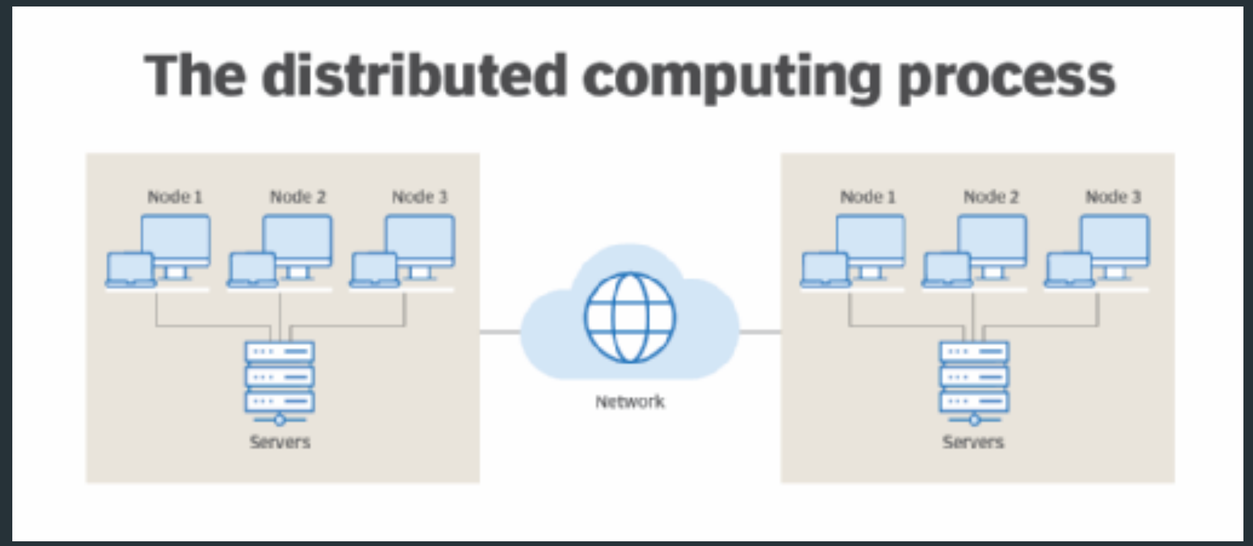
Key Characteristics of SOAP Web Services

XML-Based Message Format

- SOAP web services use an XML-based message format
- This makes it easier to exchange structured data
- Ensures interoperability between different systems

Designed for Distributed Computing

- SOAP web services are designed for distributed computing
- Facilitate communication between different software components
- Can be located on different systems or platforms



Built-in Error Handling and Security

- SOAP web services have built-in error handling capabilities
- This makes it easier to handle errors that may occur during message transmission or processing
- SOAP web services also support security and transaction handling

Advantages of SOAP Web Services

- Robust error handling
- Wide industry support
- Built-in security features

Disadvantages of SOAP Web Services

- Complex message structure
- Slower performance compared to RESTful web services
- Large message size

Comparison between SOAP and RESTful Web Services

- SOAP:
  - Complex message structure
  - Uses XML for data exchange
  - Robust error handling
  - Built-in security features
- RESTful:
  - Simple message structure
  - Uses JSON for data exchange
  - Fast performance
  - No built-in error handling or security features

Comparison between SOAP and RESTful Web Services

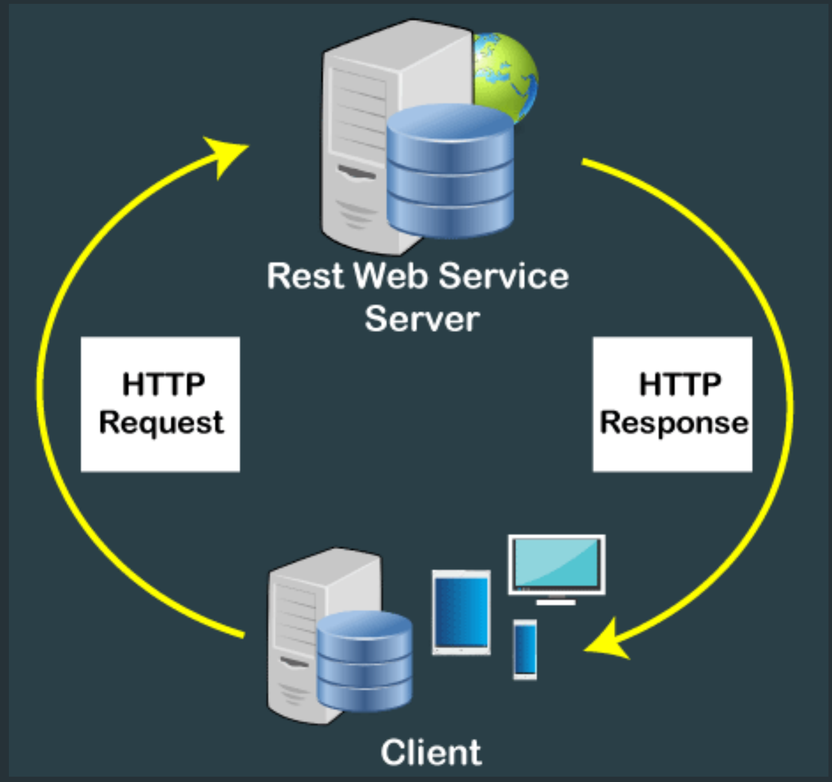
- SOAP:
  - Complex message structure
  - Uses XML for data exchange
  - Robust error handling
  - Built-in security features
- RESTful:
  - Simple message structure
  - Uses JSON for data exchange
  - Fast performance
  - No built-in error handling or security features

Comparison between SOAP and RESTful Web Services

- SOAP:
  - Complex message structure
  - Uses XML for data exchange
  - Robust error handling
  - Built-in security features
- RESTful:
  - Simple message structure
  - Uses JSON for data exchange
  - Fast performance
  - No built-in error handling or security features

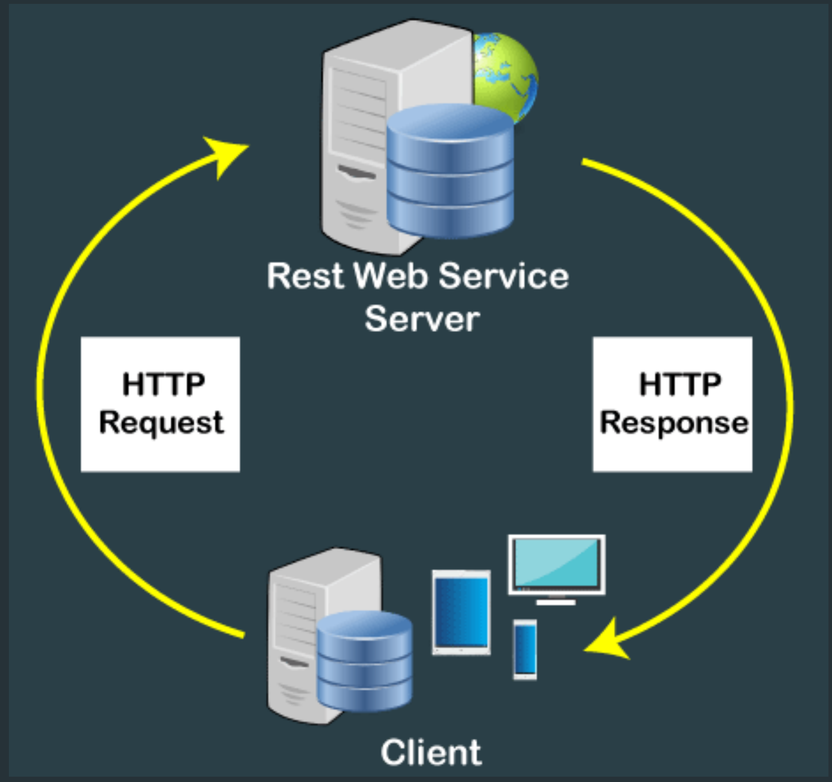
How SOAP Works

1. Client sends a SOAP request to the server.
2. Server receives the SOAP request and processes it.
3. Server sends a SOAP response back to the client.
4. Client receives the SOAP response and processes it.



How SOAP Works

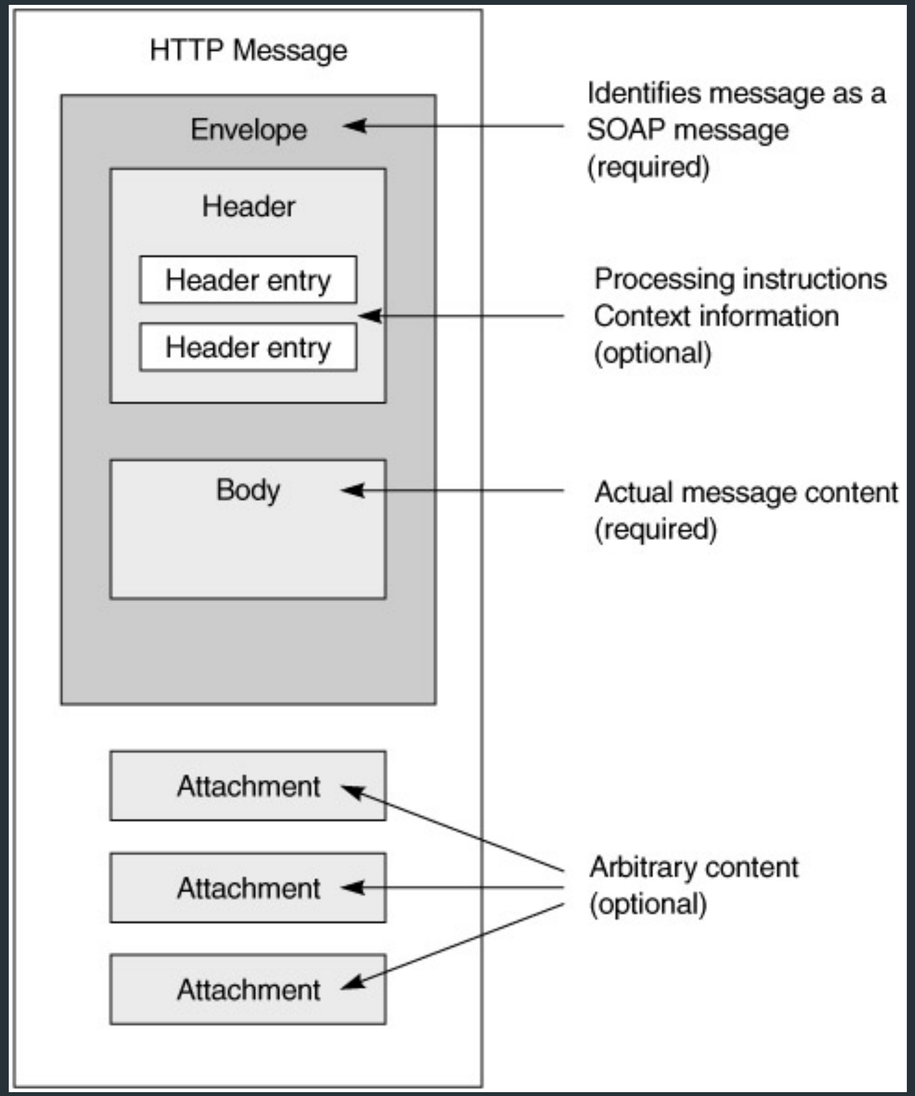
1. Client sends a SOAP request to the server.
2. Server receives the SOAP request and processes it.
3. Server sends a SOAP response back to the client.
4. Client receives the SOAP response and processes it.



Messages and Envelope Structure

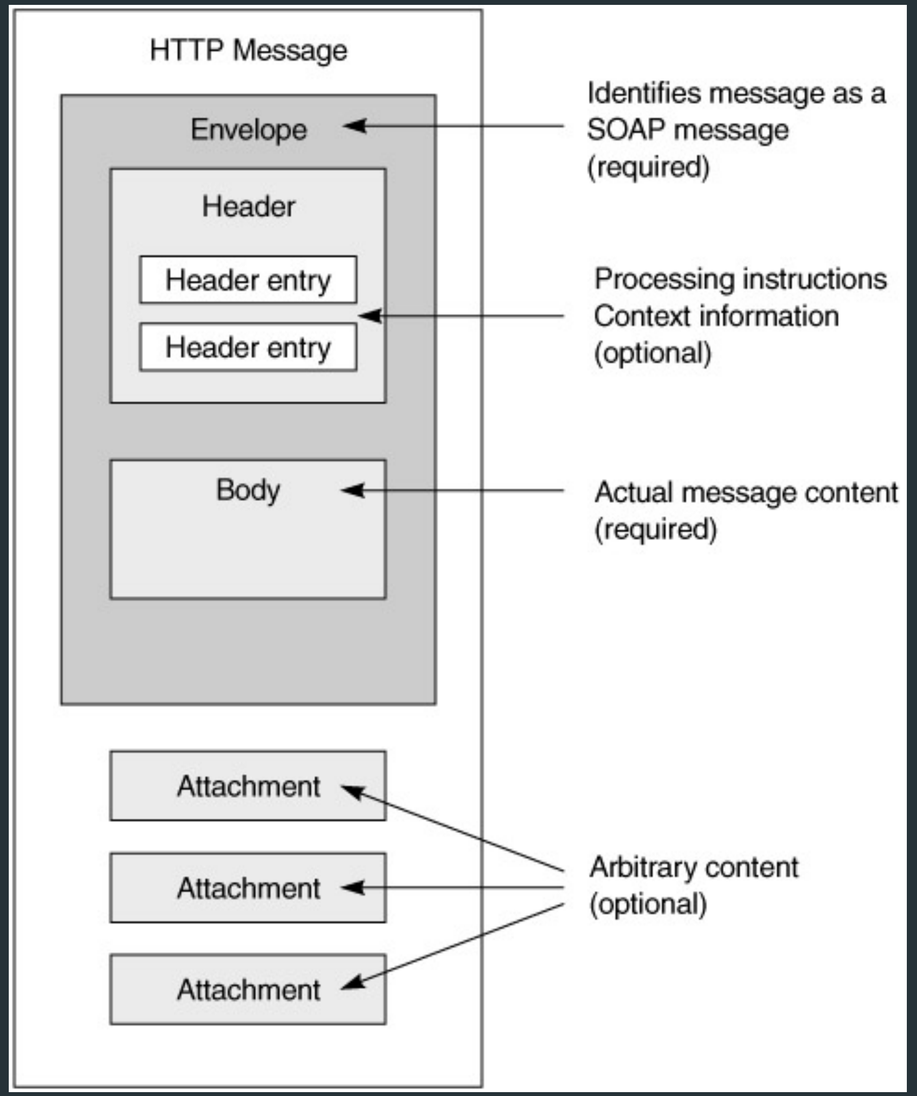
Messages and Envelope Structure

- SOAP messages have a defined structure that consists of a mandatory envelope element and optional header and body elements.
- The envelope element is the root element of the SOAP message and contains all other elements.
- The header element is optional and can contain additional information about the message, such as security credentials or routing information.
- The body element is mandatory and contains the actual message data.



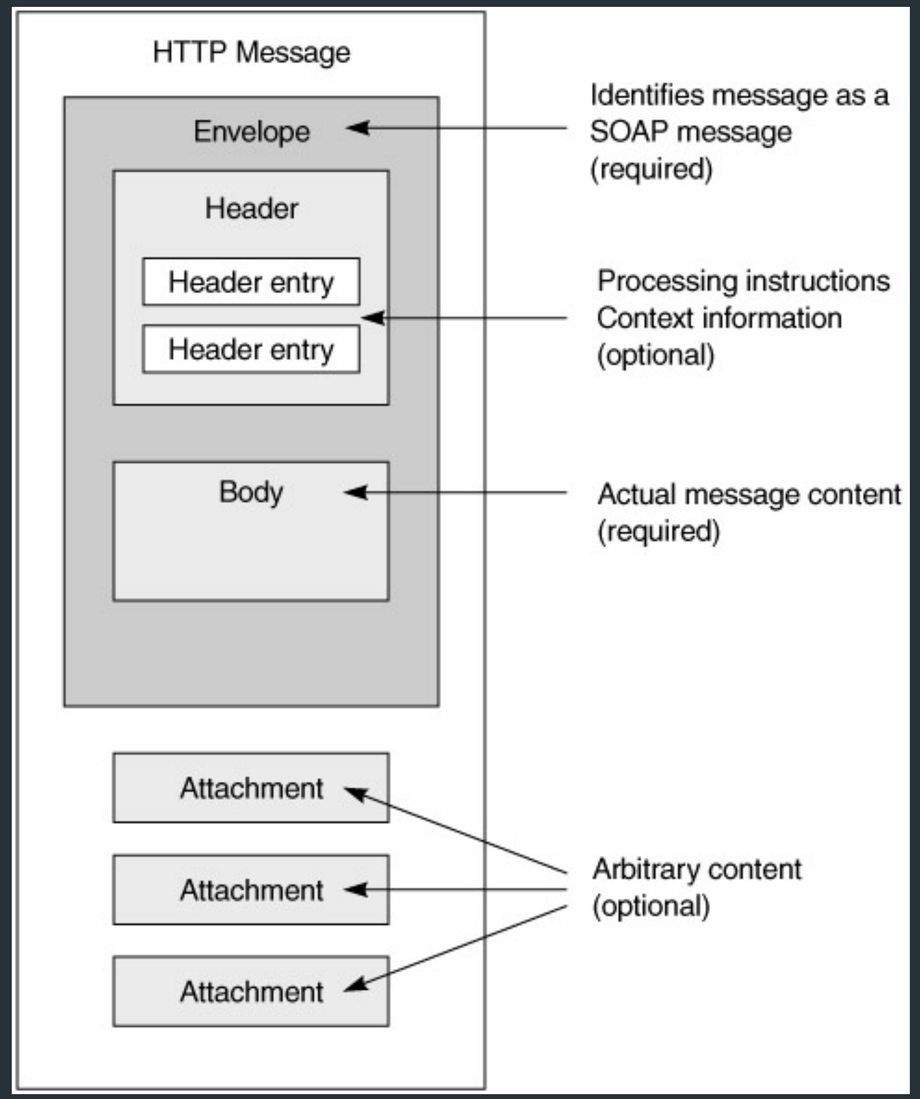
Understanding Headers

- SOAP headers are optional and can contain additional information about the message.
- Headers can be used to provide additional information, such as authentication credentials or routing information.
- Headers are defined using the soap:Header element.



Understanding the Body

- SOAP body is mandatory and contains the actual message data being exchanged between the client and server.
- The content of the SOAP body can be any type of data, but it is typically in XML format.
- The SOAP body is defined using the soap:Body element.



Understanding Faults

- SOAP faults are used to indicate errors that occur during the processing of a SOAP message.
- SOAP faults are defined using the soap:Fault element.
- SOAP faults can contain a fault code, a fault string, and a fault detail.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns0="http://bank.com/wsdl/BigBank"
env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <env:Body>
      <env:Fault>
       <faultcode>env:Server</faultcode>
       <faultstring>Internal Server Error
                (unexpected encoding style:
   expected=http://schemas.xmlsoap.org/soap/encoding/, actual=)
       </faultstring>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

Handling SOAP Faults

- Clients can handle SOAP faults using try-catch blocks.
- When a SOAP fault occurs, the client can extract the fault code, fault string, and fault detail from the SOAP message.
- Clients can use the information in the SOAP fault message to take appropriate action, such as retrying the operation or notifying the user.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns0="http://bank.com/wsdl/BigBank"
env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <env:Body>
       <env:Fault>
        <faultcode>env:Server</faultcode>
        <faultstring>Internal Server Error
                  (unexpected encoding style:
   expected=http://schemas.xmlsoap.org/soap/encoding/, actual=)
        </faultstring>
       </env:Fault>
   </env:Body>
</env:Envelope>
```

Best Practices for SOAP Faults

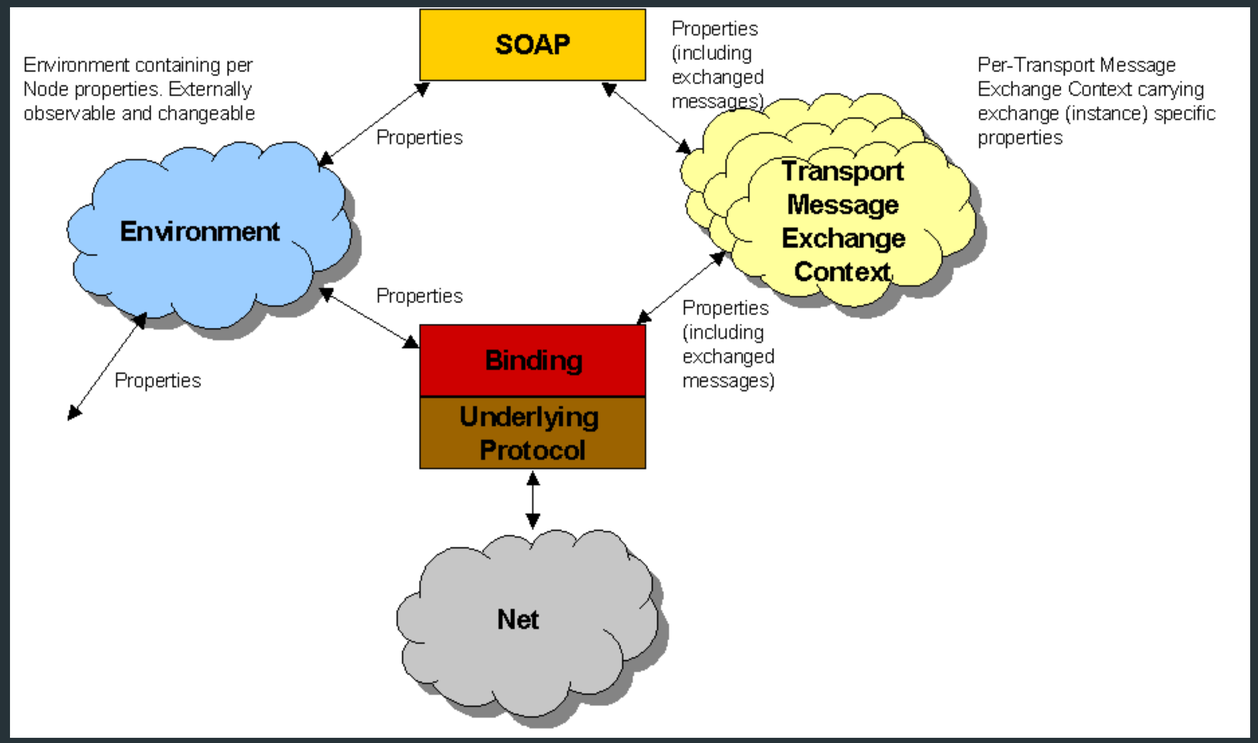
- Use meaningful fault codes and fault strings.
- Provide detailed fault details to help diagnose and fix errors.
- Use appropriate HTTP status codes to indicate success or failure.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns0="http://bank.com/wsdl/BigBank"
env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <env:Body>
      <env:Fault>
       <faultcode>env:Server</faultcode>
       <faultstring>Internal Server Error
                (unexpected encoding style:
   expected=http://schemas.xmlsoap.org/soap/encoding/, actual=)
       </faultstring>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

SOAP Binding and Transport Protocol

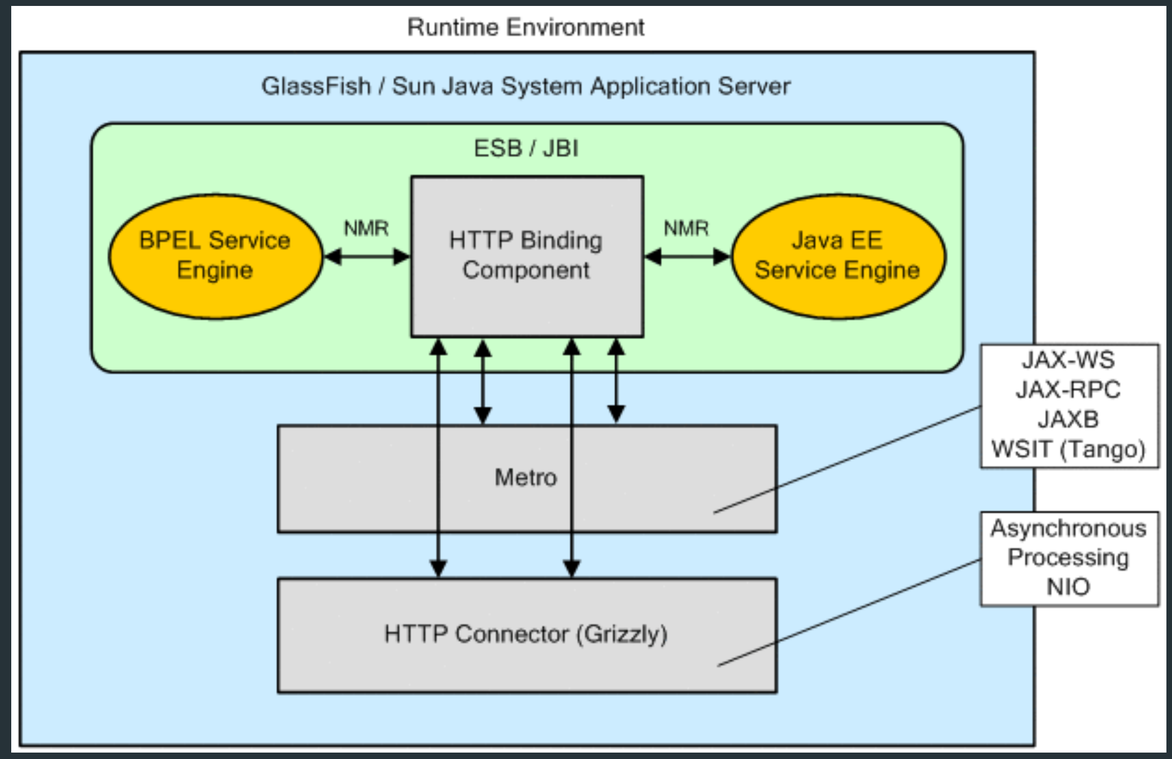
Binding and Transport Protocol

- SOAP binding specifies how SOAP messages are mapped onto a transport protocol.
- SOAP binding can be either HTTP or SMTP.
- Transport protocol defines how messages are transmitted over the network.



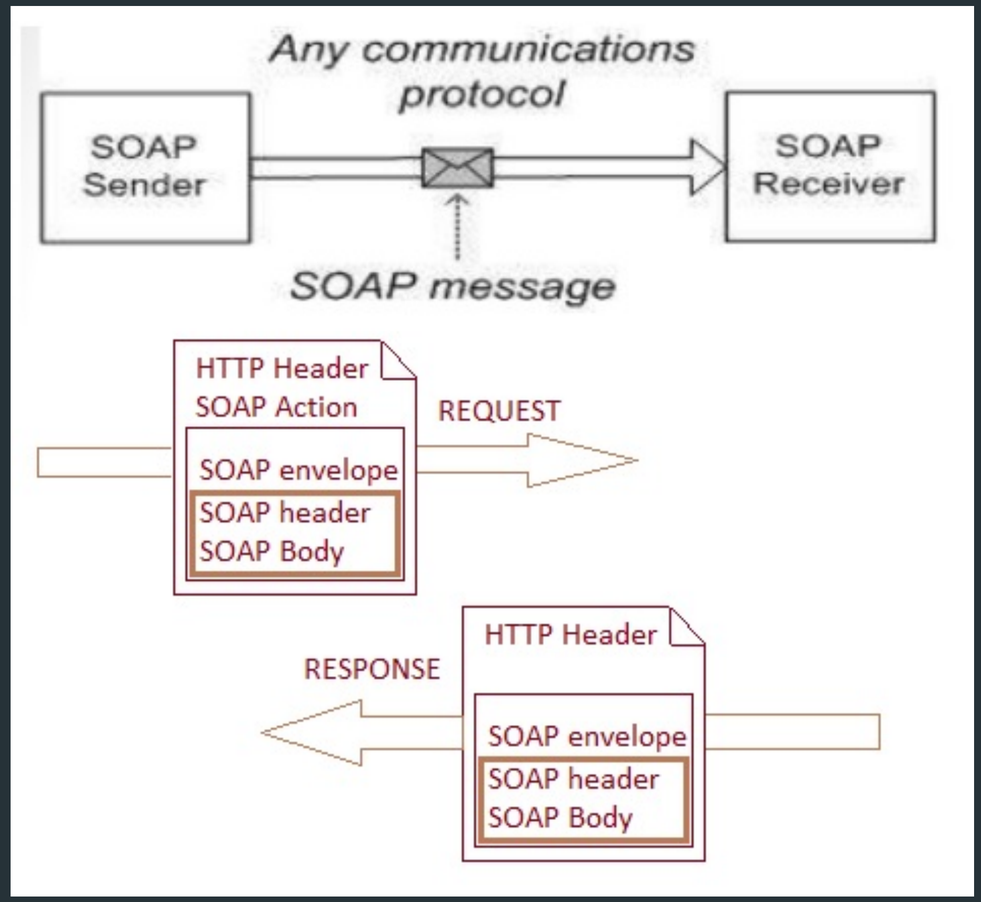
SOAP with HTTP and HTTPS

- SOAP with HTTP is the most common SOAP binding.
- HTTP provides a lightweight and flexible protocol for transmitting SOAP messages.
- HTTPS provides a secure version of HTTP and is commonly used for SOAP-based web services that require encryption and authentication.



SOAP with SMTP and TCP

- SOAP with SMTP is less common than SOAP with HTTP.
- SMTP is used for SOAP-based email services.
- SOAP with TCP is less common than SOAP with HTTP and is used for low-level network communication.



Creating a SOAP Service

Creating a Simple SOAP Web Service in Java

- Java provides several libraries and tools for creating SOAP-based web services.
- The Java API for XML Web Services (JAX-WS) is a standard Java API for building SOAP-based web services.
- Creating a simple SOAP web service in Java involves defining the service interface, implementing the service, and deploying the service to a web server.

```
@WebService
public class HelloWorld {

    @WebMethod
    public String sayHello(String name) {
        return "Hello, " + name + "!";
    }

}
```

Defining the Service Interface

- The service interface defines the methods that the service will expose.
- The @WebService and @WebMethod annotations are used to define the service interface.
- The service interface can be defined using either Java SE or Java EE.

```
@WebService
public interface HelloWorld {

    @WebMethod
    public String sayHello(String name);

}
```

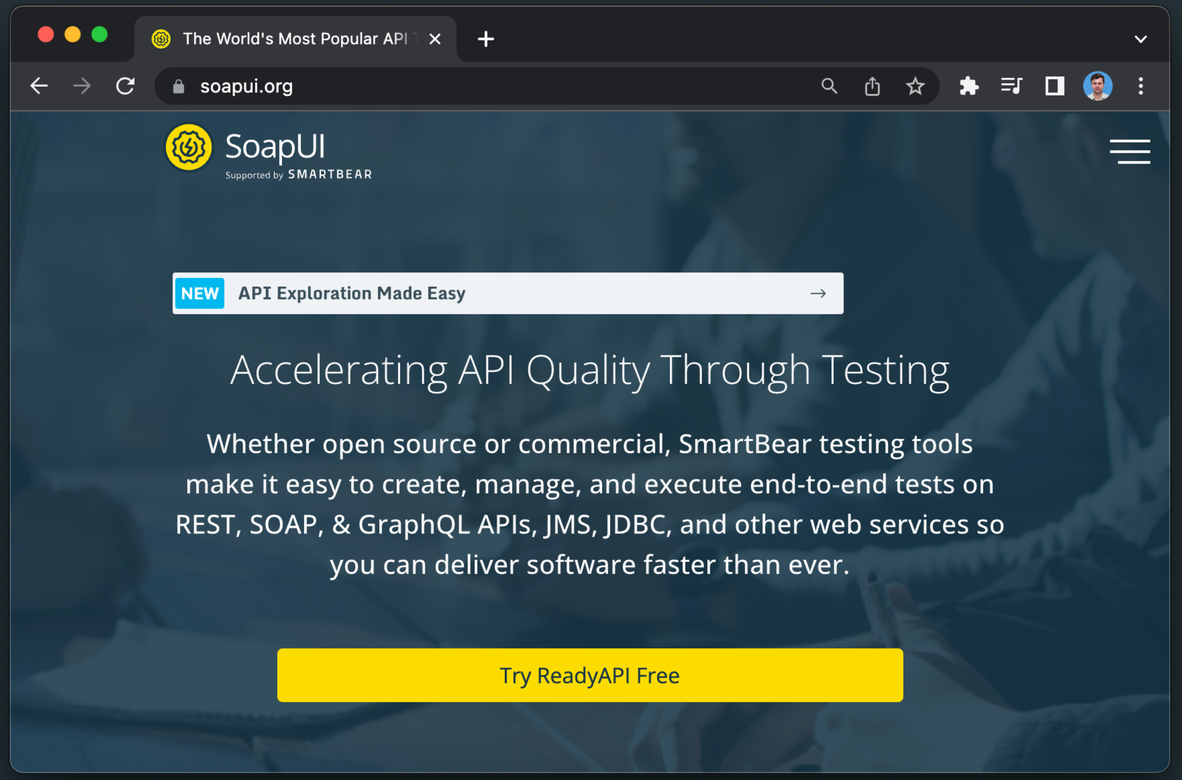
Implementing the Service

- The service implementation contains the actual implementation of the methods defined in the service interface.
- The @WebService and @WebMethod annotations are used to define the service implementation.
- The service implementation can be defined as a standalone Java class or as a Java EE session bean.

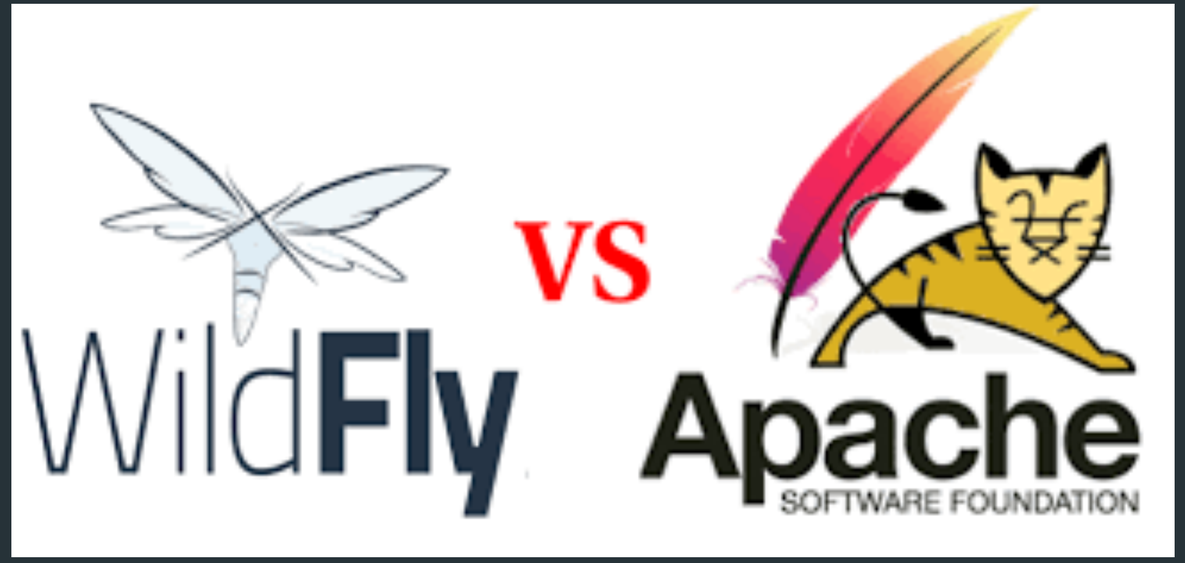
```
@WebService
public class HelloWorldImpl implements HelloWorld {

    @Override
    public String sayHello(String name) {
        return "Hello, " + name + "!";
    }

}
```

```java
@WebService
public interface HelloWorld {
    @WebMethod
    String sayHello(String name);
}

@WebService(endpointInterface = "com.example.HelloWorld")
public class HelloWorldImpl implements HelloWorld {
    @Override
    public String sayHello(String name) {
        return "Hello, " + name + "!";
    }
}

public class Main {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/hello", new HelloWorldImpl());
    }
}
```

Testing the SOAP Web Service



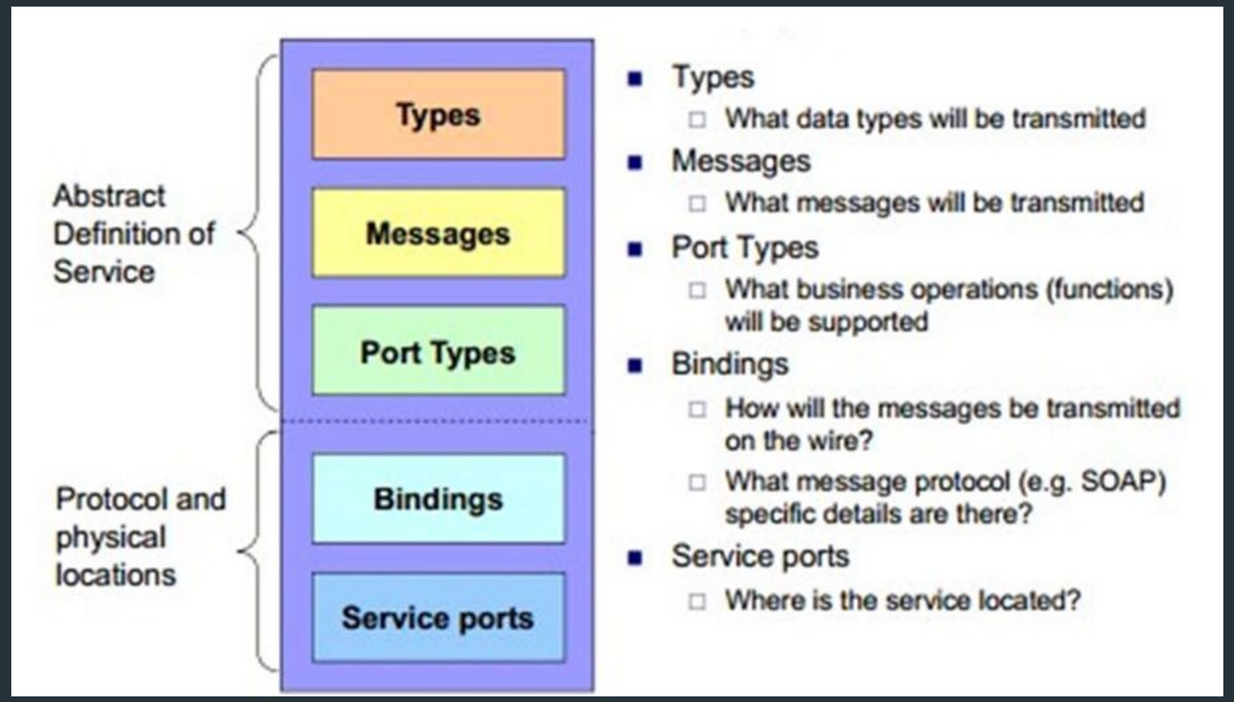
Deploying the Service



Using WSDL to Describe SOAP Web Services

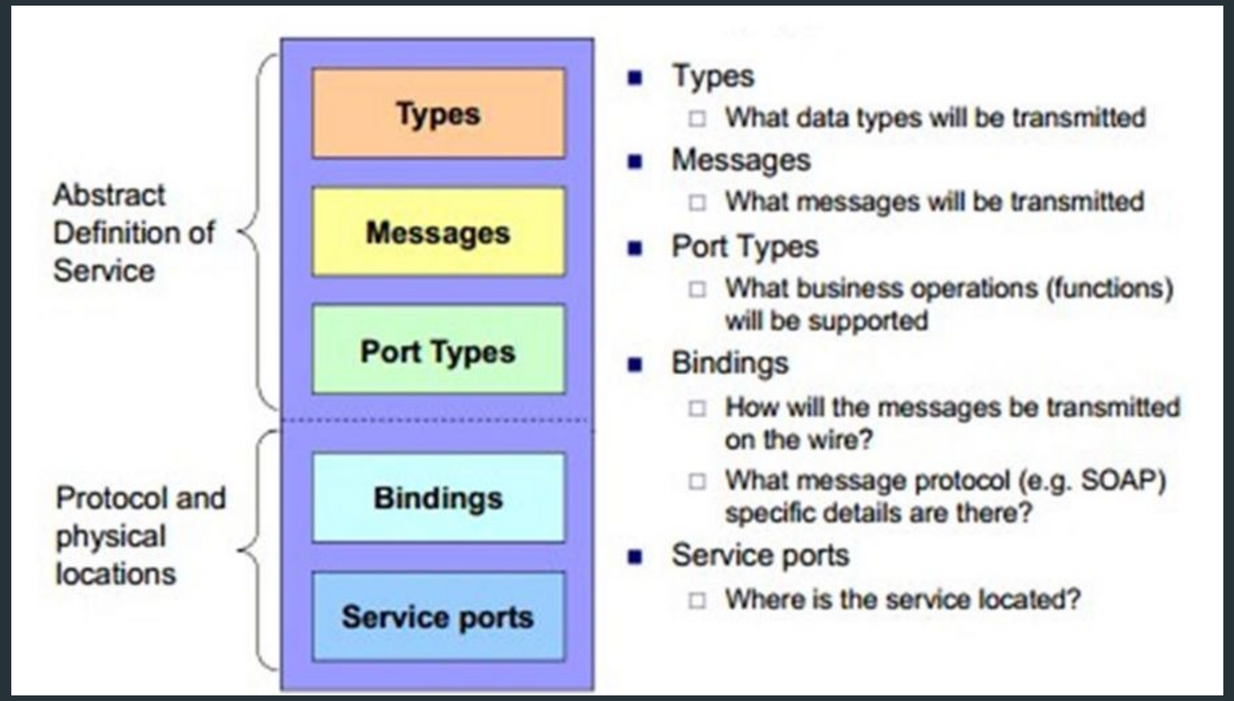
What is WSDL?

- WSDL describes the operations, input and output messages, and the binding details of a web service.
- WSDL can be used by web service clients to discover and interact with the web service.
- WSDL is typically generated automatically by the web service development tools.



Anatomy of a WSDL Document

- A WSDL document consists of several elements that describe the various aspects of a web service.
- The types element defines the data types used by the web service.
- The message element defines the structure of the input and output messages for each operation.
- The portType element defines the operations that the web service provides.
- The binding element specifies the protocol and message format used for each operation.
- The service element describes the location and protocol binding for the web service.



Generating WSDL for a SOAP Web Service

- To generate a WSDL file using wsgen, you need to specify the Java class that contains the web service endpoint and the location where the WSDL file should be saved.
- Once the WSDL file is generated, it can be used by web service clients to discover and interact with the web service.
- WSDL can also be used to generate client code for calling the web service.

```
wsgen -cp . com.example.HelloWorldImpl -wsdl -keep
```

```
<definitions name="TemperatureConversionService"
             targetNamespace="http://example.com/temperature-conversion"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://example.com/temperature-conversion"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <types>
    <xsd:schema targetNamespace="http://example.com/temperature-conversion">
      <xsd:element name="Fahrenheit" type="xsd:double"/>
      <xsd:element name="Celsius" type="xsd:double"/>
    </xsd:schema>
  </types>

  <message name="FahrenheitToCelsiusRequest">
    <part name="Fahrenheit" type="xsd:double"/>
  </message>

  <message name="FahrenheitToCelsiusResponse">
    <part name="Celsius" type="xsd:double"/>
  </message>

  <portType name="TemperatureConversion">
    <operation name="FahrenheitToCelsius">
      <input message="tns:FahrenheitToCelsiusRequest"/>
```

```
<message name="FahrenheitToCelsiusResponse">
  <part name="Celsius" type="xsd:double"/>
</message>

<portType name="TemperatureConversion">
  <operation name="FahrenheitToCelsius">
    <input message="tns:FahrenheitToCelsiusRequest"/>
    <output message="tns:FahrenheitToCelsiusResponse"/>
  </operation>
</portType>

<binding name="TemperatureConversionSoapBinding" type="tns:TemperatureConversion">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="FahrenheitToCelsius">
    <soap:operation soapAction="http://example.com/temperature-conversion/FahrenheitToCelsius"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>

<service name="TemperatureConversionService">
  <port name="TemperatureConversionSoap" binding="tns:TemperatureConversionSoapBinding">
```

```
  <portType name="TemperatureConversion">
    <operation name="FahrenheitToCelsius">
      <input message="tns:FahrenheitToCelsiusRequest"/>
      <output message="tns:FahrenheitToCelsiusResponse"/>
    </operation>
  </portType>

  <binding name="TemperatureConversionSoapBinding" type="tns:TemperatureConversion">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="FahrenheitToCelsius">
      <soap:operation soapAction="http://example.com/temperature-conversion/FahrenheitToCelsius"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="TemperatureConversionService">
    <port name="TemperatureConversionSoap" binding="tns:TemperatureConversionSoapBinding">
      <soap:address location="http://example.com/temperature-conversion"/>
    </port>
  </service>

</definitions>
```

Generating Client-Side Proxy Stubs with WSDL

Generating Client-Side Proxy Stubs with WSDL

- A client-side proxy stub is a code generated from the WSDL that can be used to access the web service from the client-side application.
- Generating client-side proxy stubs with WSDL can be done using tools such as Apache Axis, CXF, or the wsimport tool.

Using Apache CXF to Generate Client-Side Proxy Stubs with WSDL

- Apache CXF is an open-source web services framework that can be used to develop and deploy web services and clients.
- The wsdl2java tool can be used to generate client-side proxy stubs from a WSDL using Apache CXF.
- The generated code includes Java classes and interfaces that can be used to access the web service.


```
wsdl2java -d /path/to/output/dir http://example.com/calculator?wsdl
```

Using wsimport to Generate Client-Side Proxy Stubs with WSDL

- wsimport is a command-line tool that is included with the Java SDK.
- wsimport can be used to generate client-side proxy stubs from a WSDL.
- The generated code includes Java classes and interfaces that can be used to access the web service.


```
wsimport -d /path/to/output/dir http://example.com/calculator?wsdl
```

Working with SOAP Clients in .NET

Generating Client-Side Proxy Stubs with WSDL

- .NET provides several ways to create a SOAP client.
- One approach is to use the "Add Service Reference" option in Visual Studio, which generates client proxy code based on the service's WSDL (Web Services Description Language) file.
- Another approach is to manually create a client proxy by adding a Service Reference to the service's WSDL file.

```
// Generate a client proxy using the "Add Service Reference"
// option in Visual Studio
var client = new MyServiceClient();

// Manually create a client proxy using a WSDL file
var wsdlUrl = "http://example.com/myservice?wsdl";
var binding = new BasicHttpBinding();
var endpoint = new EndpointAddress(wsdlUrl);
var client = new MyServiceClient(binding, endpoint);
```

Consuming a SOAP Web Service in .NET

- Once a SOAP client has been created, developers can consume a SOAP web service using the client proxy.
- To call a web service method, developers simply invoke the corresponding method on the client proxy object.
- The client proxy object handles all communication with the web service and returns the method's response to the caller.

```
try
{
    // Call a method on the SOAP service using the client proxy
    var result = client.MyMethod(param1, param2);

    // Handle the method's response
    Console.WriteLine("Result: " + result);
}
catch (FaultException ex)
{
    // Handle the SOAP fault
    Console.WriteLine("SOAP Fault: " + ex.Message);
}
```

WS-Security

Consuming a SOAP Web Service in .NET

- WS-Security is a widely used specification for securing SOAP messages
- It defines a set of security tokens and mechanisms for protecting the confidentiality and integrity of SOAP messages
- WS-Security can be used to secure SOAP messages in various scenarios, including authentication, encryption, and digital signatures

WS-Security Architecture

- WS-Security uses a layered architecture for securing SOAP messages
- At the bottom layer, security tokens are used to authenticate users and provide proof of identity
- At the middle layer, message encryption and signature are used to ensure message confidentiality and integrity
- At the top layer, security policies are defined to specify which security measures should be applied to the SOAP message

Example Code for WS-Security

```
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.ServiceModel.Security;

// create a new binding with WS-Security
var binding = new BasicHttpBinding();
binding.Security.Mode = BasicHttpSecurityMode.TransportWithMessageCredential;
binding.Security.Message.ClientCredentialType = BasicHttpMessageCredentialType.UserName;

// create a new client with the binding
var client = new CalculatorServiceClient(binding, new EndpointAddress("http://example.com/calculator"));

// add WS-Security headers to the SOAP message
client.ClientCredentials.UserName.UserName = "username";
client.ClientCredentials.UserName.Password = "password";

// call the Add method
var result = client.Add(2, 3);
```

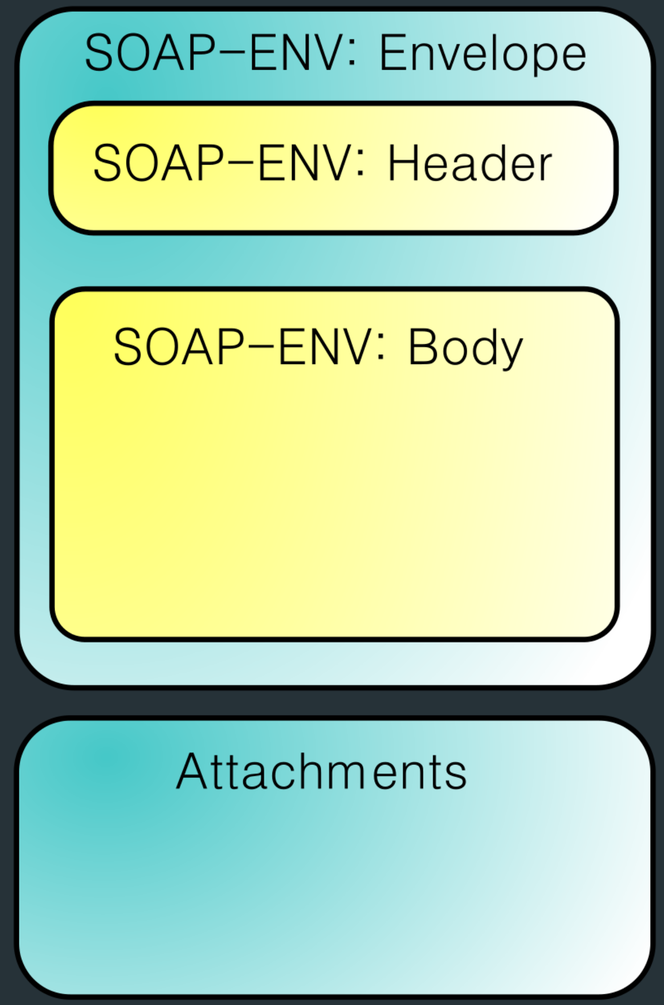
Example Code for WS-Security

```
// create a new client using the Apache CXF library
JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setServiceClass(MyWebService.class);
factory.setAddress("http://example.com/MyWebService");

MyWebService client = (MyWebService) factory.create();

// create a new WSS4J interceptor and set its properties
Map<String, Object> outProps = new HashMap<>();
outProps.put(WSHandlerConstants.ACTION, WSHandlerConstants.USERNAME_TOKEN);
outProps.put(WSHandlerConstants.USER, "myUsername");
outProps.put(WSHandlerConstants.PASSWORD_TYPE, WSConstants.PW_TEXT);
outProps.put(WSHandlerConstants.PW_CALLBACK_CLASS, ClientPasswordCallback.class.getName());

WSS4JOutInterceptor wssOut = new WSS4JOutInterceptor(outProps);

// add the interceptor to the client's endpoint
ClientProxy.getClient(client).getOutInterceptors().add(wssOut);

// call a method on the client
String result = client.myMethod("some parameter");
```

Overview of SOAP Attachments

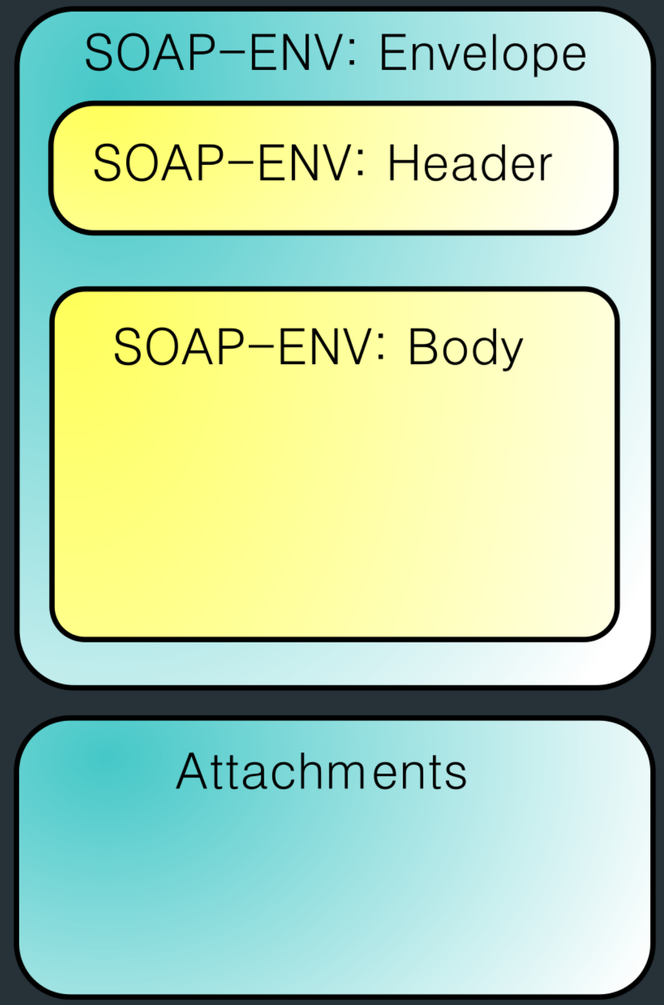
Introduction to SOAP Attachments

- Definition of SOAP Attachments
- Importance of SOAP Attachments
- Brief History of SOAP Attachments



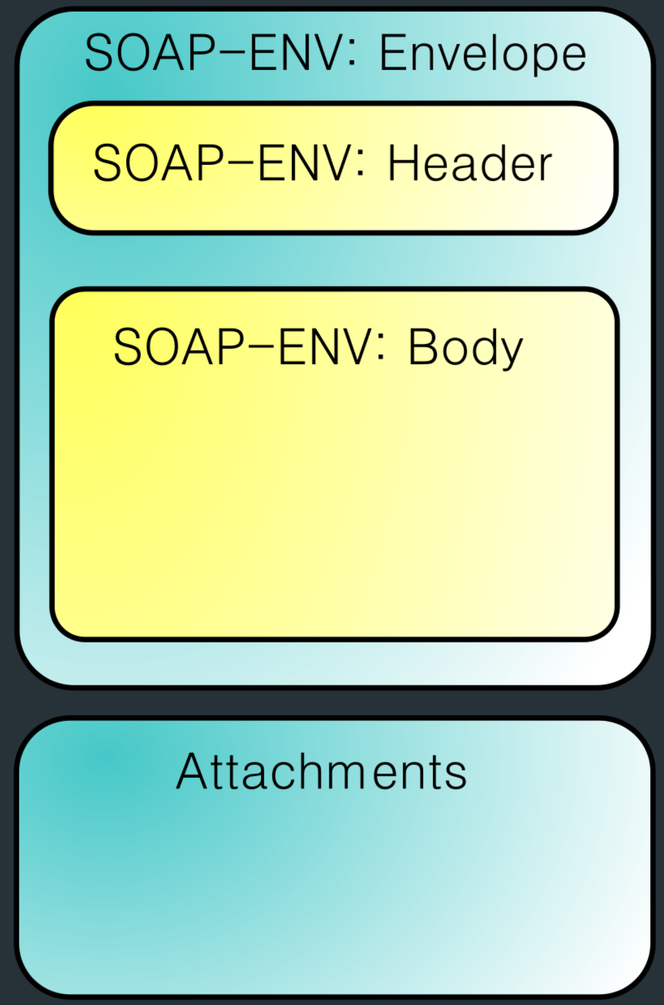
Types of SOAP Attachments

- Inline Attachments
- SwA (SOAP with Attachments)
- MTOM (Message Transmission Optimization Mechanism)



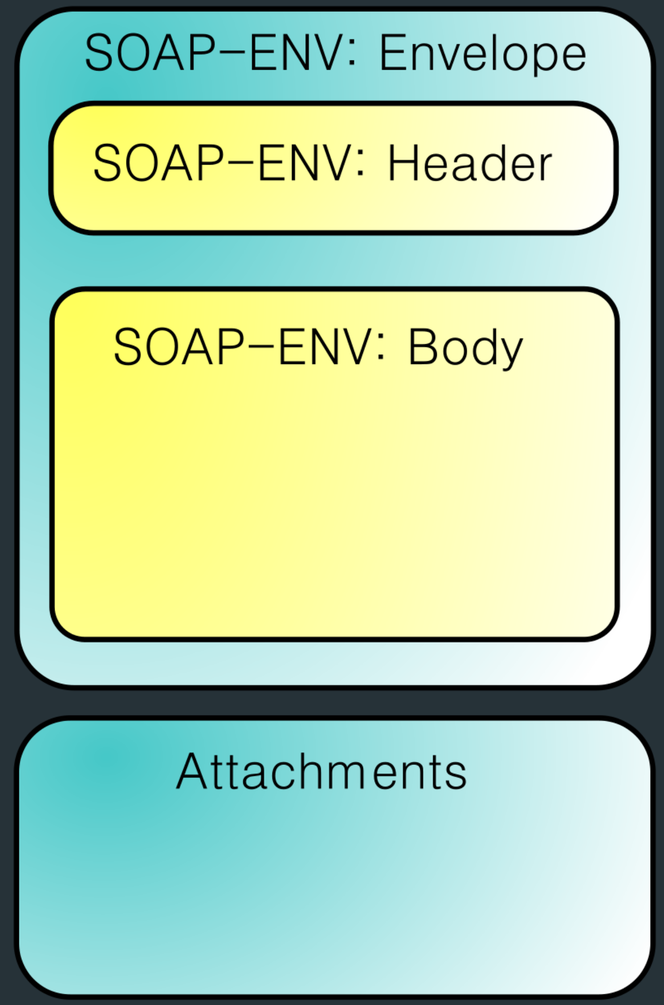
Pros and Cons of SOAP Attachments

- Pros:
  - Enables transmission of complex data types
  - Provides a mechanism to transfer binary or text data
  - Increases performance by reducing message size
- Cons:
  - Increases complexity of the SOAP message
  - Requires additional processing by the SOAP client and server
  - May require additional security measures



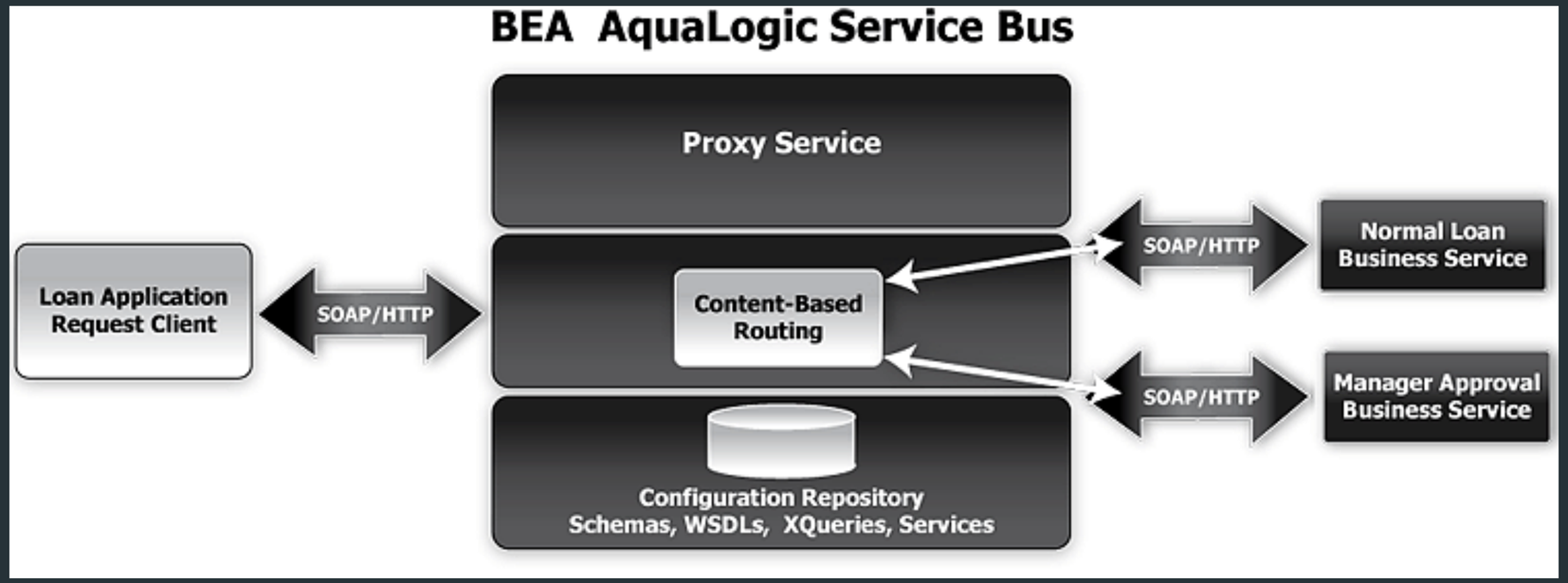
Best Practices for Using SOAP Attachments

- Use MTOM for large binary data
- Use SwA for small to medium-sized binary data
- Use Inline Attachments for text data
- Consider the impact on message size and performance
- Ensure the security of the attachments



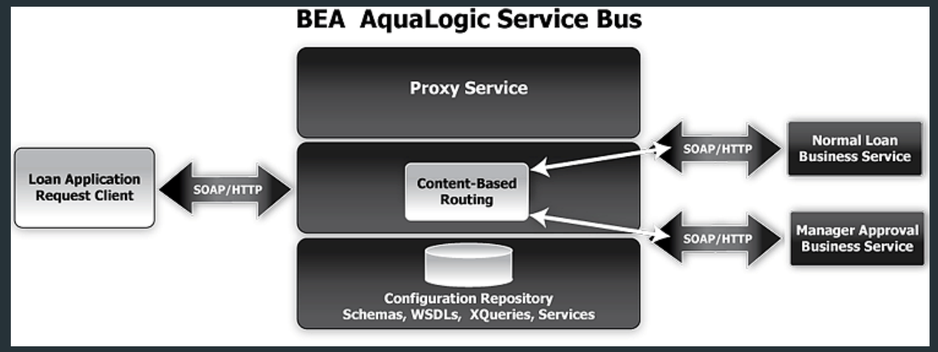
SOAP Routing

How SOAP Routing Works



Benefits of SOAP Routing

- Simplifies network architecture
- Improves scalability
- Enables flexible routing based on message content



Routing Rules

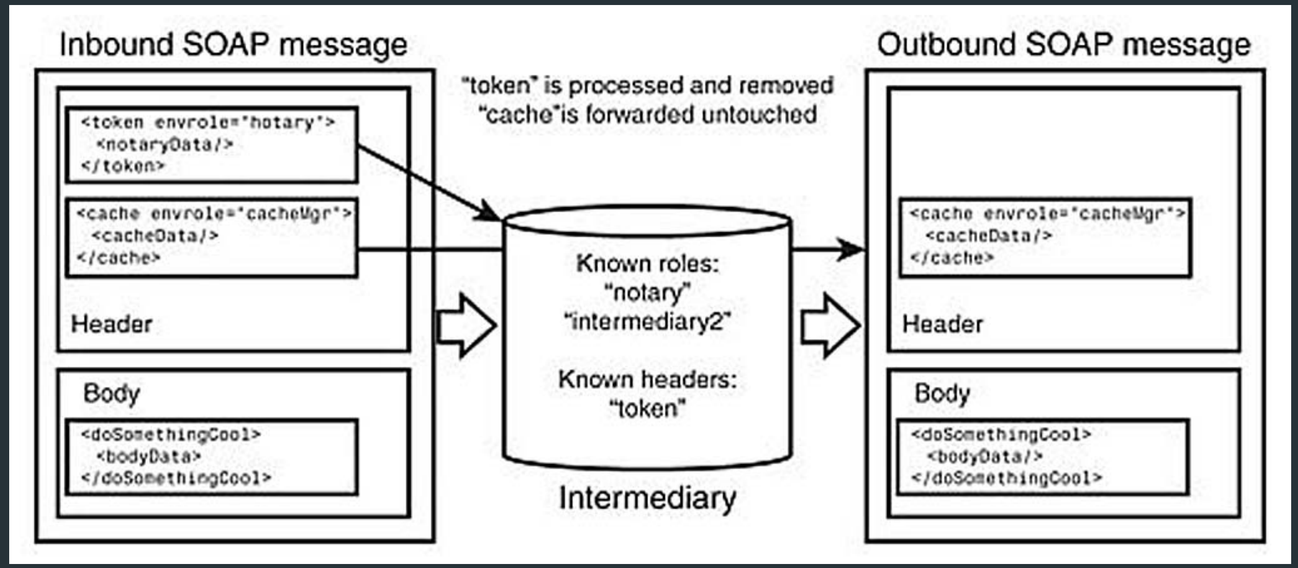
- XPath Routing Rule: This rule allows the routing of a message based on the value of an XPath expression.
- Regular Expression Routing Rule: This rule allows the routing of a message based on the matching of a regular expression.
- Content-Based Routing Rule: This rule allows the routing of a message based on the contents of the message.
- Header Routing Rule: This rule allows the routing of a message based on the value of a SOAP header.

```
<rule-definition name="RouteByCountry">
  <xpath-rule match="/Envelope/Body/Order[country='USA']"/>
  <target-endpoint>http://example.com/ordersUSA</target-endpoint>
</rule-definition>
```

SOAP Intermediaries

Routing Rules

- Message handlers: intercept and modify SOAP messages as they flow through a SOAP engine
- Transport handlers: intercept and modify the underlying transport protocol, such as HTTP, before or after the SOAP message is processed
- Service providers: act as a service endpoint and process SOAP requests like a regular web service



Message Handler in Apache Axis

```
import org.apache.axis.MessageContext;
import org.apache.axis.handlers.BasicHandler;

public class MyHandler extends BasicHandler {

  public void invoke(MessageContext msgContext) {
    // Implement logic for handling the message
    // This could include modifying the message or logging information
  }

}
```

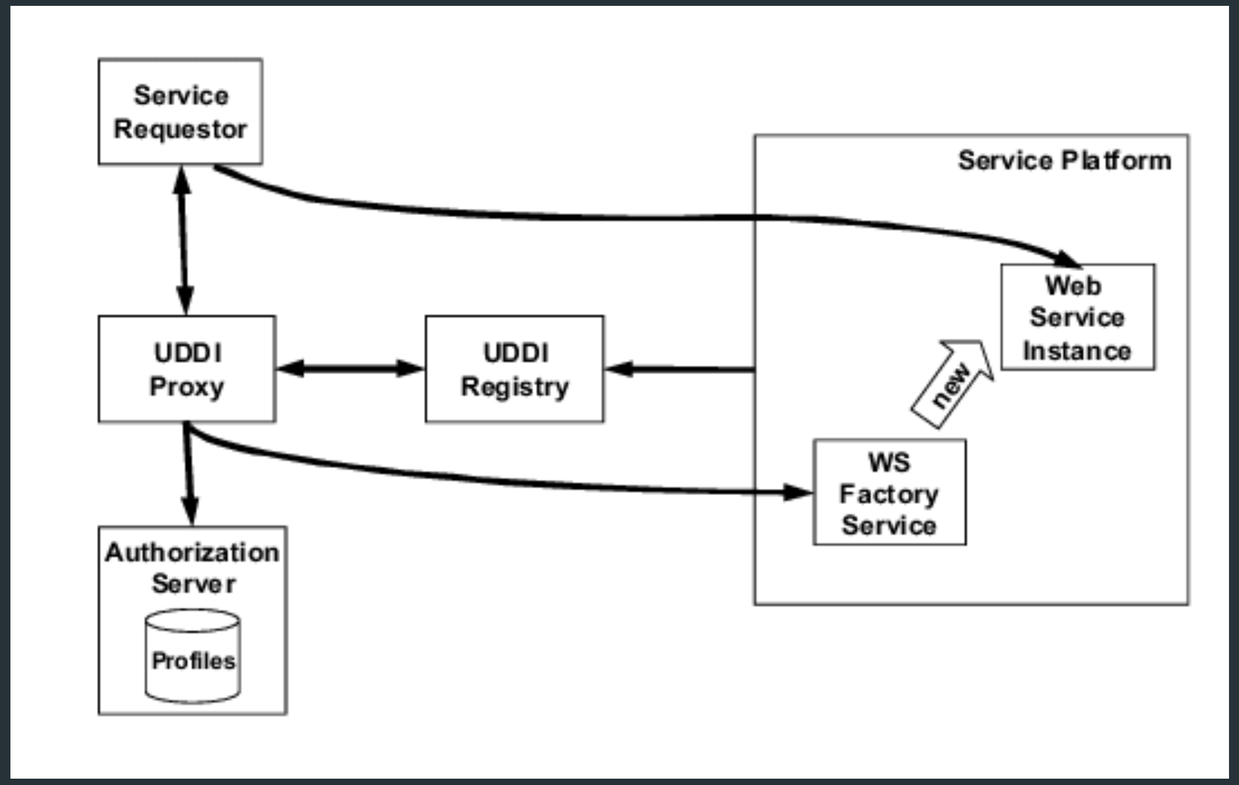
Implementing a transport handler in Apache Axis

```
public class MyTransportHandler extends AbstractHandler {

    public InvocationResponse invoke(MessageContext msgContext) throws AxisFault {
        // get the message
        Message msg = msgContext.getRequestMessage();

        // manipulate the message
        // ...

        // call the next handler in the chain
        return invokeNext(msgContext);
    }
}
```

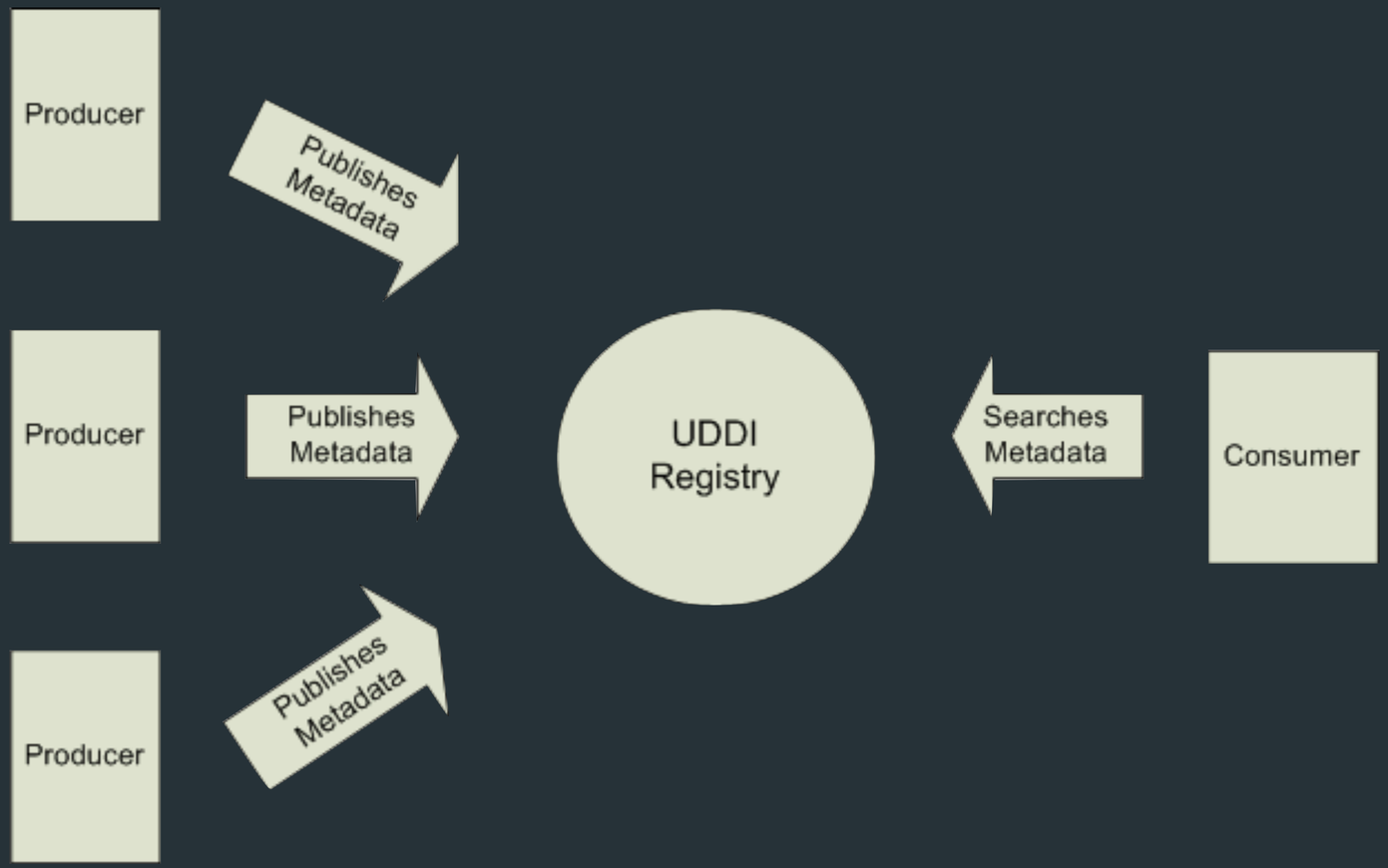
Universal Description, Discovery, and Integration (UDDI) Overview

UDDI Architecture

- UDDI registry
- UDDI server
- UDDI client



UDDI Registry



```
import org.apache.juddi.api_v3.*;
import org.uddi.api_v3.*;
import org.uddi.api_v3.KeyType.*;
import org.uddi.api_v3.Name.*;
import org.uddi.api_v3.Description.*;
import org.uddi.api_v3.FindService.*;
import org.uddi.api_v3.ServiceList.*;
import org.uddi.api_v3.ServiceInfos.*;
import org.uddi.api_v3.ServiceInfo.*;
import org.uddi.api_v3.GetServiceDetail.*;
import org.uddi.api_v3.ServiceDetail.*;

UDDIInquiryPortType inquiry = new UDDIClient().getTransport().getUDDIInquiryService();
FindService fs = new FindService();
Name name = new Name();
name.setValue("Stock Quote Service");
fs.getName().add(name);
ServiceList sl = inquiry.findService(fs);
for (ServiceInfo si : sl.getServiceInfos().getServiceInfo()) {
  GetServiceDetail gsd = new GetServiceDetail();
  gsd.getServiceKey().add(si.getServiceKey());
  ServiceDetail sd = inquiry.getServiceDetail(gsd);
  System.out.println(sd);
}
```

Lecture outcomes

- SOAP Basics
- Benefits
- Architecture
- WSDL
- UDDI