# Web Services

Rushikesh Dumbre

## I. Introduction

- Web service is a method that can be invoked using internet based protocols
- It is hosted on a web server
- It provides interoperability between different platforms and languages
- We can Invoke web service methods using interoperable standards like WSDL

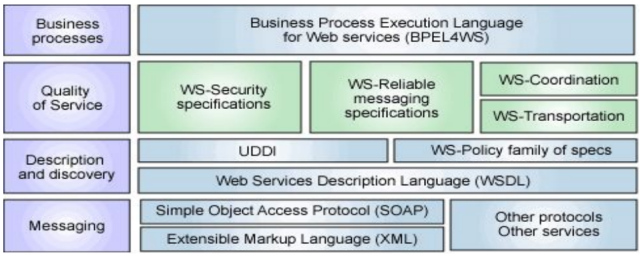


Figure 1. Protocol Stack

## II. SOAP

- It is a protocol specification for exchanging structured information between Web Services
- Relies on XML for message format
- Relies on HTTP/RPC for message negotiation and transmission

### A. Envelope:

- Every SOAP message has a root Envelope element
- xmlns:soap defines the soap envelope
- encodingStyle defines data types used in the document

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

Figure 2. Envelope

### B. Header:

- Contains application specific information
- MustUnderstand indicates whether a header entry is mandatory to process by the recipient
- If "1" then the recipient must recognize the element
- If reciepient fails to recognize the element "MustUnderstand" fault is returned

```
<soap:Header>
  <m:Trans xmlns:m="https://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
```

Figure 3. Header

### C. Body:

- Mandatory element
- Contains information for the reciever
- 'GetPrice' is a web method and 'Item' is a web parameter

```
<soap:Body>
  <m:GetPrice xmlns:m="https://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>
```

Figure 4. Body

### D. Fault

- In case of error, Soap Fault element is returned
- Present inside the Body element
- Returns predefined error code (faultcode), description (faultstring)
- Only one Fault element per SOAP message

```
<SOAP-ENV:Fault>
   <faultcode xsi:type = "xsd:string">SOAP-ENV:Client</faultcode>
   <faultstring xsi:type = "xsd:string">
      Failed to locate method (ValidateCreditCard) in class (examplesCreditCard)
         /usr/local/ActivePerl-5.6/lib/site_perl/5.6.0/SOAP/Lite.pm line 1555.
   </faultstring>
</SOAP-ENV:Fault>
```

Figure 5. Fault

## III. WSDL

- XML based language for describing a web service, its location and the methods provided by it
- Documents organised in two sections: Abstract and Concrete
- Abstract section contains messages and operations
- Concrete section contains messages, operations, binding and transport specific information

### A. Definitions

- Root tag of WSDL document
- Provides name of the web service, target namespace
- TargetNamespace is the logical namespace for the information of about this service

- Tns stands for ‘this namespace’

```
<definitions targetNamespace="http://pack/" name="convertor">
```

Figure 6. Definitions

### B. Types

- Defines the data types used by the service
- Also has the XML Schema Definition location

```
<types>
   <schema targetNamespace="http://example.com/stockquote.xsd"
          xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
         <complexType>
            <all>
               <element name="tickerSymbol" type="string"/>
            </all>
         </complexType>
      </element>
```

Figure 7. Types

### C. Message

- Lists different messages the service exchanges
- Each message defines data elements for its operation
- 2 messages for each operation, Request and Response
- Message name uniquely identifies a message

```
- <message name="add">
    <part name="parameters" element="tns:add"/>
  </message>
- <message name="addResponse">
    <part name="parameters" element="tns:addResponse"/>
  </message>
```

Figure 8. Message

### D. PortType

- Defines web service, operations involved
- Each operation lists the message used for input/output
- Each operation may be in one-way / Request-Response format

```
- <portType name="convertor">
  - <operation name="add">
      <input wsam:Action="http://pack/convertor/addRequest" message="tns:add"/>
      <output wsam:Action="http://pack/convertor/addResponse" message="tns:addResponse"/>
    </operation>
  - <operation name="hello">
      <input wsam:Action="http://pack/convertor/helloRequest" message="tns:hello"/>
      <output wsam:Action="http://pack/convertor/helloResponse" message="tns:helloResponse"/>
    </operation>
  </portType>
```

Figure 9. PortType

### E. Binding

- Name attribute provides unique name among all bindings
- Type attribute defines portType that is bound
- Soap:binding element has 2 attribute, transport and style
- Transport defines protocol used by SOAP
- Style can be ‘rpc’ or ‘document’
- Operation element defines each operation
- Also specifies the encoding of I/O parameters

```
<binding name="convertorPortBinding" type="tns:convertor">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  - <operation name="add">
      <soap:operation soapAction=""/>
    - <input>
        <soap:body use="literal"/>
      </input>
    - <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  - <operation name="hello">
      <soap:operation soapAction=""/>
    - <input>
        <soap:body use="literal"/>
      </input>
    - <output>
        <soap:body use="literal"/>
      </output>
    </operation>
</binding>
```

Figure 10. Binding

### F. Service

- Port
  - Defines an individual endpoint by specifying a single address for a binding
  - Name attribute provides a unique name among all ports
  - Location gives the location of an endpoint
- Name attribute provides a unique name among all services
- If several ports have same portType but different addresses or bindings then these ports are alternatives providing same service

```
<service name="convertor">
  - <port name="convertorPort" binding="tns:convertorPortBinding">
      <soap:address location="http://localhost:8080/convertor/convertor"/>
    </port>
</service>
```

Figure 11. Service

## IV. Web Services Reliable Messaging

- Allows 2 systems to reliably exchange SOAP messages
- Ensure delivery of message over an unreliable infrastructure
- Application Source (AS) send message to Reliable Messaging Source (RMS)
- RMS uses WS-RM protocol to send message to Reliable Messaging Destination (RMD)
- RMD delivers message to Application Destination (AD)

### A. WS-RM Protocol

- WS-RM protocol supports a number of Delivery Assurances
- AtleastOnce:- Message delivered atleast once
- AtMostOnce:- Message delivered atmost once
- ExactlyOnce:- Message delivered exactly once
- InOrder:- Order of messages is maintained between RMS and RMD
- If unable to deliver RMS/RMD raises error

### B. WS-RM Protocol Example

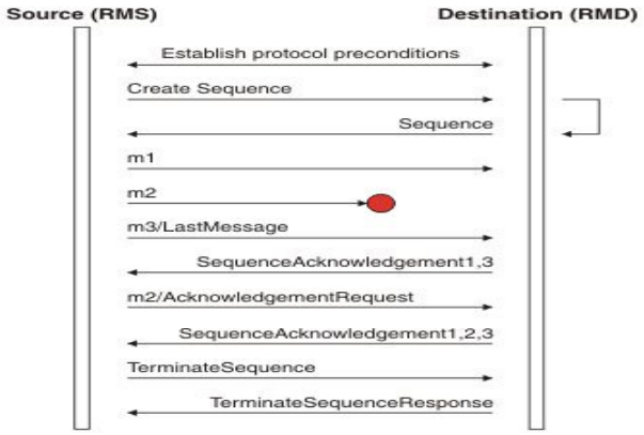


Figure 12. Example

- Preconditions include Policy Exchange, Endpoint Resolution
- RMS requests creation of new sequence
- RMD creates sequence and returns its unique id
- RMS sends message beginning from msg1
- msg2 is lost in the transit
- msg3 includes AckRequested header to ensure it recieves Sequence Acknowledgement
- RMD acknowledges msg1 and msg3
- RMS resends msg2 with same sequence id and msg number
- If both original and duplicate msg are received RMD passes only one of them to AD
- Msg2 also includes a AckRequested header
- RMS sends Terminate Sequence to indicate sequence is complete
- RMD sends back Terminate Sequence Response to RMS

## V. Web Service Security

- Timestamp is attached to the message to verify that the message is not stale
- Username and password are also provided
- Password digest is sent in the message
- This digest is generated using the nonce
- Together the username, nonce and password digest authenticate a user

```
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
               xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
               soap:mustUnderstand="1">
    <wsu:Timestamp wsu:Id="TS-2">
        <wsu:Created>2012-08-29T02:58:29.834Z</wsu:Created>
        <wsu:Expires>2012-08-29T03:03:29.834Z</wsu:Expires>
    </wsu:Timestamp>
    <wsse:UsernameToken wsu:Id="UsernameToken-1">
        <wsse:Username>joe</wsse:Username>
        <wsse:Password
                Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest">
            qOJPIhBbzqsE7dz71CoUjzBJbxs=
        </wsse:Password>
        <wsse:Nonce
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">
            hdSN3eeqCZxr4huNCRTG+A==
        </wsse:Nonce>
        <wsu:Created>2012-08-29T02:58:29.831Z</wsu:Created>
    </wsse:UsernameToken>
</wsse:Security>
```

Figure 13. WSS

## VI. Web Service Addressing

- Defines Endpoint References and Message Information Headers
- Endpoints are processors or resources where messages can be targeted
- References provide information needed to access web services endpoints
- Headers contains source and destination addresses and message identity

```
(001) <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
                  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
(002)   <S:Header>
(003)    <wsa:MessageID>
(004)      uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
(005)    </wsa:MessageID>
(006)    <wsa:ReplyTo>
(007)      <wsa:Address>http://business456.example/client1</wsa:Address>
(008)    </wsa:ReplyTo>
(009)    <wsa:To>http://fabrikam123.example/Purchasing</wsa:To>
(010)    <wsa:Action>http://fabrikam123.example/SubmitPO</wsa:Action>
(011)   </S:Header>
(012)   <S:Body>
(013)     ...
(014)   </S:Body>
(015) </S:Envelope>
```

Figure 14. WSA

### A. Endpoint References

- Consists of:
  - Address:- identifies the endpoint
  - Reference properties:- used to represent endpoints following a different set of policies
  - Reference parameters:- associated with endpoints to facilitate a particular interaction
  - Selected portType:- Qname of the primary portType of the endpoint
  - Service-port:- Qname of identifying WSDL service
  - Policy:- describe the requirements and capabilities of endpoint

### B. Message Information Headers

- Consists of:
  - Destination:- Address of the intended receiver of the message
  - Source Endpoint:- Endpoint reference of the message origin
  - Reply Endpoint:- Endpoint reference of the intended receiver for replies of this message
  - Fault Endpoint:- Reference of receiver of the faults related to this message
  - Action:- An identifier that uniquely identifies semantics implied by the message
  - Message id:- Uniquely identifies message in time and space
  - Relationship:- Pair of values indicating how this message relates to another message

## VII. Security Assertion Markup Language (SAML)

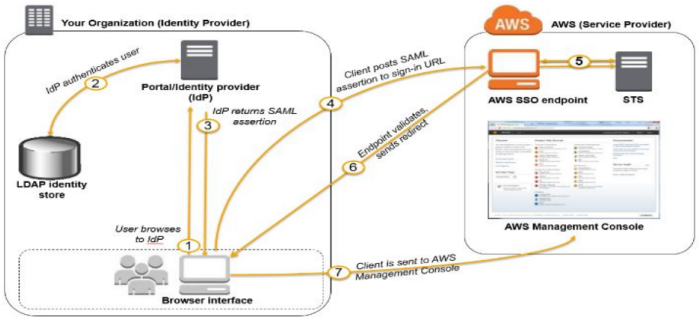


Figure 15. AWS SAML

- The user browses to your organization's portal and selects the option to go to the AWS Management Console.
- The portal is typically a function of your identity provider (IdP) that handles the exchange of trust between your organization and AWS.
- The portal verifies the user's identity in your organization.
- The portal generates a SAML authentication response that includes **assertions** that identify the user and include attributes about the user.
- You can also configure your IdP to include a SAML assertion attribute called SessionDuration that specifies how long the console session is valid.
- The portal sends this response to the client browser.
- The client browser is redirected to the AWS **single sign-on** endpoint and posts the SAML assertion
- The endpoint requests temporary security credentials and creates a console sign-in URL using those credentials
- AWS sends the sign-in URL back to the client as a redirect
- The client browser is redirected to the AWS Management Console
- Three types of assertion statements:
  - Authentication:- Asserts to service provider that principal authenticated with identity provider
  - Attribute:- Asserts that a subject is associated with certain attributes. It is a Name-Value pair
  - Authorization Decision:- Asserts that subject is permitted to perform action A on resource R given evidence E

## VIII. eXtensible Access Control Markup Language ( XACML)

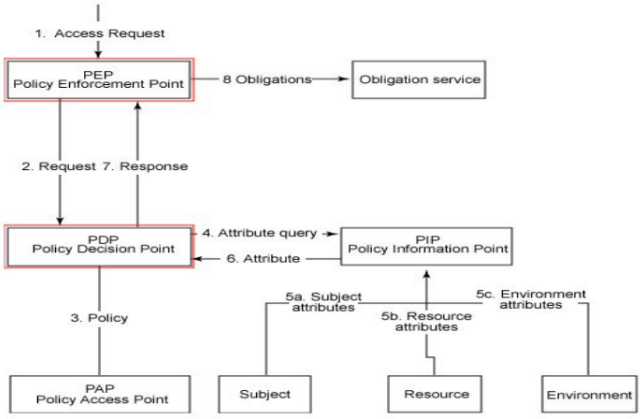


Figure 16. XACML

- Defines the rules necessary for authorization decisions
- Policy Enforcement Point (PEP) recieves the request for authorization
- PEP sends XACML request to Policy Decision Point (PDP)
- PDP evaluates request and sends back response
- Response can be permitted or denied
- PDP arrives at decision after evaluating relevant policies
- PDP uses Policy Access Point (PAP) to retrieve policies
- PAP writes policies and sets policy
- PDP may invoke Policy Information Point (PIP)
- PIP retrieves attribute values related to subject, resource or environment
- PDP then sends the authorization decision, it arrived at, to PEP

## IX. Representational State Transfer

- REST, or REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other.
- REST-compliant systems, often called RESTful systems, are characterized by how they are **stateless** and **separate the concerns of client and server.**
- In the REST architectural style, the implementation of the client and the implementation of the server can be done independently without each knowing about the other.
- This means that the code on the client side can be changed at any time without affecting the operation of the server, and the code on the server side can be changed without affecting the operation of the client.
- As long as each side knows what format of messages to send to the other, they can be kept modular and separate.
- By using a REST interface, different clients hit the same REST endpoints, perform the same actions, and receive the same responses.
- Systems that follow the REST paradigm are stateless, meaning that the server does not need to know anything about what state the client is in and vice versa.

- In this way, both the server and the client can understand any message received, even without seeing previous messages.
- This constraint of statelessness is enforced through the use of resources, rather than commands.
- Because REST systems interact through standard operations on resources, they do not rely on the implementation of interfaces.

### A. Making Requests

- REST requires that a client make a request to the server in order to retrieve or modify data on the server. A request generally consists of:
  - an HTTP verb, which defines what kind of operation to perform
  - a header, which allows the client to pass along information about the request
  - a path to a resource
  - an optional message body containing data

### B. HTTP verbs

- There are 4 basic HTTP verbs we use in requests to interact with resources in a REST system:
  - GET — retrieve a specific resource (by id) or a collection of resources
  - POST — create a new resource
  - PUT — update a specific resource (by id)
  - DELETE — remove a specific resource by id

### C. Header

- In the header of the request, the client sends the type of content that it is able to receive from the server.
- This is called the Accept field, and it ensures that the server does not send data that cannot be understood or processed by the client.
- The options for types of content are MIME Types
- For example,
  - a text file containing HTML would be specified with the type text/html
  - If this text file contained CSS instead, it would be specified as text/css.
  - A generic text file would be denoted as text/plain.

### D. Paths

- A path like fashionboutique.com/customers/223/orders/12 is clear in what it points to, the order with id 12 for the customer with id 223.
- If we are trying to access a single resource, we would need to append an id to the path. For example: GET fashionboutique.com/customers/:id — retrieves the item in the customers resource with the id specified. DELETE fashionboutique.com/customers/:id — deletes the item in the customers resource with the id specified.

### E. Content Types

- In cases where the server is sending a data payload to the client, the server must include a content-type in the header of the response.
- This content-type header field alerts the client to the type of data it is sending in the response body.
- These content types are MIME Types, just as they are in the accept field of the request header.
- The content-type that the server sends back in the response should be one of the options that the client specified in the accept field of the request.

### F. Examples

```
POST http://fashionboutique.com/customers
Body:
{
  “customer”: {
    “name” = “Scylla Buss”
    “email” = “scylla.buss@codecademy.org”
  }
}
```

Figure 17. Request

```
201 (CREATED)
Content-type: application/json
```

Figure 18. Response