

# Web Services 2

# Contents

- WSDL
- REST architecture



# Contents

- WSDL
- REST architecture



# Role of WSDL

- SOAP Web Service Definition by W3C: “a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts.”
- Given that the client and service are different entities – we need to know:
  - where is the service?
  - what does it offer me?
  - what XML should I send/receive to interact with it?
  - how can I expose it to others?

# WSDL

- WSDL stands for **Web Services Description Language**
- WSDL is written in XML
- WSDL is used to describe web services
- WSDL is used to locate Web services



# WSDL Document Structure

```
<definitions>
  <types>
    definition of types.....
  </types>
  <message>
    definition of a message....
  </message>
  <portType>
    definition of a port.....
  </portType>
  <binding>
    definition of a binding....
  </binding>
  <service>
    definition of the service.....
  </service>
</definitions>
```

# WSDL Document Structure (cont.)

- Abstract part
  - **Types**— a container for data type definitions using some type system (such as XSD).
  - **Message**— an abstract, typed definition of the data being communicated.
  - **Port Type**—an abstract set of operations supported by one or more endpoints.
    - **Operation**— an abstract description of an action supported by the service.
- Concrete part
  - **Binding**— a concrete protocol and data format specification for a particular port type.
  - **Service**— a collection of related endpoints.
    - **Port**— a single endpoint defined as a combination of a binding and a network address.

# WSDL Types

- The **<types>** element defines the data type that are used by the web service
- WSDL uses XML Schema syntax to define data types





# WSDL Messages

- The **<message>** element defines the data elements of an operation
- The messages let clients know about the input and output. Each message is a series of name/type pairs
- **Example:**

```
<message name="getTermRequest">  
  <part name="term" type="xs:string"/>  
</message>
```


```
<message name="getTermResponse">  
  <part name="value" type="xs:string"/>  
</message>
```

# WSDL PortType

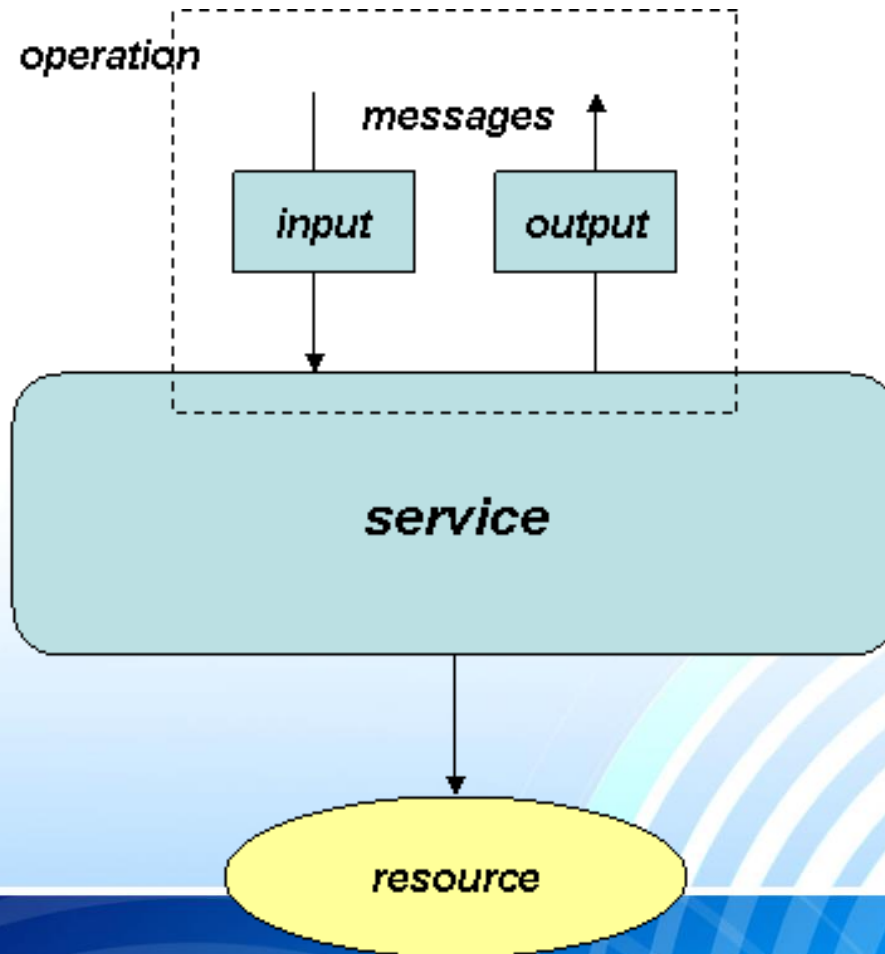
- The **<portType>** element describes the operations that can be performed, and the messages that are involved
- Each port type is a group of operations
- Example: 

```
<portType name="glossaryTerms">  
  <operation name="getTerm">  
    <input message="getTermRequest"/>  
    <output message="getTermResponse"/>  
  </operation>  
</portType>
```

# WSDL PortType (cont.)

Type	Definition
One-way	The operation can receive a message but will not return a response 
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response
Notification	The operation can send a message but will not wait for a response

# Messages and operations





# WSDL Bindings

- The <binding> element defines the message format and protocol details for a web service
- The binding element has the *name* and the *type* attributes
  - The name attribute defines the name of the binding
  - The type attribute points to the port for the binding.



## WSDL Bindings (cont.)

- The soap:binding element has the *style* and the *transport* attributes  
  - The *style* attribute can be "rpc" or "document"
- The *operation* element defines each operation that the port exposes

```
<binding type="glossaryTerms" name="b1">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
    <soap:operation soapAction="http://example.com/getTerm"/>
    <input><soap:body use="literal"/></input>
    <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

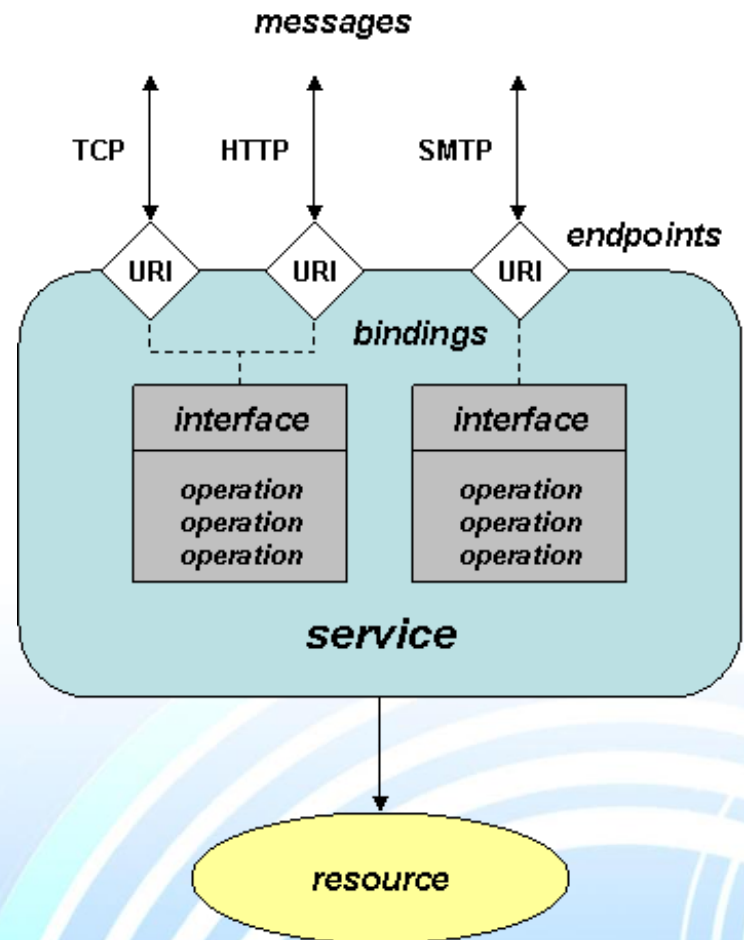
# WSDL Service and Ports

- The <service> element defines a collection of **ports**, or **endpoints**, and which binding to use
- The <port> element defines the connection point to a web service
  - Each port has a name and is assigned to a binding
  - Within the port element, the address details is defined for that specific binding

```
<service name="MathService">  
  <port name="MathEndpoint" binding="y:MathSoapHttpBinding">  
    <soap:address location="http://localhost/math/math.asmx"/>  
  </port>  
</service>
```

# Service and Bindings

- A service can have multiple bindings for a given interface/portType
- Each binding can be only accessible at a unique URL (endpoint/port)



# Contents

- WSDL
- REST architecture



# REST (REpresentational State Transfer)

- **REST is an architecture**
  - NOT a technology or standards specification or a protocol
- While REST is not a standard, it uses standards:
  - HTTP
  - URL
  - XML/HTML/JSON/etc
  - MIME Types or Media types such as text/xml, image/gif, application/json, audio/mpeg



# REST Constraints

- REST constraints are design rules that are applied to establish the distinct **characteristics** of the REST architectural style.
- The formal REST constraints are:
  - Client/Server
  - Stateless
  - Cacheable
  - Uniform Interface
  - Layered Systems
  - Code-On-Demand

# Client/Server

- This essentially means that client application and server application **MUST** be able to evolve separately without any dependency
  - separating the user interface concerns (e.g. of clients) from the data storage concerns (e.g. of servers)
  - allows the components to evolve independently
  - improve the **portability** of the user interface across multiple platforms
  - improves **scalability** by simplifying the server components

# Stateless

- Make all client-server interaction stateless. Server will not store anything about latest HTTP request client made. It will treat each and every request as **new. No session, no history.**
  - each request from client to server must contain all of the information necessary to understand the request
  - it cannot take advantage of any stored context on the server
  - Improves scalability and reliability

# Cacheable

- In REST, caching shall be applied on resources when applicable and then these resources **MUST** declare themselves cacheable.
- Caching brings performance improvement for client side, and better scope for **scalability** for server because load has reduced.
- If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests
- Reliability might be **reduced** if stale data within the cache differs significantly from the data that would have been obtained had the request been sent directly to the server

# Uniform interface

- APIs interface **MUST** be decided for resources inside system which are exposed to API consumers and follow religiously. A resource in system should have **only one** logical URI, and then should provide way to fetch related or additional data.
  - all resources are accessed with a generic interface (e.g., HTTP **GET, POST, PUT, DELETE**)
  - the overall system architecture is simplified
  - But “degrades efficiency, since information is transferred in a standardized form rather than one which is specific to an application's needs”



# Guiding principles of the uniform interface

- Identifying the resource
- Resource representation
- Self-descriptive Messages
- Hypermedia as the Engine of Application State (HATEOAS)

# Identifying the resource

- Each resource must have a specific and cohesive URI to be made available
- **REST is based on these notions:**
  - **A resource**
    - Any information that can be named can be a resource
    - A document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person)
    - Nouns instead of verbs
  - **A resource identifier**
    - Each resource accessible by a URI/URL
    - A resource identifier (URI) identifies a particular resource
    - 'Nice' URIs

# Identifying the resource: Example

HTTP/1.1 GET

<https://www.googleapis.com/customsearch/v1?parameters>



# REST methods

Method	Meaning
GET	Retrieve a copy of a Resource
DELETE	Remove a Resource
POST	Create or sometimes update
PUT	Update a Resource or sometimes create
PATCH	Update only part of a resource

- The difference between PUT and POST
  - POST is not idempotent but PUT is
  - The client uses PUT when it's in charge of deciding which URI the new resource should have.
  - The client uses POST when the server is in charge of deciding which URI the new resource should have

# Resource representation

- A representation of a resource
  - A document capturing current state of a resource
  - A resource can have different representations
  - Client can specify which representation can accept (e.g. in http accept header)
  - A resource representation provides links for navigation and accessing further resources
- This representation can be in HTML, XML, JSON, TXT, and more.



# Self-descriptive Messages

- Each message includes enough information to describe how to process the message.
- Beyond what we have seen so far, the passage of meta information is needed (metadata) in the request and response. Some of this information are: HTTP response code, Host, Content-Type etc.

# Hypermedia as the Engine of Application State (HATEOAS)

- It means that hypertext should be used to find your way through the API
- For example, below given JSON response may be from an API like HTTP GET

`http://api.domain.com/management/departments`

```
1 {  
2   "departmentId": 10,  
3   "departmentName": "Administration",  
4   "locationId": 1700,  
5   "managerId": 200,  
6   "links": [  
7     {  
8       "href": "10/employees",  
9       "rel": "employees",  
10      "type": "GET"  
11     }  
12   ]  
13 }
```

# Layered system

- REST allow you to use a layered system architecture where you deploy the APIs
  - The architecture is transparent. A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.
  - This can be used to improve system scalability by enabling load balancing of services across multiple networks and processors

# Code on demand

- This *optional* constraint is primarily intended to allow logic within clients (such as Web browsers) to be updated independently from server-side logic.
- Allows client functionality to be extended by downloading and executing code in the form of applets or scripts

# REST vs SOAP

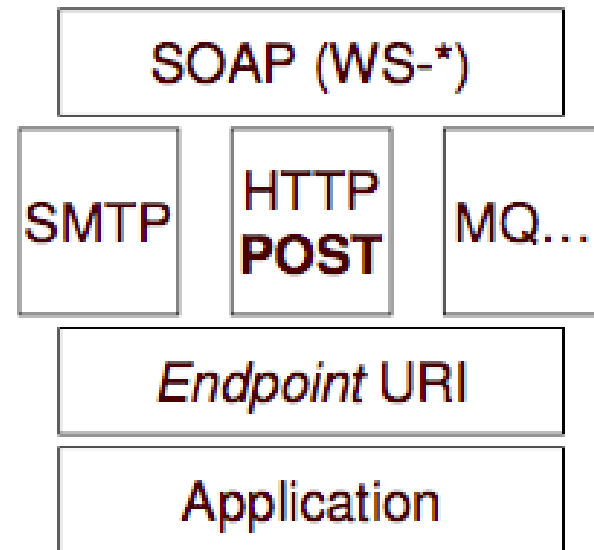
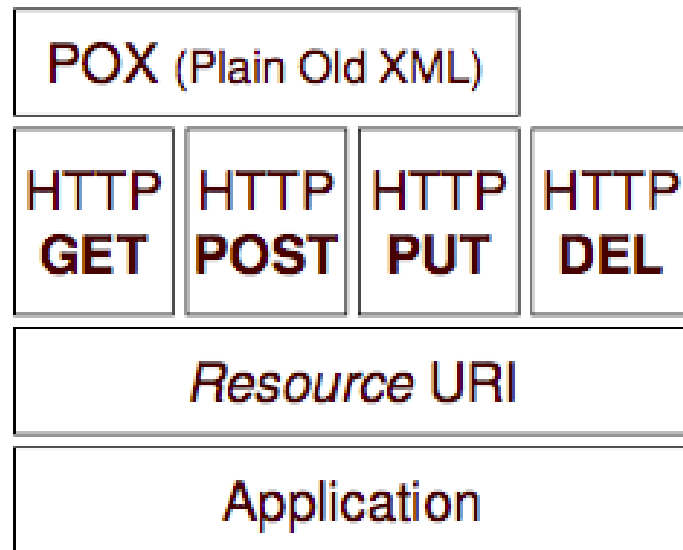
- Main differences
- Conceptual Comparison
- Technology Comparison
- Protocol Comparison





# REST vs SOAP: Main differences

- “The Web is the universe of globally accessible information”  
(Tim Berners Lee)
  - Applications should publish their data on the Web (through URI)
- “The Web is the universal transport for messages”
  - Applications get a chance to interact but they remain “outside of the Web”



# REST vs SOAP: Main differences

REST	SOAP
REST operates through a solitary, consistent interface to access named resources.	SOAP operates through different interfaces
Mainly accesses data	Mainly performs operations through a more standardized set of messaging patterns
Direct point-to-point communication	Works well in distributed enterprise environments

# REST vs SOAP: Conceptual Comparison

REST	SOAP
The Web as an open publishing medium	The Web as a cross enterprise communication medium
Resource Documents originally meant for human consumption	Business Documents for business process driven consumption
Resource oriented access informational resources	Activity/Service oriented – actions that may be performed orthogonally to the resources upon which they act

# REST vs SOAP: Technology Comparison

REST	SOAP
User-driven interactions via forms	Orchestrated reliable event flows
Few operations (generic interface) on many resources	Many operations (service interface) on few resources
URI: Consistent naming mechanism for resources	Lack of standard naming mechanism
Focus on scalability and performance of large scale distributed hypermedia systems	Focus on design of integrated (distributed) applications

# REST vs SOAP: Protocol Comparison

REST	SOAP
XML in – XML out (with POST)	SOAP in – SOAP out (with POST)
URI in – XML out (with GET)	Strong Typing (XML Schema)
“Self-Describing” XML	“Transport independent”
HTTP only	Heterogeneity in QoS needs. Different protocols may be used
HTTP/SSL is enough – no need for more standards	HTTP as a transport protocol
HTTP is an application protocol	Synchronous and Asynchronous
Synchronous	Foundation for the whole WS* advanced protocol stack
Do-it-yourself when it comes to “reliable message delivery”, “distributed transactions”	



# Reference

- <https://www.vs.inf.ethz.ch/edu/WS0405/VS/VS-050124.pdf>
- [https://www.tutorialspoint.com/uddi/uddi\\_overview.htm](https://www.tutorialspoint.com/uddi/uddi_overview.htm)
- <http://whatisrest.com/>