



Click to add text

Architectural styles for cloud applications

Need For Architectural Styles

- ▶ Cloud is based on client –server architecture
- ▶ When application is created
 - ▶ communication between two systems of different architecture should be in a structured format.
 - ▶ Neutrality:-ability of application protocol to use different transport protocols ,TCP/UDP
 - ▶ Independent :- ability to accommodate different programming styles.
 - ▶ Extensibility:-ability to incorporate other functionalities like security.

SOA(Service oriented Architecture)


- ▶ SOA, or service-oriented architecture, defines a way to make software components reusable and interoperable via service interfaces. Services use common interface standards and an architectural pattern so they can be rapidly incorporated into new applications.
- ▶ This removes tasks from the application developer who previously redeveloped or duplicated existing functionality or had to know how to connect or provide interoperability with existing functions.

Service

- ▶ Each service in an SOA embodies the code and *data* required to execute a complete, discrete business function (e.g. checking a customer's credit, calculating a monthly loan payment, or processing a mortgage application).
- ▶ The service interfaces provide loose coupling, meaning they can be called with little or no knowledge of how the *service* is implemented underneath, reducing the dependencies between applications.

Interface

- ▶ This interface is a service contract between the service provider and service consumer. Applications behind the service interface can be written in Java, Microsoft .Net, Cobol or any other programming language, supplied as packaged software applications by a vendor (e.g., SAP), SaaS applications (e.g., Salesforce CRM), or obtained as open source applications. Service interfaces are frequently defined using Web Service Definition Language (WSDL) which is a standard tag structure based on xml (extensible markup language).

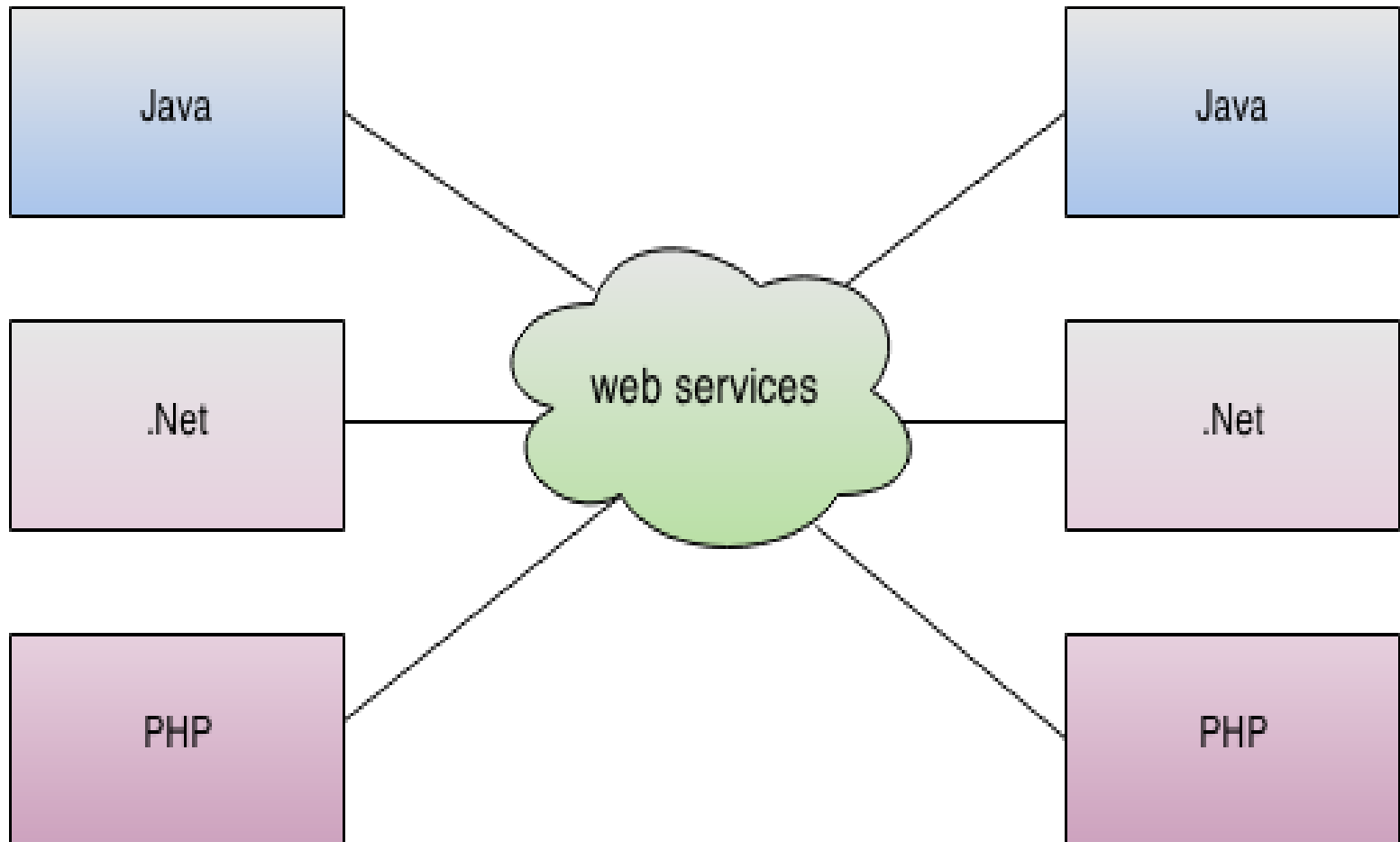
- 
- ▶ The services are exposed using standard network protocols—such as SOAP (simple object access protocol)/HTTP or Restful HTTP (JSON/HTTP)—to send requests to read or change data. Service governance controls the lifecycle for development and at the appropriate stage the services are published in a *registry* that enables developers to quickly find them and reuse them to assemble new applications or business processes.
 - ▶ These services can be built from scratch but are often created by exposing functions from legacy systems of record as service interfaces.

SOA Vs Microservices

- ▶ SOA is an integration architectural style and an enterprise-wide concept. It enables existing applications to be exposed over loosely-coupled interfaces, each corresponding to a business function, that enables applications in one part of an extended enterprise to reuse functionality in other applications.
- ▶ Microservices architecture is an application architectural style and an application-scoped concept. It enables the internals of a single application to be broken up into small pieces that can be independently changed, scaled, and administered. It does not define how applications talk to one another—for that we are back to the enterprise scope of the service interfaces provided by SOA.

Web service

- ▶ Web Services define a platform and language-independent standard based on XML to communicate within distributed systems.

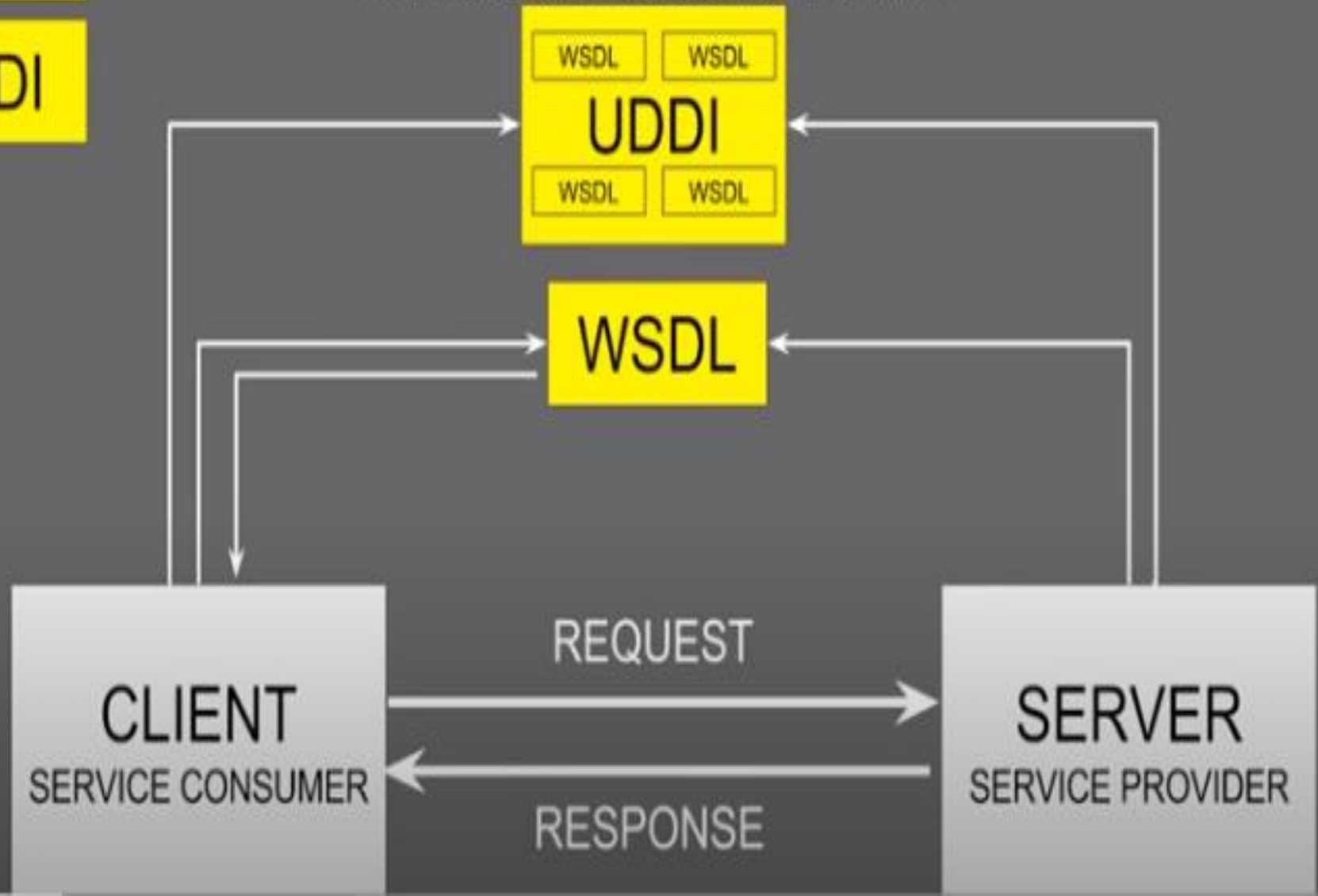


WEB SERVICES

Services available over the web

WSDL

UDDI



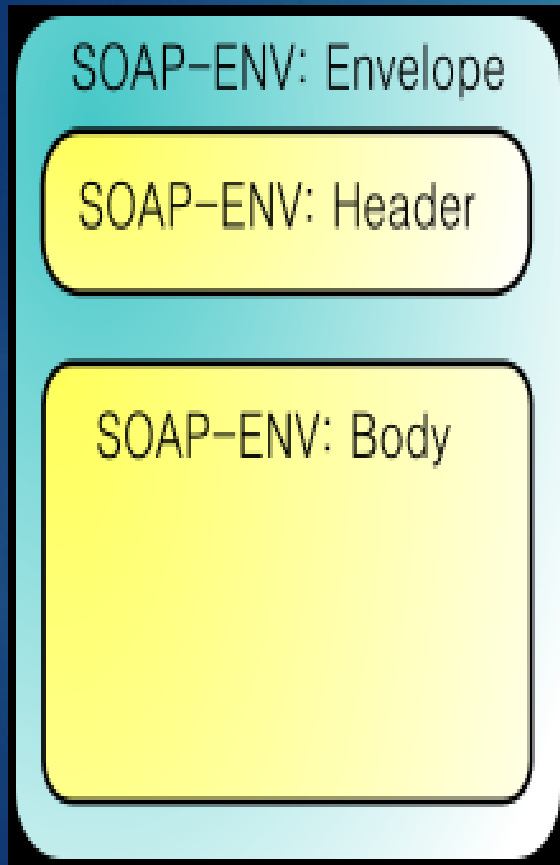
Components of Web Service

- ▶ **WSDL(Web Services Description Language)**
 - ▶ is an XML-based language for describing web services and how to access them.
- ▶ **SOAP** is an XML-based protocol for exchanging information between computers.
- ▶ **UDDI(Universal Description, Discovery, and Integration)**
 - ▶ is an XML-based standard for describing, publishing, and finding web services.

Simple Object Access Protocol (SOAP)

- ▶ A protocol specification for exchanging structured information in the implementation of Web Services in computer networks.
- ▶ It relies on XML Information Set for its message format
- ▶ Relies on other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.
- ▶ SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built.

SOAP Structure



- An Envelope element that identifies the XML document as a SOAP message
- A Header element that contains header information
- A Body element that contains call and response information

SOAP Skeleton

```
<?xml version="1.0"?>
< soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">

  < soap:Header>
  ...
< /soap:Header>

  < soap:Body>
  ...
    <soap:Fault>
    ...
    </soap:Fault>
  < /soap:Body>

< /soap:Envelope>
```


SOAP Request Message

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"
<!-- -->
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

SOAP Response Message

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

Advantages

- ▶ SOAP is versatile enough to allow for the **use of different transport protocols**. The standard stacks use HTTP as a transport protocol, but other protocols such as SMTP can also be used.
- ▶ Since the SOAP model tunnels fine in the HTTP post/response model, it can tunnel easily over existing firewalls and proxies, without modifications to the SOAP protocol, and can use the existing infrastructure.

Disadvantages

- ▶ Because of the verbose XML format, SOAP can be considerably slower than competing middleware technologies such as CORBA .This may not be an issue when only small messages are sent.
- ▶ SOAP only supports XML and other lightweight formats like JSON are not supported
- ▶ When relying on HTTP as a transport protocol and not using WS-Addressing, the roles of the interacting parties are fixed. Only one party (the client) can use the services of the other.

Representational State Transfer (REST)

- ▶ Representational state transfer (REST) is an architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system.
- ▶ REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements.
- ▶ The REST architectural style is also applied to the development of Web services as an alternative to other distributed-computing specifications such as SOAP.

Representational State Transfer(REST)

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: C."


- Dr. Roy T. Fielding

Architectural Constraints

- ▶ **Resource identification through URI:** Resources are identified by their URIs (typically links on internet). So, a client can directly access a RESTful Web Services using the URIs of the resources (same as you put a website address in the browser's address bar and get some representation as response).
- ▶ **Uniform interface:** Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.
- ▶ **Client-Server:** A clear separation concerns is the reason behind this constraint. Separating concerns between the Client and Server helps improve portability in the Client and Scalability of the server components.
- ▶ **Stateless:** each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.

Architectural Constraints

- ▶ **Cache:** to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.
- ▶ **Named resources** - the system is comprised of resources which are named using a URL.
- ▶ **Interconnected resource representations** - the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.
- ▶ **Layered components** - intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.
- ▶ **Self-descriptive messages:** Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.

- 
- ▶ One can characterize applications conforming to the REST constraints as "RESTful". If a service violates any of the required constraints, it cannot be considered RESTful.
 - ▶ Complying with these constraints, and thus conforming to the REST architectural-style, enables any kind of distributed hypermedia system to have desirable emergent properties, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability.

Concept

- ▶ REST was initially described in the context of HTTP, but it is not limited to that protocol.
- ▶ RESTful architectures may be based on other Application Layer protocols if they already provide a rich and uniform vocabulary for applications based on the transfer of meaningful representational state.
- ▶ RESTful applications maximize the use of the existing, well-defined interface and other built-in capabilities provided by the chosen network protocol, and minimize the addition of new application-specific features on top of it.

RESTful API HTTP methods

- ▶ **GET**: It defines a reading access of the resource without side-effects.
- ▶ **POST**: It creates a new resource.
- ▶ **PUT** :It updates an existing resource or creates a new resource. It must also be idempotent
- ▶ **DELETE** : It removes the resources.

RESTful API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as <code>http://example.com/resources</code>	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	Delete the entire collection.
Element URI, such as <code>http://example.com/resources/item17</code>	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it doesn't exist, create it.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.	Delete the addressed member of the collection.

Common SOAP 1.1



Common SOAP 1.1

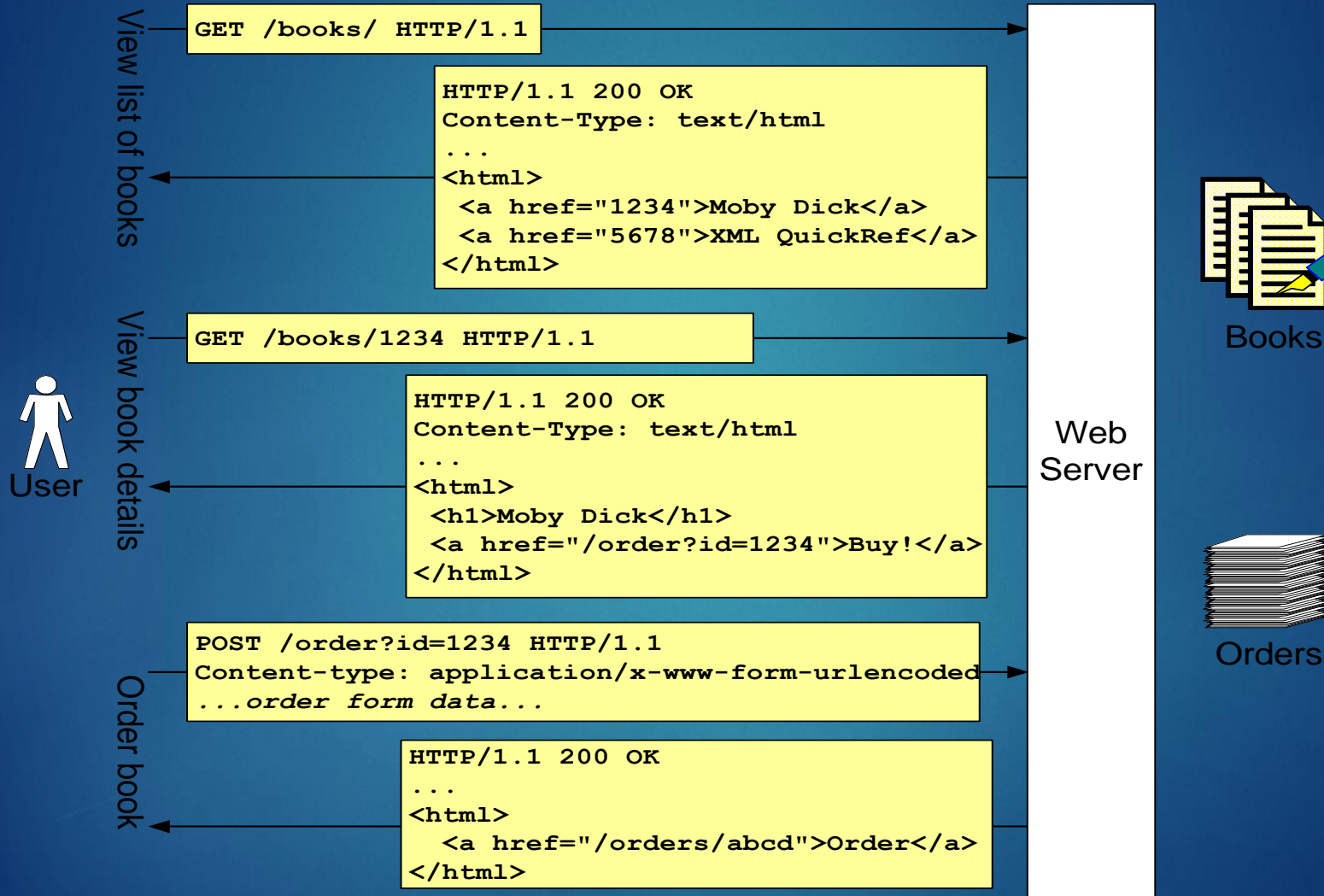
Order Book

```
POST /Store.asmx HTTP/1.1
...
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="...">
  <soap:Body>
    <OrderBook xmlns="...">
      <BookID>1234</BookID>
      <Payment>...</Payment>
      <Shipping>...</Shipping>
    </OrderBook>
  </soap:Body>
</soap:Envelope>
```

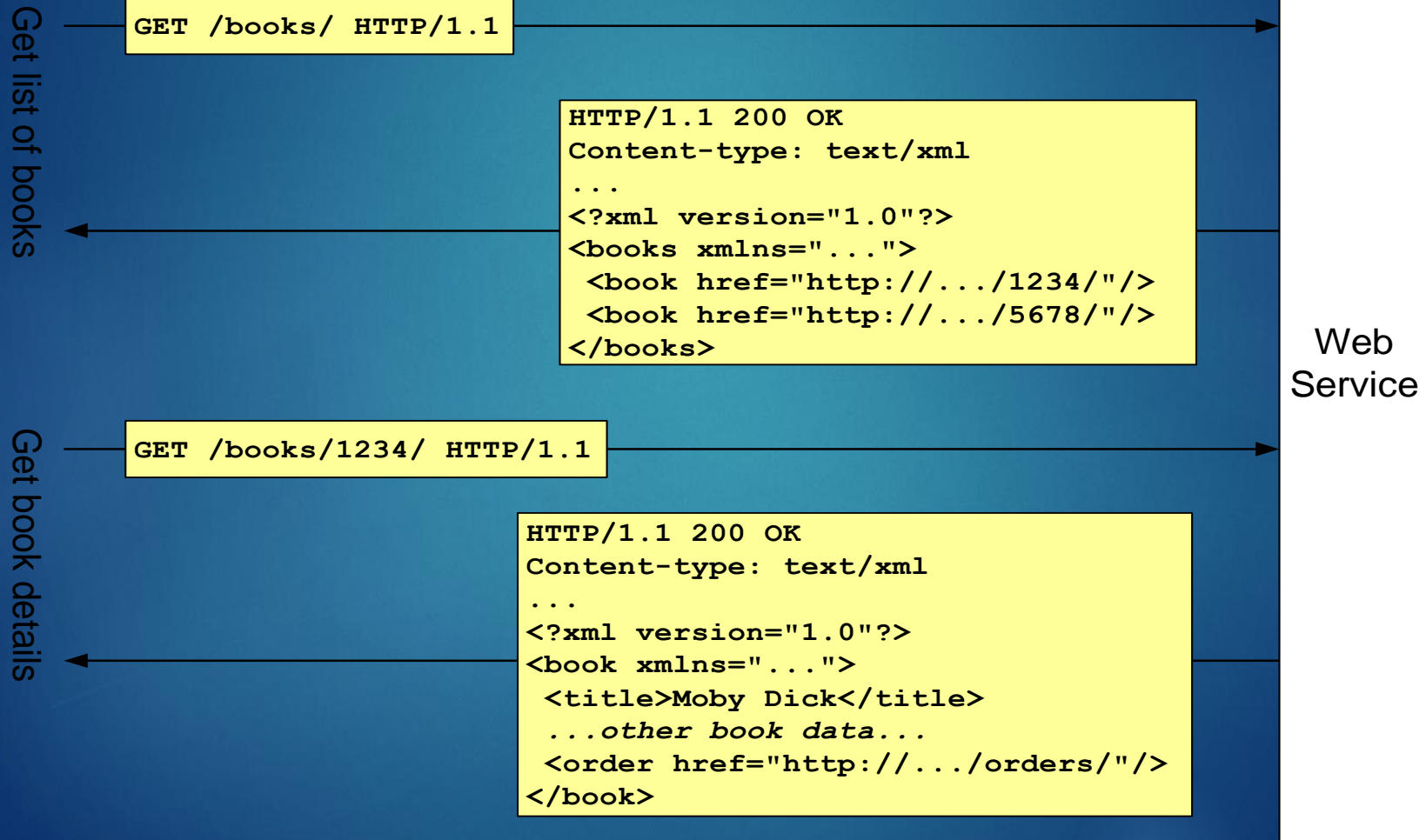
Web
Service

```
HTTP/1.1 200 OK
...
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="...">
  <soap:Body>
    <OrderBookResponse xmlns="...">
      <OrderID>abcd</OrderID>
    </OrderBookResponse>
  </soap:Body>
</soap:Envelope>
```

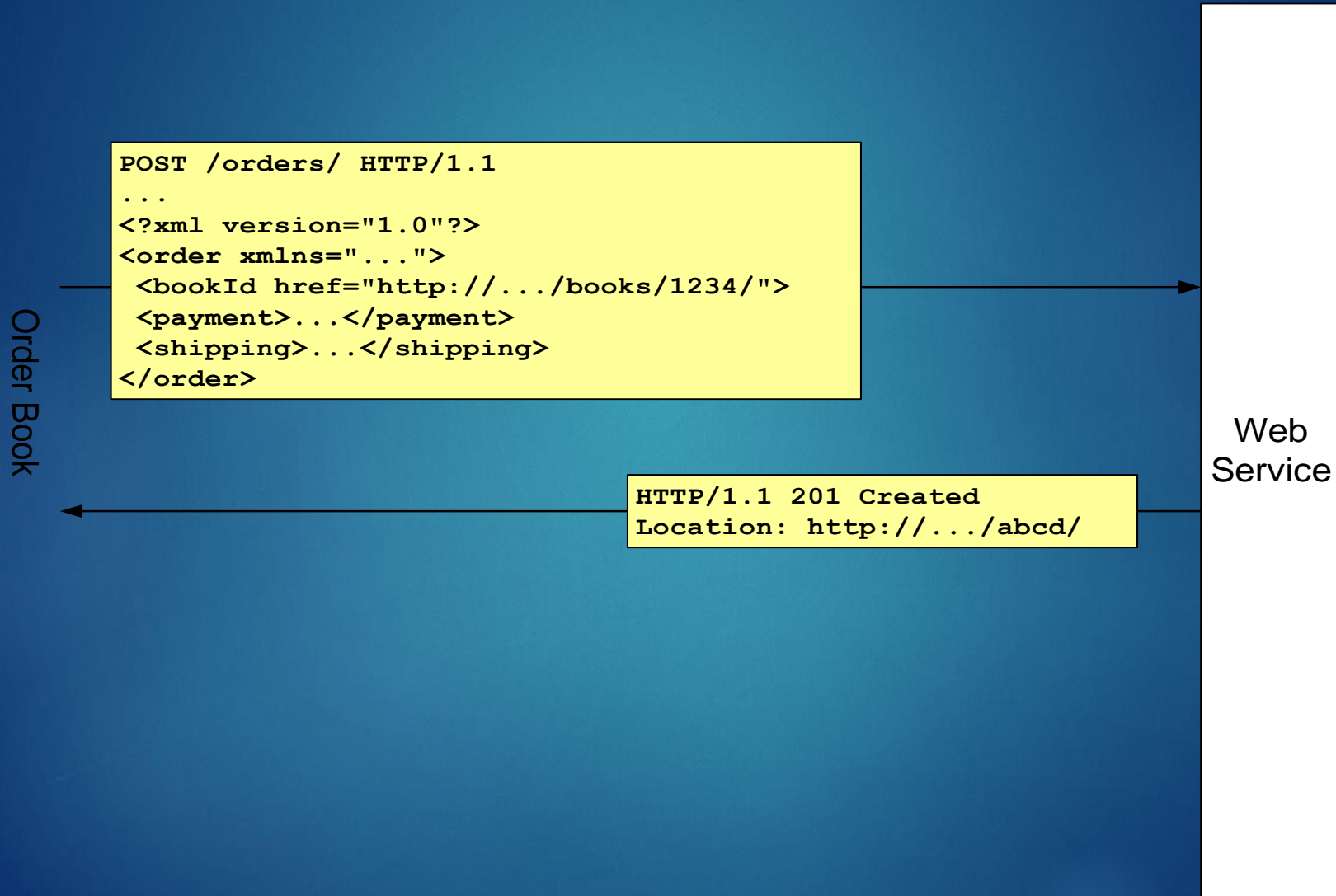
REST & the HTML Web



REST & the XML Web



REST & the XML Web



No. SOAP		REST
1)	SOAP is a protocol .	REST is an architectural style .
2)	SOAP stands for Simple Object Access Protocol .	REST stands for REpresentational State Transfer .
3)	SOAP can't use REST because it is a protocol.	REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP.
4)	SOAP uses services interfaces to expose the business logic .	REST uses URI to expose business logic .
5)	JAX-WS is the java API for SOAP web services.	JAX-RS is the java API for RESTful web services.
6)	SOAP defines standards to be strictly followed.	REST does not define too much standards like SOAP.
7)	SOAP requires more bandwidth and resource than REST.	REST requires less bandwidth and resource than SOAP.
8)	SOAP defines its own security .	RESTful web services inherits security measures from the underlying transport.
9)	SOAP permits XML data format only.	REST permits different data format such as Plain text, HTML, XML, JSON etc.
10)	SOAP is less preferred than REST.	REST more preferred than SOAP.

Web Service Description Language (WSDL)

Used for describing the functionality offered by a web service.

Contained within WSDL are essential objects to support message transfer, including these:

- ▶ The **service object**, a container where the service resides.
- ▶ The **port or endpoint**, which is the unique address of the service.
- ▶ The **binding**, which is the description of the interface (e.g. RPC) and the transport (e.g. SOAP).

WSDL

- ▶ The **portType**, or interface that defines the capabilities of the Web service, and what operations are to be performed, as well as the messages that must be sent to support the operation.
- ▶ The **operation** that is to be performed on the message.
- ▶ The **message** content, which is the data and metadata that the service operation is performed on.
- ▶ The **types** used to describe the data

The main structure of a WSDL document looks like this:

```
<definitions>

<types>
  data type definitions.....
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations.....
</portType>

<binding>
  protocol and data format specification....
</binding>

</definitions>
```

portType defines the capabilities of the Web service, and what operations are to be performed, as well as the messages that must be sent to support the operation.

WSDL Example

- ▶ Assuming the service provides a single publicly available function, called *sayHello*. This function expects a single string parameter and returns a single string greeting. For example if you pass the parameter *world* then service function *sayHello* returns the greeting, "Hello, world!".

```
<definitions name="HelloService"
  targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
  </message>
  <message name="SayHelloResponse">
    <part name="greeting" type="xsd:string"/>
  </message>

  <portType name="Hello_PortType">
    <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
    </operation>
  </portType>
```

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="sayHello">
    <soap:operation soapAction="sayHello"/>
    <input>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:examples:helloservice"
        use="encoded"/>
    </input>
    <output>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:examples:helloservice"
        use="encoded"/>
    </output>
  </operation>
</binding>

<service name="Hello_Service">
  <documentation>WSDL File for HelloService</documentation>
  <port binding="tns:Hello_Binding" name="Hello_Port">
    <soap:address
      location="http://www.examples.com/SayHello/">
    </port>
  </service>
</definitions>
```

Analysis of the Example

- ▶ **Definition :** HelloService
- ▶ **Type :** Using built-in data types and they are defined in XMLSchema.
- ▶ **Message :**
 - ▶ sayHelloRequest : firstName parameter
 - ▶ sayHelloresponse: greeting return value
- ▶ **Port Type:** sayHello **operation** that consists of a request and response service.
- ▶ **Binding:** Direction to use the SOAP HTTP transport protocol.
- ▶ **Service:** Service available at <http://www.examples.com/SayHello/>.
- ▶ **Port:** Associates the binding with the URI <http://www.examples.com/SayHello/> where the running service can be accessed.