Introduction

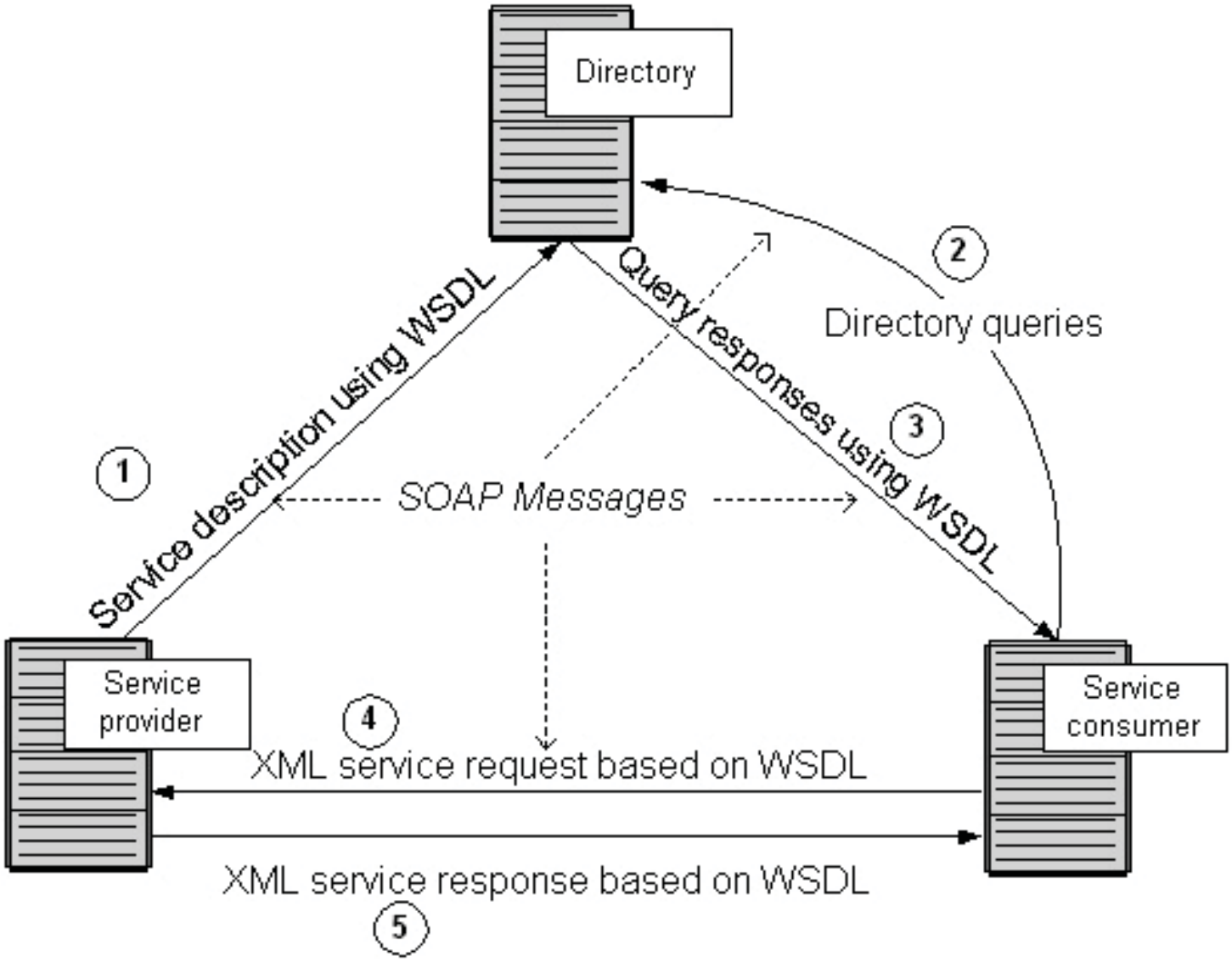
- Web Services and Services are different things.
  - Services provide some functionality
  - Web Services are used to connect Services
- Web Services became wrongly equated with some of the technologies used to implement then (WSDL / SOAP / XML / …)
- Two different approaches:
  - WS-* Standards
  - REST style

WS-*

- WS-* basic standards
  - UDDI (where a service is)
  - WSDL (what a service does)
  - SOAP (communicate with the service)

- More standards:
  - WS-PolicyAssertions
  - WS-Security
  - WS-Trust
  - WS-SecureConversation
  - …

WS-*


Directory
Service description using WSDL
Query responses using WSDL
2
Directory queries
3
1
SOAP Messages
Service provider
Service consumer
4
XML service request based on WSDL
XML service response based on WSDL
5

UDDI

- Universal Description, Discovery and Integration
- Provides an infrastructure for a Web Services-based software environment for both publicly available services and services only exposed internally within an organization
- UDDI registry is intended to as a search engine of Web Services described using WSDL .

UDDI – Information Types

- **White pages:**
  - Company contact information and description
  - Allows others to discover your web service based upon your business identification.
- **Yellow pages:**
  - Define categories for Web Services offered
  - Allows others to discover your web service based upon its categorization
- **Green pages:**
  - technical information that describes the behaviors and supported functions of a web service hosted by your business.
  - Includes pointers to the grouping information of web services and where the web services are located.

WSDL

- Stands for Web Services Description Language
- Document written in XML.
- Specifies the location of the service and the operations (or methods) the service exposes.

WSDL

- WDSL Document Root

```
<definitions name ='weatherservice‘
   xmlns='http://schemas.xmlsoap.org/wsdl/'>

   <service name='WeatherService' >
   ……
   </service>

</definitions>
```

WSDL - Abstract Definitions

- **Types:** Machine- and language-independent type definitions.
  - `xsd:string`
  - `xsd:boolean`

- **Messages:** Contains function parameters (inputs separate from outputs) or document descriptions.

```
<message name='Weather.GetTemperature'>
  <part name='zipcode' type='xsd:string'/>
  <part name='celsius' type='xsd:boolean'/>
</message>
<message name='Weather.GetTemperatureResponse'>
  <part name='Result' type='xsd:float'/>
</message>
```

WSDL - Abstract Definitions

- **Operations:** describe methods signatures (operation name, input parameters, output parameters).

```
<operation name='GetTemperature' parameterOrder='zipcode
  celsius'>
  <input message='wsdlns:Weather.GetTemperature' />
  <output message='wsdlns:Weather.GetTemperatureResponse' />
</operation>
```

- **PortTypes:** Collection of all operations exposed by the service

```
<portType name='WeatherSoapPort'>
  <operation name='GetTemperature' parameterOrder='zipcode
  celsius'>
            <input message='wsdlns:Weather.GetTemperature' />
            <output
  message='wsdlns:Weather.GetTemperatureResponse' />
  </operation>
  <!-- other operations would go here -->
</portType>
```

WSDL - Concrete Descriptions

- **Bindings:** Specifies binding(s) of each operation in the PortTypes section.

```
<binding name='WeatherSoapBinding' type='wsdlns:WeatherSoapPort' >
   <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />

   <operation name='GetTemperature' >

      <soap:operation
        soapAction='http://tempuri.org/action/Weather.GetTemperature' />
      <input>
        <soap:body use='encoded'
        namespace='http://tempuri.org/message/'encodingStyle='http://schemas.xml
        soap.org/soap/encoding/' />
      </input>
      <output>
         <soap:body use='encoded' namespace='http://tempuri.org/message/‘
         encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
         </output>

   </operation>

</binding>
```

WSDL - Concrete Descriptions

- **Services:** Specifies port address(es) of each binding.

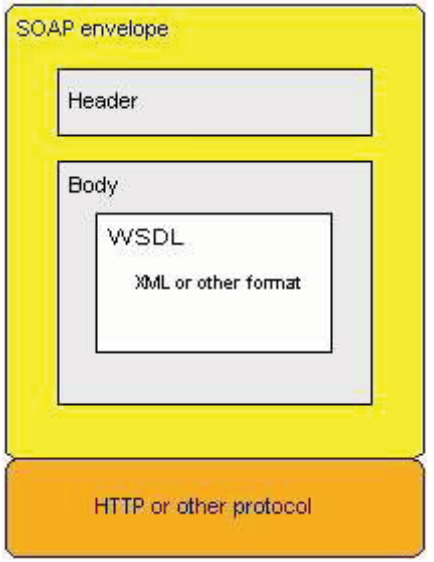
```
<port name='WeatherSoapPort‘ binding='wsdlns:WeatherSoapBinding' >
   <soap:address location='http://localhost/weatherservice.asp' />
</port>
```

SOAP

- Once stood for Simple Object Access Protocol. From version 1.2 lost its meaning.
- SOAP provides the envelope for sending Web Services messages over the Internet.
- SOAP commonly uses HTTP, but other protocols such as Simple Mail Transfer Protocol (SMTP) may by used.
- SOAP can be used to exchange complete documents or to call a remote procedure.

SOAP - Envelope

- The SOAP envelope can contain:
  - An optional header providing information on authentication, encoding of data, or how a recipient of a SOAP message should process the message.
  - The body that contains the message. These messages can be defined using the WSDL specification.
  - Optional Fault element that provides information about errors that occurred while processing the message



Soap Example

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml;
    charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/20
    01/12/soap-envelope" … >

<soap:Body
    xmlns:m="http://www.example.org/
    stock">
    <m:GetStockPrice>
            <m:StockName>IBM</m:Sto
    ckName>
    </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

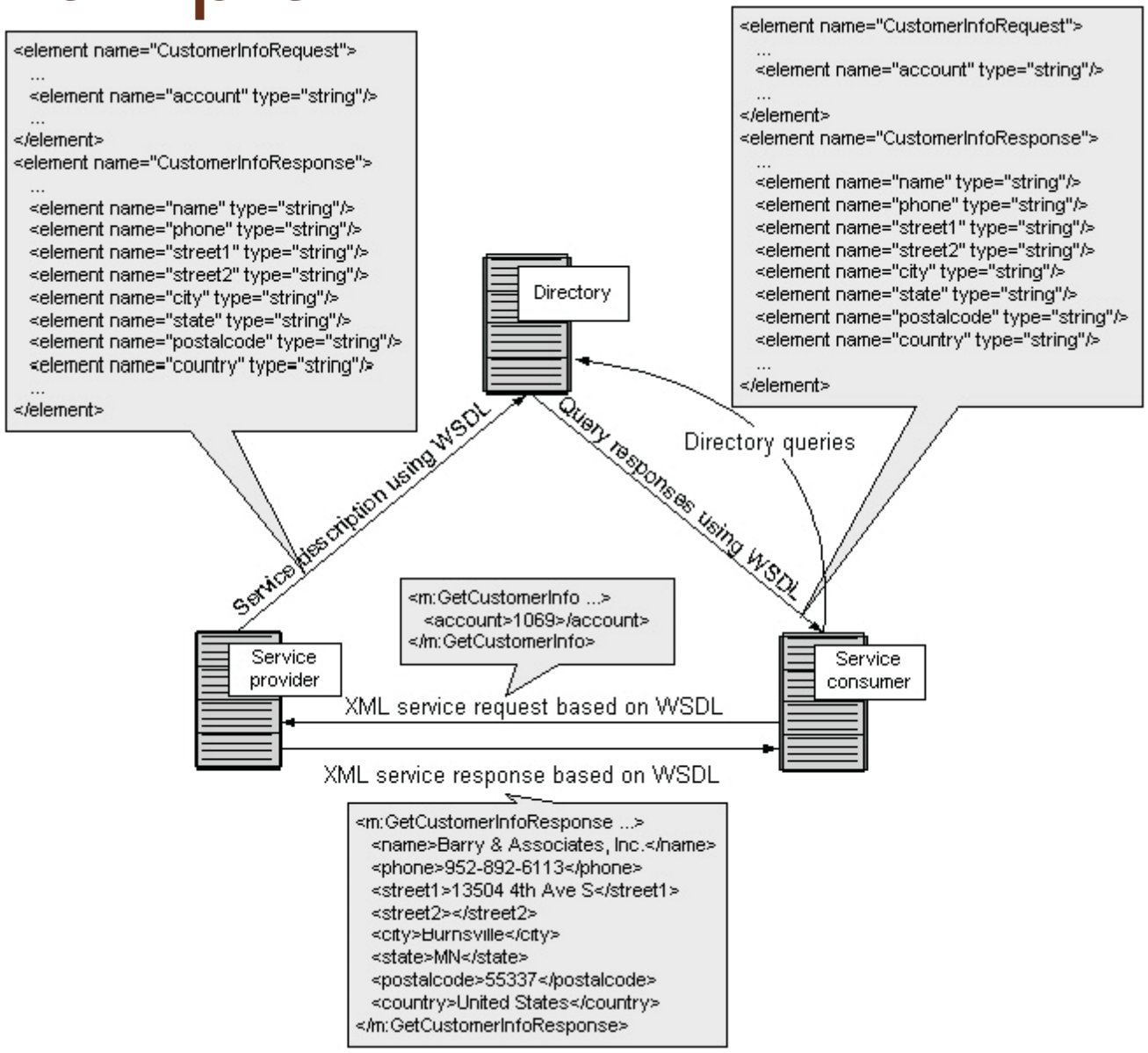
```
HTTP/1.1 200 OK
Content-Type: application/soap+xml;
    charset=utf-8
Content-Length: nnn

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

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

</soap:Envelope>
```

Example


<element name="CustomerInfoRequest">
...
<element name="account" type="string"/>
...
</element>
<element name="CustomerInfoResponse">
...
<element name="name" type="string"/>
<element name="phone" type="string"/>
<element name="street1" type="string"/>
<element name="street2" type="string"/>
<element name="city" type="string"/>
<element name="state" type="string"/>
<element name="postalcode" type="string"/>
<element name="country" type="string"/>
...
</element>
Directory
<element name="CustomerInfoRequest">
...
<element name="account" type="string"/>
...
</element>
<element name="CustomerInfoResponse">
...
<element name="name" type="string"/>
<element name="phone" type="string"/>
<element name="street1" type="string"/>
<element name="street2" type="string"/>
<element name="city" type="string"/>
<element name="state" type="string"/>
<element name="postalcode" type="string"/>
<element name="country" type="string"/>
...
</element>
Service description using WSDL
Query responses using WSDL
Directory queries
<m:GetCustomerInfo ...>
<account>1069</account>
</m:GetCustomerInfo>
Service provider
Service consumer
XML service request based on WSDL
XML service response based on WSDL
<m:GetCustomerInfoResponse ...>
<name>Barry & Associates, Inc.</name>
<phone>952-892-6113</phone>
<street1>13504 4th Ave S</street1>
<street2></street2>
<city>Burnsville</city>
<state>MN</state>
<postalcode>55337</postalcode>
<country>United States</country>
</m:GetCustomerInfoResponse>

The REST Approach

- REpresentational State Transfer
- First described in 2000 by Roy Fielding
- Is a style of software architecture
- Is NOT a set of standards.
- Applications or architectures are sometimes referred to as *RESTful* or *REST-style* applications or architectures.

REST*full* Characteristics

- Stateless client-server architecture
- Web services are viewed as **resources** and can be identified by their URLs.
- Every resource is uniquely addressable using a uniform and minimal set of **commands** (typically using HTTP commands of GET, POST, PUT, or DELETE over the Internet)
- Each resource return a **representation**

REST*ful* Characteristics

- While REST is not a standard, it does use standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/etc (Resource Representations)
  - text/xml, text/html, image/gif, image/jpeg, etc (MIME Types)

Resource Oriented Architecture

- Appeared as a response to the SOA standard from REST supporters
- Guidelines to implement a REST*ful* style architecture
  - Each resource knows how to Represent, make a transition from states and self-destruct (Resource definition)
  - Each Resource is unambiguously accessed by a unique URI (Addressability)
  - Requests to Resources have all necessary information (Statelessness)
  - Resources should link to other dependent resource in their representation (Connectedness)

REST Example

- The Boeing defines access to a resource (a 747):

  http://www.boeing.com/aircraft/747

- Using a GET command this resource returns a representation of its state:

  Boing747.html

- This state can represent access to other dependent resource (ex: links)

REST Example

```
POST /register/user/09723 HTTP/1.0
Accept: */*
Connection: close
Content-Type: text/xml
Content-Length: 618
Pragma: no-cache

<?xml version="1.0" encoding="UTF-
    8"?>
<request>
    <name>xtpo</name>
    <address>here</address>
    ...
</request>
```

```
200 OK
Content-Type: text/xml
Connection: close

<?xml version="1.0"
    encoding="UTF-8"?>

<response>
    <success>true</success>

</request>
```

Google Data API

- Uses REST style APIs
- Resources are representation is based on the Atom 1.0 and RSS 2.0 syndication formats.
- Example of a retrieving Google Calendar events for a user

  GET http://www.google.com/calendar/feeds/userID/private-magicCookie/full

WS-* and Rest

- Advantages of WS-*
  - Easy to consume - sometimes
  - Rigid - type checking, adheres to a contract
  - Development tools
- Advantages of REST style
  - Lightweight
  - Results are less verbose, hence more human readable
  - Easy to build - no toolkits required

WS-* and Rest

- A WS-* based design may be appropriate when:
  - A formal contract is needed for a service (WSDL).
  - Complex need to be handled in a standardized way.
  - Requirements for asynchronous service invocation are present.

WS-* and Rest

- A RESTfull design may be appropriate when:
  - Completely stateless web services are needed.
  - Limited bandwidth between service consumer and provider. Mobile Device communication is an excellent example.
  - Both service consumer and provider have mutual understanding of the context and content being passed.
  - Front-end technologies like as AJAX are being used.