# 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-*
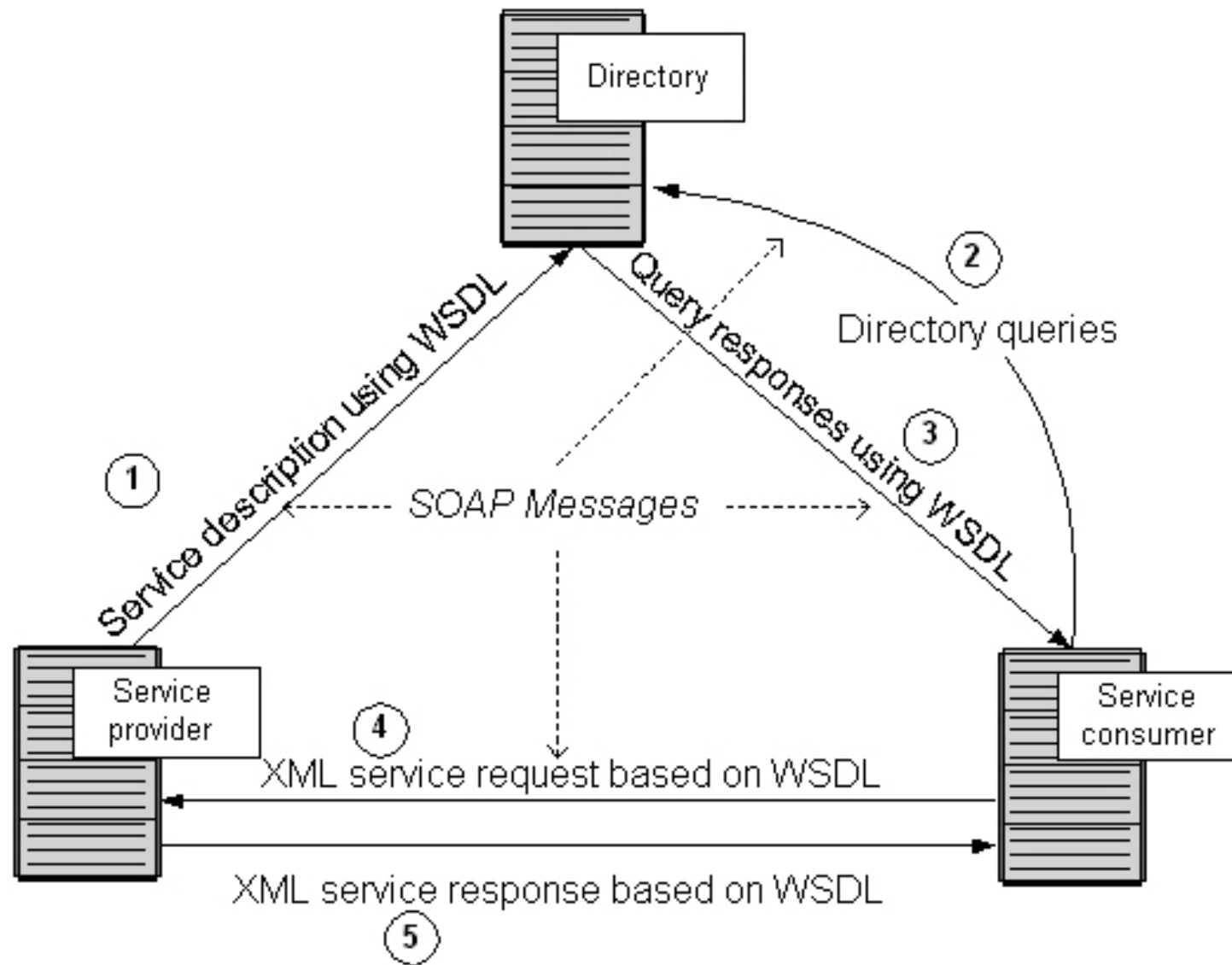
UDDI

- 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 <u>search engine of Web Services</u> 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 **W**eb **S**ervices **D**escription **L**anguage

- Document written in XML.

- Specifies the location of the service and the operations (or methods) the service exposes.

WSDL

- WDSL Documet 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.

```xml
<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>
```
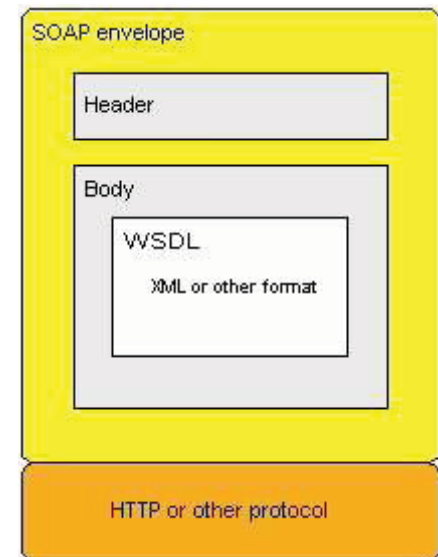
SOAP

- 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>
```
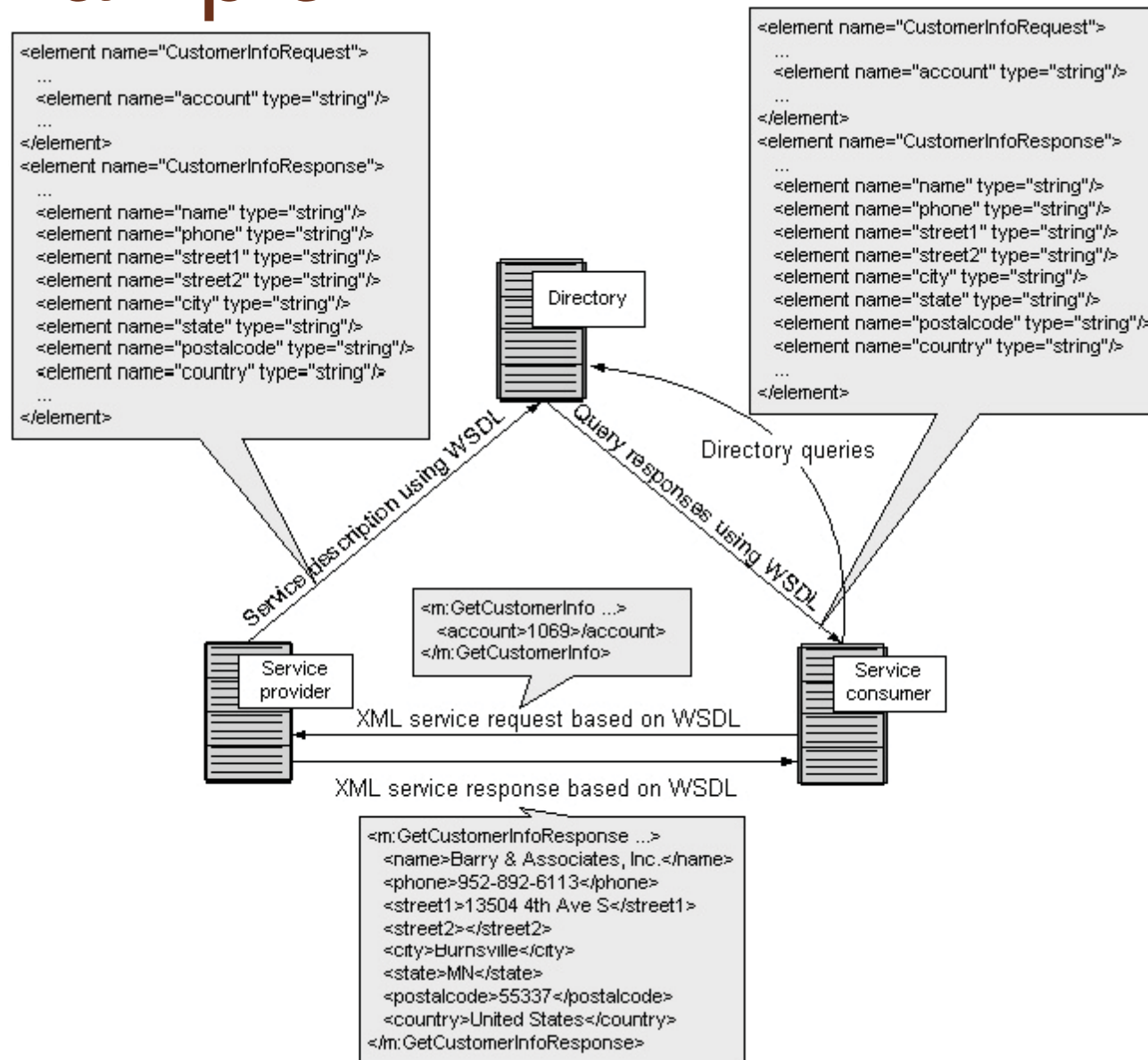
Example

Tools: Apache AXIS (v2)

- Apache Axis is an Open Source SOAP server and client(from ws.apache.org)

- Axis helps to speed up development when using on WS-* usage by generation code for both producer and consumer sides.

- One good functionality is that Axis can quickly expose a class as web service

- Resources:
  - Axis 1.x: http://ws.apache.org/axis/
  - Axis 2: http://ws.apache.org/axis2/
  - Actually look at the all of the content in http://ws.apache.org

http://ws.apache.org

Apache <Web Services /> Project

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

- <u>Guidelines</u> 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.

Exercise

- WS-*
  - Use UDDI to find the WSDL for one service.
  - Create a simple java client to invoke the service

- REST
  - Create an account in the **del.icio.us** service
  - Create a client to read REST APIs from del.icio.us

  API details: http://www.peej.co.uk/articles/restfully-delicious.html