CSCC09F
Programming on the Web

Web-services: RESTful and SOAP

RESTful Web Services
(REpresentational State Transfer)

- The Web consists of resources accessed using URL’s.
- When a resource is requested, a representation of the resource is returned (may be text, (X)HTML, XML, JSON, or other MIME-type object).
- Receipt of this representation is associated with a client application state. If a new request is made based on that representation (e.g. follow a link), the client application will transition into another state.
- Thus, the client application changes state with the receipt (transfer) of each resource representation

RESTful Web Services
(REpresentational State Transfer)

- From Roy Fielding's thesis:

  "Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of Web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

HTTP Methods and CRUD

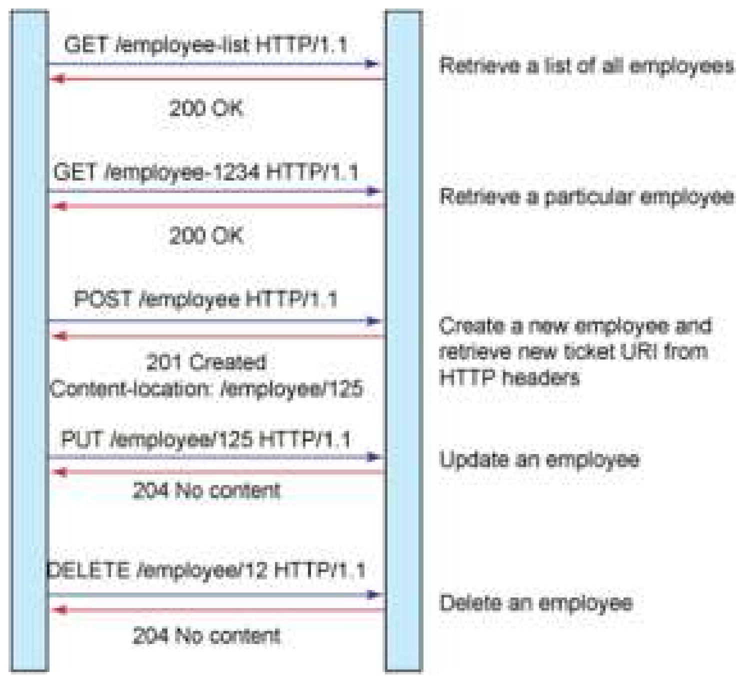
- REST developers use HTTP methods explicitly and in a way that's consistent with the HTTP protocol definition.
- This basic REST design principle establishes a 1:1 mapping between create, read, update, and delete (CRUD) operations and HTTP methods:
  - Create a resource on the server: **POST**
  - Retrieve a resource: **GET**
  - Update the state of a resource (or create it): **PUT**
  - Delete a resource: **DELETE**

REST HTTP Interface

- RESTful services are limited to HTTP with the four well-defined methods (SOAP is not limited to HTTP):
  - GET: retrieve the state of a resource
  - POST: create a resource
  - PUT: update the state of a resource
  - DELETE: delete/remove a resource
- The most common messaging format for RESTful Web services are XML and JSON, but without the standardized formal structure of SOAP and WSDL Web services
- Simple and lightweight, like the document Web; easy to deploy, without need for complicated tools

Web Apps using RESTful APIs

- Does this look familiar?
- Backbone uses RESTful services (API) to persist model data on the server
- Node.js implements RESTful services (API) in its router (app.js) and route handler (eatz.js)


GET /employee-list HTTP/1.1
200 OK
Retrieve a list of all employees
GET /employee-1234 HTTP/1.1
200 OK
Retrieve a particular employee
POST /employee HTTP/1.1
201 Created
Content-location: /employee/125
Create a new employee and retrieve new ticket URI from HTTP headers
PUT /employee/125 HTTP/1.1
204 No content
Update an employee
DELETE /employee/12 HTTP/1.1
204 No content
Delete an employee

Query-String Extensibility

- Extensibility Principle: a Web-service provider (API implementor) should <u>ignore</u> any query parameters it does not understand
  - follows the same approach as the document Web: browsers ignore tags they don't understand (just display element content) – in particular they <u>don't</u> "<u>break</u>" like a program might (with e.g. a traceback)
- If Web-service app needs to interact with other Web services as a client, it should <u>pass</u> all ignored parameters to those other services (<u>transparency</u>)
- This practice supports <u>extensibility</u> of Web services by allowing new functionality to be added without breaking existing services

RESTful API's are Testable

- RESTful API's are well suited to automated testing

```
curl -X GET http://mathlab.utsc.utoronto.ca:nnnnn/auth

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 51
Set-Cookie:
eatz.sess=s%3AMRiBgzvmdOlLwAOpn4wkR3tT.JFg7La48hd5L8IRFVenTGnE
BIMtNUVA%2FAX1Y34CorKw; Path=/; Expires=Wed, 19 Nov 2014
13:50:58 GMT; HttpOnly
Date: Wed, 19 Nov 2014 13:48:58 GMT
Connection: keep-alive

{
  "userid": "",
  "username": ""
}
```

RESTful API’s are Testable

- RESTful API’s are well suited to automated testing

```
curl -X PUT http://mathlab.utsc.utoronto.ca:nnnnn/auth
-i -H "Content-Type: application/json"
-d '{"username":"a","password":"a","login":"1"}'
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 76
Set-Cookie:
eatz.sess=s%3AaFjJy4mouyObydcbvssvKqLE.s6s5nYg9MyD4S9p58Hu7%2FCFNspl4XgN
hGMJOkxD%2BJNA; Path=/; Expires=Wed, 19 Nov 2014 13:53:28 GMT; HttpOnly
Date: Wed, 19 Nov 2014 13:51:28 GMT
Connection: keep-alive

{
  "userid": "54635fe6a1342684065f6959",
  "username": "a"
}
```

When is REST Appropriate?

- KISS, agile, lightweight app-development environments
- Overhead of service requests and responses must be minimized, e.g. for mobile or other low- bandwidth or low-processing-power clients
- Ease of integration with existing applications is important
  - a RESTful service can easily be integrated through use of JavaScript and Ajax to transform the output of a service for direct inclusion in a Web application
- Service provider and consumer must have mutual understanding/agreement of interface and data schemas
  - since the service method APIs and parameter data types are not defined using a standard framework like WSDL

Who Uses REST?

- Twitter, Facebook, and all of Yahoo’s Web services use REST, including Flickr.
- Both eBay and Amazon provide Web services using both REST and SOAP.
- Web 2.0 applications rely on Web API’s like those of the various Google tools, which are usually based on REST rather than SOAP.
- REST-style services are usually the building blocks of “mashups”, which combine existing services/data into new applications.

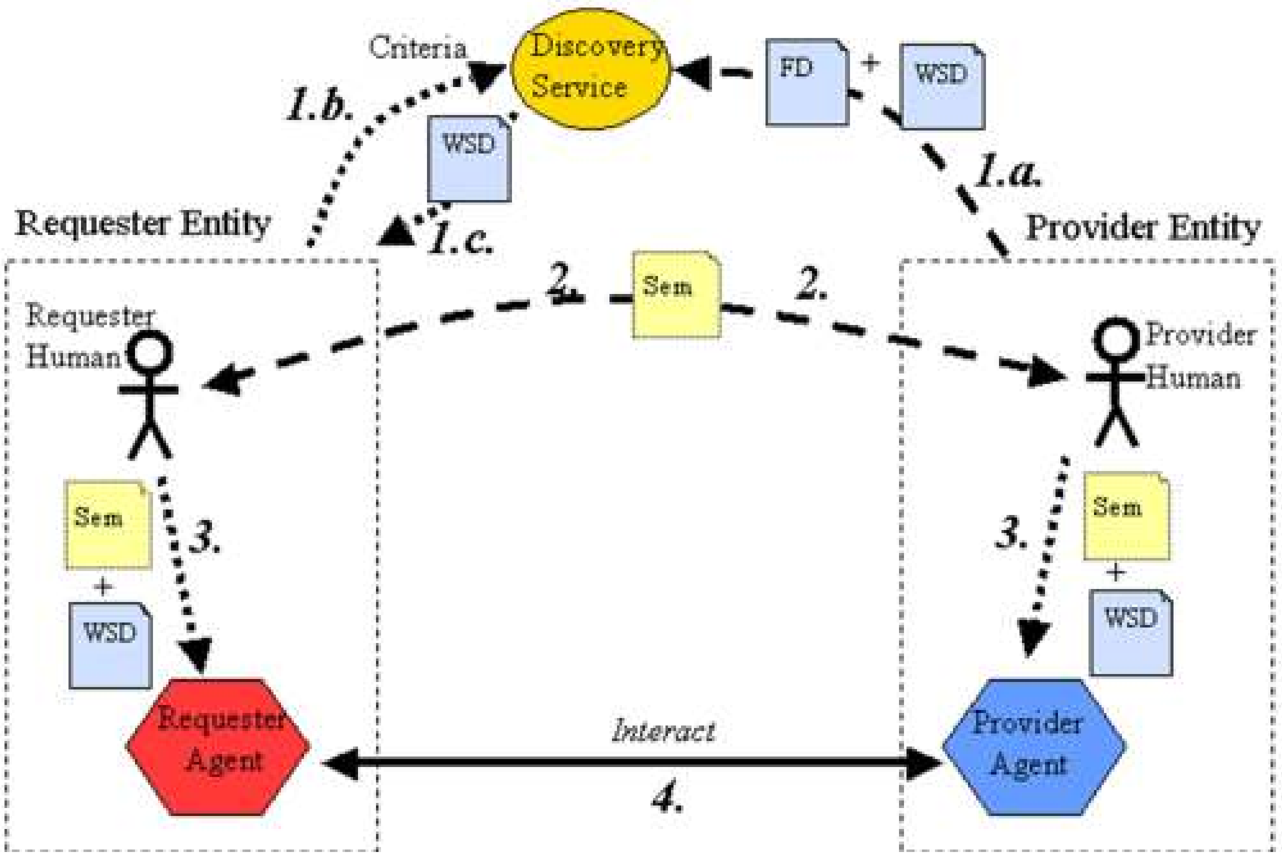
Who Uses RESTful API’s?

- Google search REST API: https://developers.google.com/custom-search/json-api/v1/overview
- Amazon’s S3 REST API (REST for other Amazon services also): http://docs.aws.amazon.com/AmazonS3/latest/API/APIRest.html
- Twitter has an extensive REST API: https://dev.twitter.com/rest/public
- Flickr’s REST API: http://www.flickr.com/services/api/
- Yahoo’s Social REST API’s: https://developer.yahoo.com/social/rest_api_guide/web-services-intro.html
- Oanda RESTful FX-trading API: http://developer.oanda.com/rest-live/introduction/
- Geocoder.ca REST API: http://www.geocoder.ca/

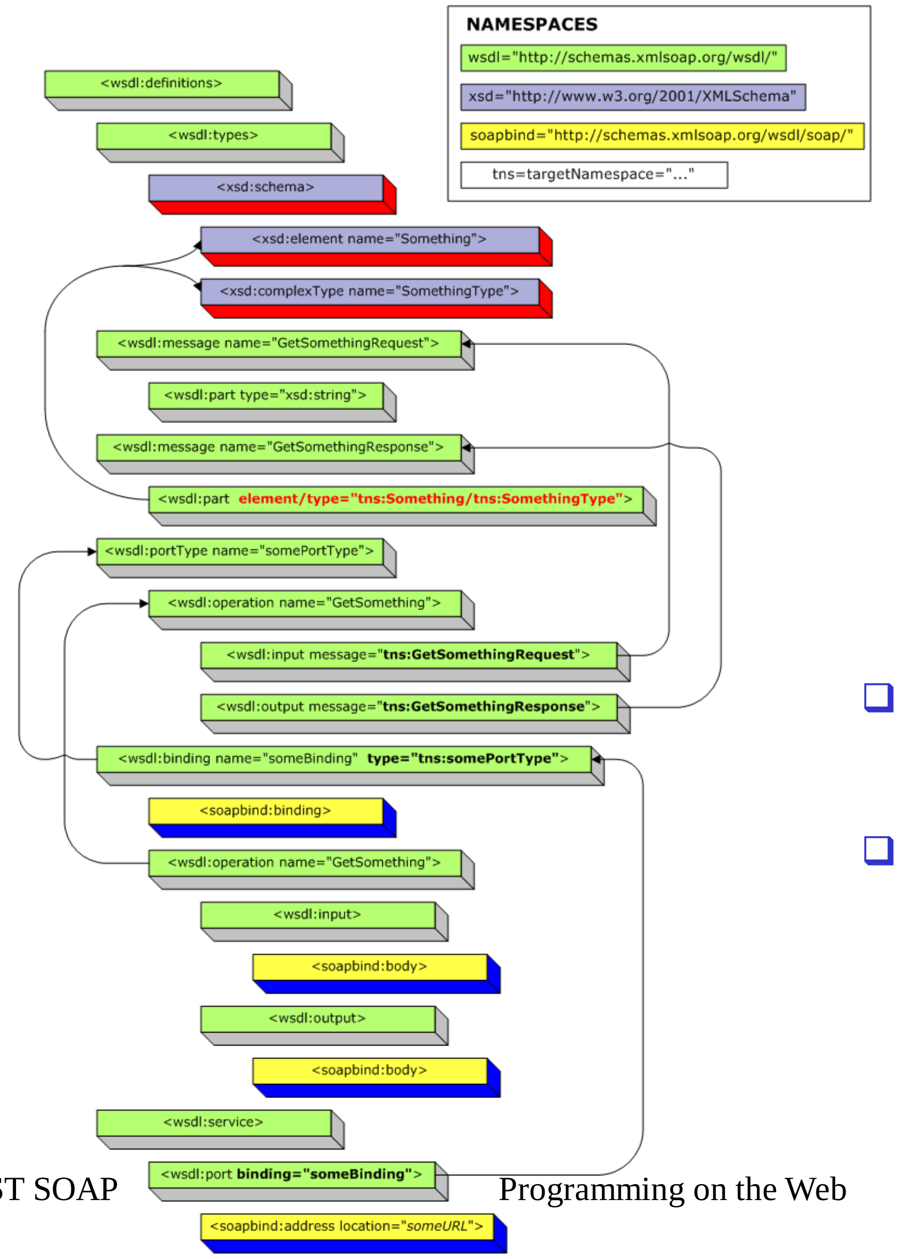
What Else is there?

- The first generation of commercial Web-services utilized a framework referred to as SOAP (heavily promoted by MS and IBM)
- SOAP was designed to operate in environments with (very) loosely coupled clients and servers
  - clients should be able to discover Web-services dynamically
  - services come with a machine-readable data-typed API that clients use to learn how to invoke the service (method calls, parameter number and type)
- By contrast, REST services and clients are more tightly coupled – e.g. they know a priori what kind of data to pass in requests and what response data will look like
- Hmm … maybe SOAP sounds better ?

SOAP-Style Web Services



71 REST SOAP Programming on the Web 14



```
NAMESPACES
wsdl="http://schemas.xmlsoap.org/wsdl/"
xsd="http://www.w3.org/2001/XMLSchema"
soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
tns=targetNamespace="..."

<wsdl:definitions>
  <wsdl:types>
    <xsd:schema>
      <xsd:element name="Something">
      <xsd:complexType name="SomethingType">
  <wsdl:message name="GetSomethingRequest">
    <wsdl:part type="xsd:string">
  <wsdl:message name="GetSomethingResponse">
    <wsdl:part element/type="tns:Something/tns:SomethingType">
  <wsdl:portType name="somePortType">
    <wsdl:operation name="GetSomething">
      <wsdl:input message="tns:GetSomethingRequest">
      <wsdl:output message="tns:GetSomethingResponse">
  <wsdl:binding name="someBinding" type="tns:somePortType">
    <soapbind:binding>
    <wsdl:operation name="GetSomething">
      <wsdl:input>
        <soapbind:body>
      <wsdl:output>
        <soapbind:body>
  <wsdl:service>
    <wsdl:port binding="someBinding">
      <soapbind:address location="someURL">
```

WSDL Document

- Web Services Description Language
- Collection of one or more Web-service definitions

WSDL Document

- Collection of one or more Web-service definitions

```
<definitions>
   <types> ….. </types>    // types defined using XSchema
   <message> …. </message> // messages transmitted
   <portType>          // sets of abstract operations
        <operation> …. </operation>   // abstract operation
        …..
   </portType>
   <binding>           // concrete representation of portType
        <operation> …. </operation>   // encoding of operation
        ….
   </binding>
   <service>           // set of service portType implementations
        <port> … </port>       // specific URL for service
        …..                    //     portType implementation
   </service>
</definitions>
```

WSDL types

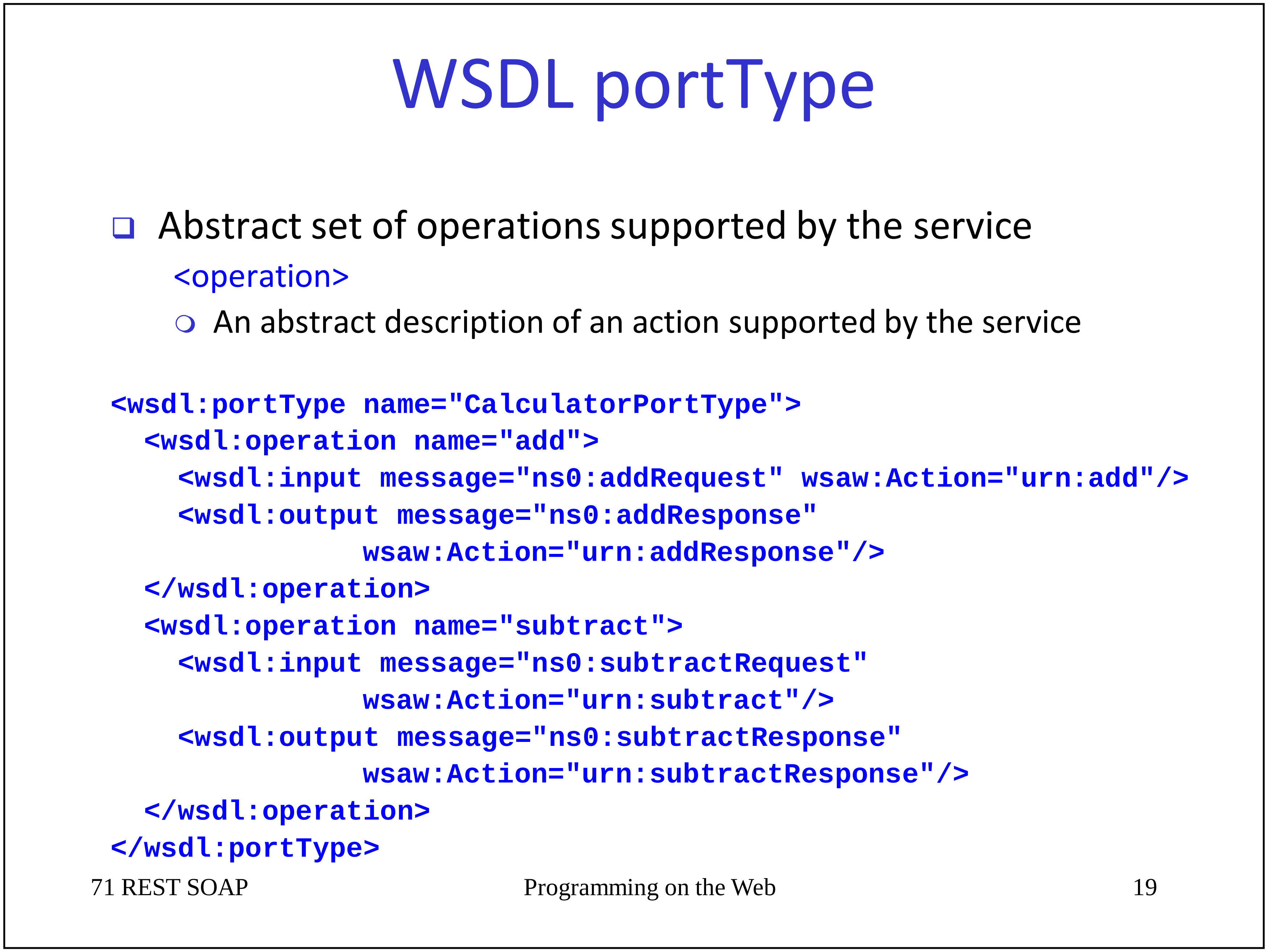
- Provide XSchema data-type definitions used to describe the messages exchanged (types defined in PL-independent way)

```
<wsdl:types>
  <xs:schema xmlns:ns0="http://ws.apache.org/axis2"
      attributeFormDefault="qualified"
      elementFormDefault="qualified"
      targetNamespace="http://ws.apache.org/axis2">
  <xs:element name="addReq">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="param0" type="xs:int"/>
        <xs:element minOccurs="0" name="param1" type="xs:int"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  . . .
 </xs:schema>
</wsdl:types>
```

WSDL message

- Abstract and typed definitions of the data being communicated

```
<wsdl:message name="addRequest">
  <wsdl:part name="parameters"
                                  element="ns0:addReq"/>
</wsdl:message>
<wsdl:message name="addResponse">
  <wsdl:part name="parameters"
                                  element="ns0:addResp"/>
</wsdl:message>
```



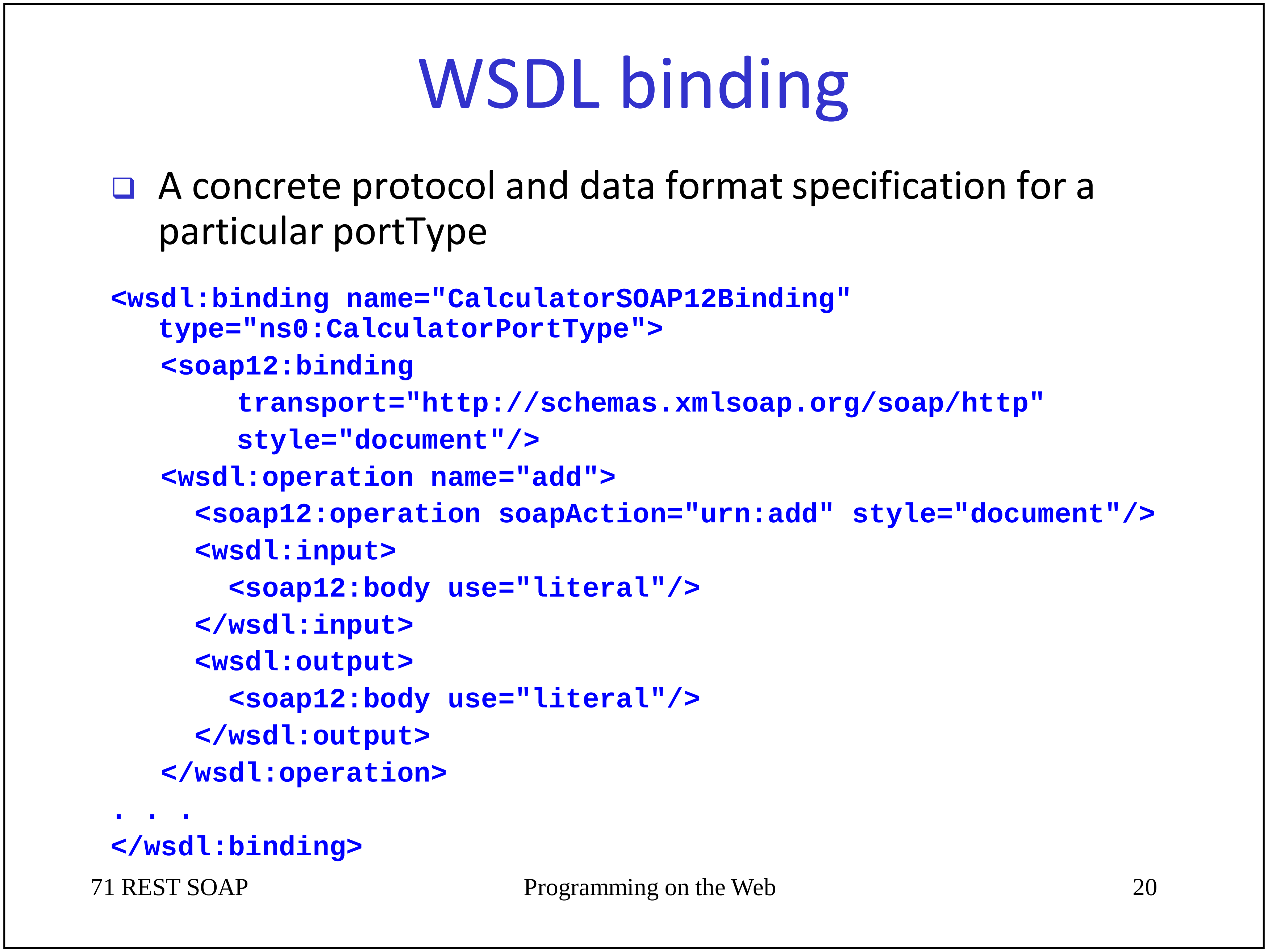
WSDL portType

- Abstract set of operations supported by the service

  <operation>

  - An abstract description of an action supported by the service

```
<wsdl:portType name="CalculatorPortType">
  <wsdl:operation name="add">
    <wsdl:input message="ns0:addRequest" wsaw:Action="urn:add"/>
    <wsdl:output message="ns0:addResponse"
                 wsaw:Action="urn:addResponse"/>
  </wsdl:operation>
  <wsdl:operation name="subtract">
    <wsdl:input message="ns0:subtractRequest"
                 wsaw:Action="urn:subtract"/>
    <wsdl:output message="ns0:subtractResponse"
                 wsaw:Action="urn:subtractResponse"/>
  </wsdl:operation>
</wsdl:portType>
```



WSDL binding

- A concrete protocol and data format specification for a particular portType

```
<wsdl:binding name="CalculatorSOAP12Binding"
   type="ns0:CalculatorPortType">
   <soap12:binding
        transport="http://schemas.xmlsoap.org/soap/http"
        style="document"/>
   <wsdl:operation name="add">
     <soap12:operation soapAction="urn:add" style="document"/>
     <wsdl:input>
       <soap12:body use="literal"/>
     </wsdl:input>
     <wsdl:output>
       <soap12:body use="literal"/>
     </wsdl:output>
   </wsdl:operation>
. . .
</wsdl:binding>
```

WSDL service

- A collection of related endpoints
  - port: a single endpoint defined as a combination of a binding and a network address

```
<wsdl:service name="Calculator">
  <wsdl:port name="CalculatorSOAP12port_http"
      binding="ns0:CalculatorSOAP12Binding">
     <soap12:address
location=http://mathlab.utsc.utoronto.ca:28001/axis/
   services/Calculator
     />
  </wsdl:port>
  . . .
</wsdl:service>
```

SOAP Protocol

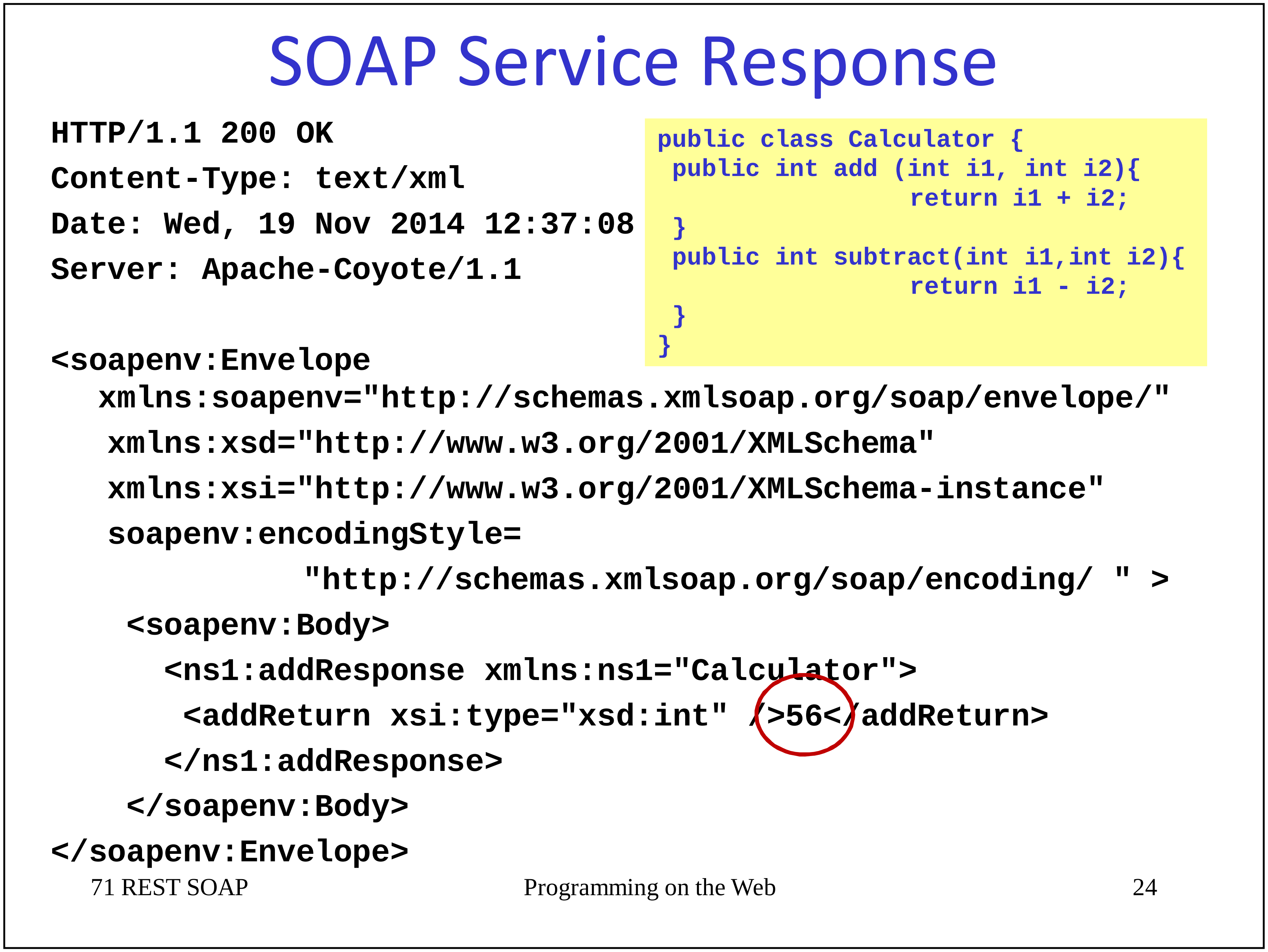
- Sometimes treated as acronym for Service-Oriented Architecture Protocol
- Like WSDL, SOAP is an XML application, and is similarly verbose
- Unlike REST, not restricted to HTTP for message transport – can send SOAP messages over SMTP (email) or other protocols
- Whereas the role of WSDL is to define the service API, the purpose of SOAP is to carry actual service requests and responses in a platform and network-protocol independent way

SOAP Service Request

```
POST /axis/Calculator.jws HTTP/1.0
Host: localhost
Content-Length: 586
Content-Type: text/xml
SOAPAction: "SomeURI"
```

```
public class Calculator {
  public int add (int i1, int i2){
              return i1 + i2;
  }
  public int subtract (int i1, int i2){
              return i1 - i2;
  }
}
```

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
     <calc:add xmlns:calc="Calculator">
       <calc:op1 xsi:type="xsd:int">50</calc:op1>
       <calc:op2 xsi:type="xsd:int">6</calc:op2>
     </calc:add>
   </soapenv:Body>
</soapenv:Envelope>
```



SOAP Service Response

```
HTTP/1.1 200 OK
Content-Type: text/xml
Date: Wed, 19 Nov 2014 12:37:08
Server: Apache-Coyote/1.1

<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    soapenv:encodingStyle=
               "http://schemas.xmlsoap.org/soap/encoding/ " >
     <soapenv:Body>
       <ns1:addResponse xmlns:ns1="Calculator">
        <addReturn xsi:type="xsd:int" />56</addReturn>
       </ns1:addResponse>
     </soapenv:Body>
</soapenv:Envelope>
```

```
public class Calculator {
 public int add (int i1, int i2){
                return i1 + i2;
 }
 public int subtract(int i1,int i2){
                return i1 - i2;
 }
}
```

71 REST SOAP Programming on the Web 24

Tool Support

- SOAP and WSDL would never have gotten off the ground if humans had to hand-craft all that XML!
- On the Web-service provider-side tools can generate a WSDL service-description directly from an interface definition or class implementation, e.g. Java2WSDL
- On the client-side, tools can generate proxies and code skeletons to request the associated service, e.g. WSDL2Java
- Web-service requests can be tool-translated into SOAP format

Py Example

```
import suds
class Client:
    def __init__(self):
        self.client =
                 suds.client.Client("http://mathlab….ca:nnnnn/
                 axis/services/Calculator?wsdl")

    def sum(self, m, n):
        return self.client.service.add(m,n)

    def difference(self, m, n):
        return self.client.service.subtract(m,n)

if(__name__ == "__main__"):
    m = 227;  n = 342
    client = Client()
    print "Sum of: ", m, " and ", n, " is: ", client.sum(m,n)
    print "Difference of: ", m, " and ", n, " is: ",
                                           client.difference(m,n)
```