ECE4574 – Large-Scale SW Development for Engineering Systems
Lecture 9 – Web Services

Creed Jones, PhD

Course Updates

- Project
  - hope Sprint 1 is going well ☺
- Quiz 3 is TODAY
  - 7 PM to 1 AM tomorrow morning
- HW1 is due this FRIDAY
  - 11:59 PM

Today's Objectives

Service-Oriented Architectures
- Service-Oriented Systems
- Web Services
- SOAP and Messaging
- UDDI, WSDL and Metadata
- Security, Transactions and Reliability

XML
- JSON
- XML and JSON compared

RESTful web services
- What is REST?
- Concept
- Examples of RESTful clients

VIRGINIA TECH

BRADLEY DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

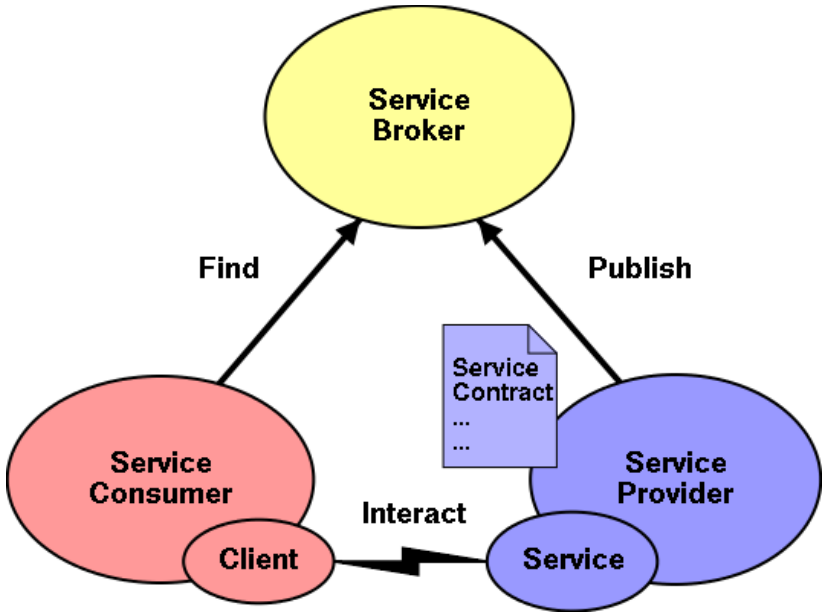
SERVICE-ORIENTED ARCHITECTURES

Service-Oriented Architectures are loosely-coupled arrangements of components that provide services to applications

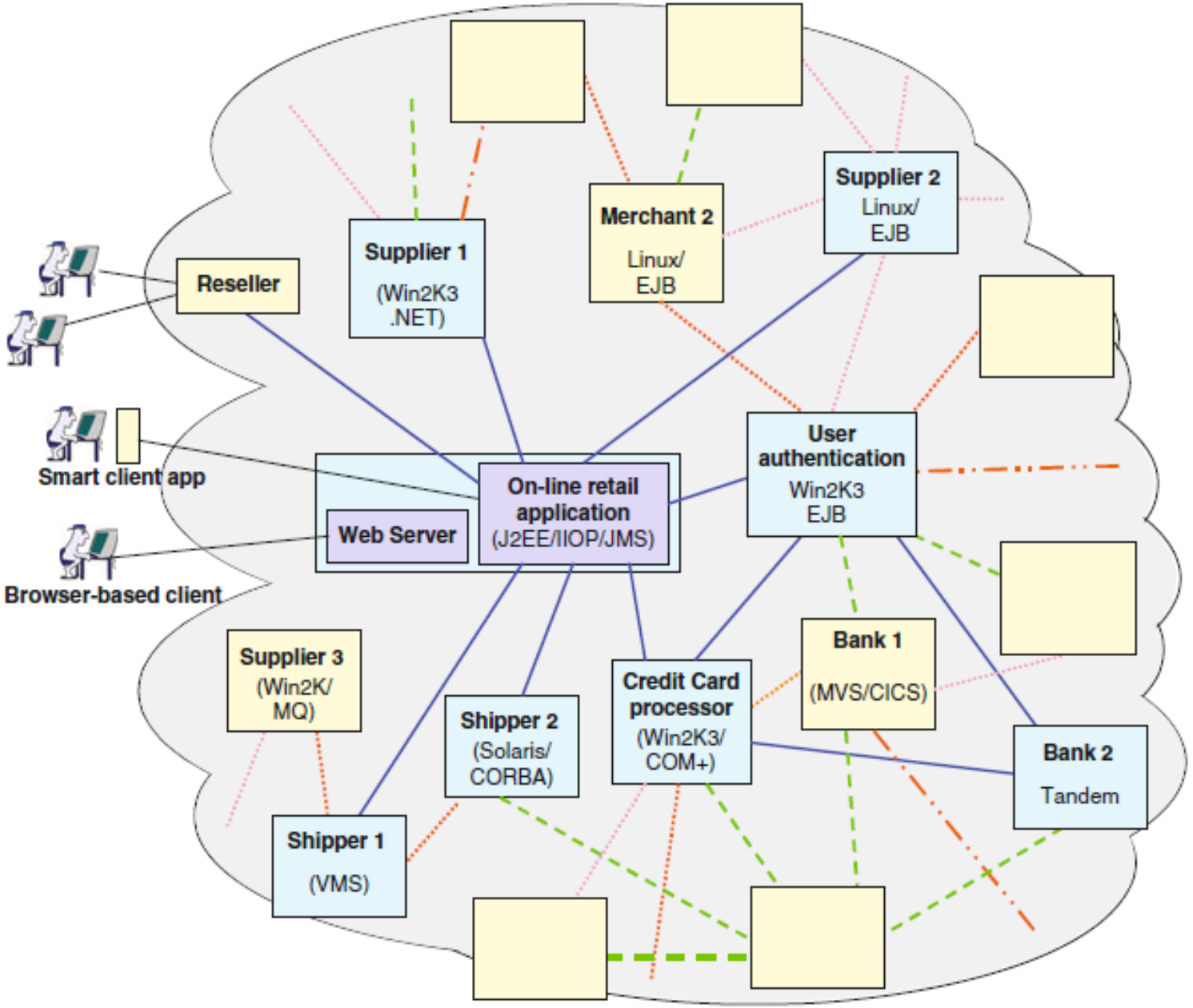
- SOA – *Service-Oriented Architecture*
- Over a network, typically
- A standard protocol governs the interactions
- Several services can be used to accomplish a useful task

- Services are things like:
  - retrieve a transaction record
  - compute an account balance
  - calculate some complex or proprietary metric

SOA is a simple arrangement



- A Service Broker is not always present – depending on the requirements of the protocol



A typical online retail application is composed of a SOA, with various servers (and perhaps protocols!)

There are some basic principles to Service-Oriented Systems

- Boundaries are explicit
  - Services are truly separate applications, not just components of a single process
- Services are autonomous
  - No knowledge of clients
  - Any request satisfying the "rules" will be served
- Share schemas and contracts, not implementations
  - Structure of messages sent and received
  - Any ordering requirements
- Service compatibility is based on policy
  - Format and protocol
  - Encryption, reliability and recovery

Web Services are standards that define simple SOAs that work well over the Internet Protocol

- Simpler than other middleware technologies
- Many vendors and open-source products support them

- Four key tasks:
  - Find suitable services that are available
  - Find out more about a specific service
  - Ask a service to do something
  - Make use of standard services like security

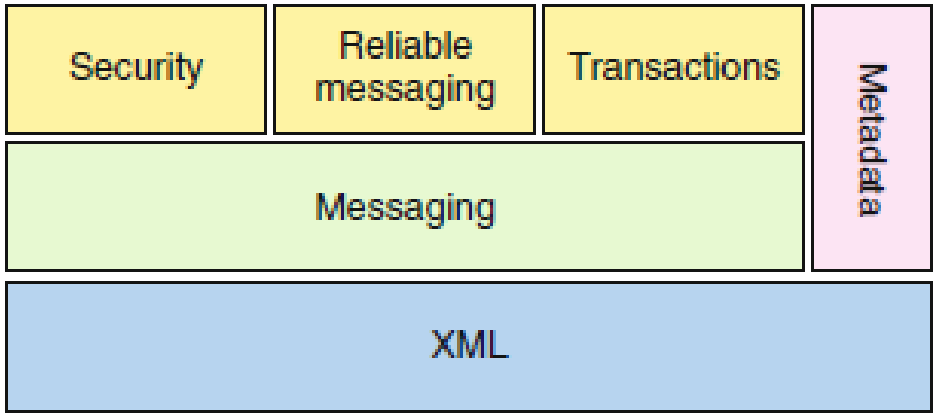
Web Services are standards that define simple SOAs that work well over the Internet Protocol

- Simpler than other middleware technologies
- Many vendors and open-source products support them

- Four key tasks:
  - Find suitable services that are available
    UDDI
  - Find out more about a specific service
    WSDL
  - Ask a service to do something
    SOAP, REST
  - Make use of standard services like security
    WS-*

Many web services are defined on the underlying assumption that XML is used to represent our messages

Fig. 5.2 Overview of Web services standards



- The messages interchanged are in XML
- Some exchanges will use direct metadata access to XML
- Other services are built on the standard definitions of messages

UDDI, WSDL and Metadata

- WSDL is the Web Services Description Language
  - A format to represent what a particular web service does, what it needs and what it returns
- UDDI is the Universal Description, Discovery and Integration directory
  - This is a format to contain and provide many WSDL records
  - Alternatively, WSDL information may be "well-known"

- Metadata – "data about data"
  - WSDL records are just one example of this

A Sample WSDL description

```
<?xml version="1.0"?>
<definitions name="StockQuote"
          targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

 <types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
        xmlns="http://www.w3.org/2000/10/XMLSchema">
   <element name="TradePriceRequest">
    <complexType>
     <all>
      <element name="tickerSymbol" type="string"/>
     </all>
    </complexType>
   </element>
   <element name="TradePrice">
    <complexType>
     <all>
      <element name="price" type="float"/>
     </all>
    </complexType>
   </element>
  </schema>
 </types>

 <message name="GetLastTradePriceInput">
  <part name="body" element="xsd1:TradePriceRequest"/>
 </message>

 <message name="GetLastTradePriceOutput">
  <part name="body" element="xsd1:TradePrice"/>
 </message>

  <portType name="StockQuotePortType">
   <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
   </operation>
  </portType>

  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
   <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="GetLastTradePrice">
    <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
    <input>
     <soap:body use="literal"/>
    </input>
    <output>
     <soap:body use="literal"/>
    </output>
   </operation>
  </binding>

  <service name="StockQuoteService">
   <documentation>My first service</documentation>
   <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
    <soap:address location="http://example.com/stockquote"/>
   </port>
  </service>

</definitions>
```

A Sample WSDL description

```
<?xml version="1.0"?>
<definitions name="StockQuote"
          targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
          xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
        <complexType>
          <all>
            <element name="price" type="float"/>
          </all>
        </complexType>
      </element>
    </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>

  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>

  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
      <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>

</definitions>
```

XML namespace(s)

messages used by this service

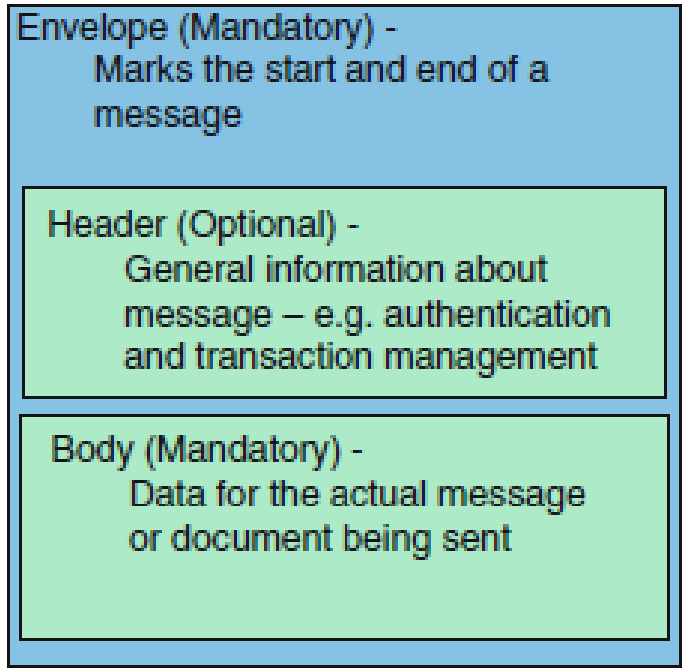
a "port" is a specific service

functions actually exposed via this service

SOAP used to stand for "Simple Object Access Protocol" – but now it's just SOAP

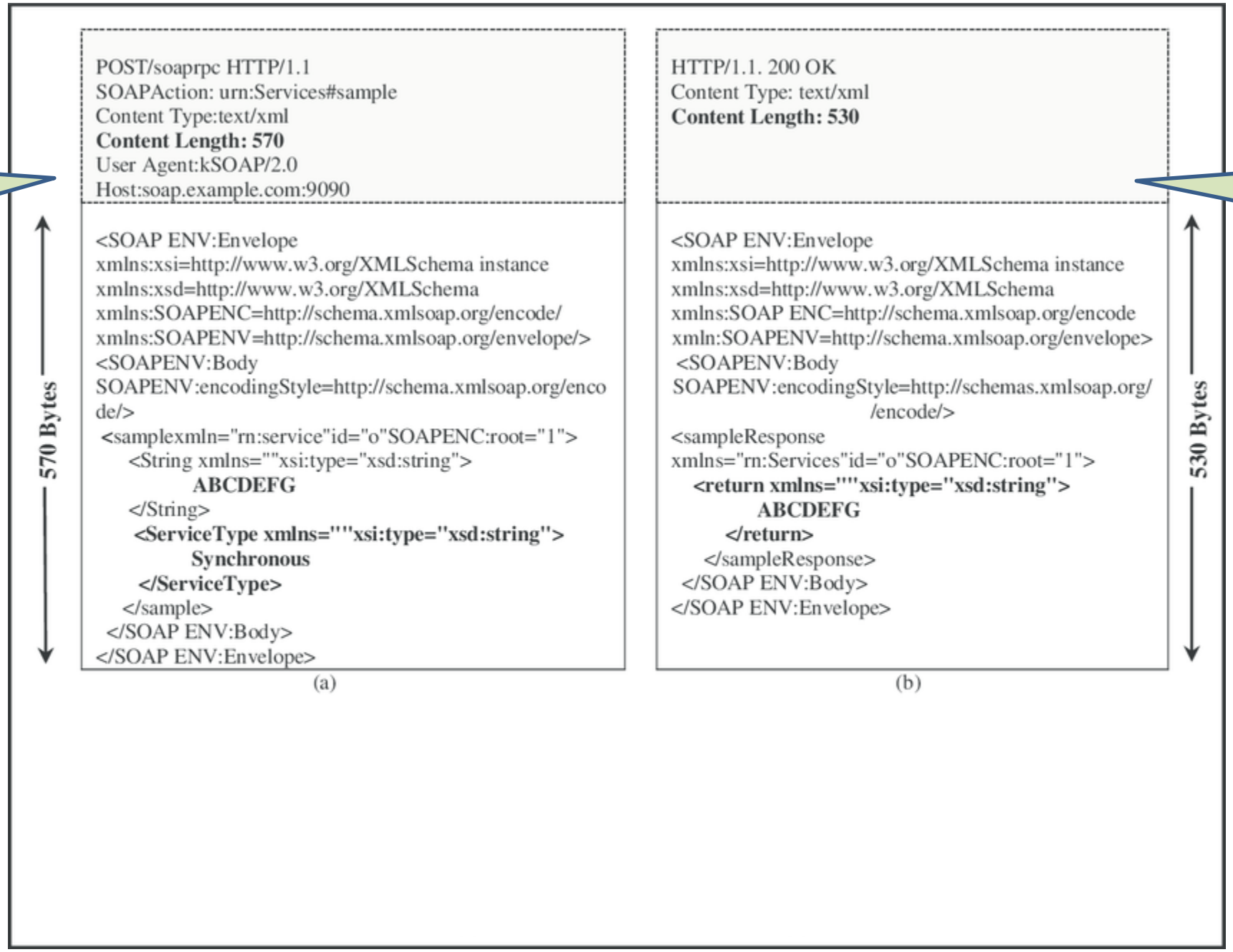
- SOAP defines message formats
- Compliant web services interpret them (in accordance with their WSDL description) and send response messages

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1"
      xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">
      http://tempuri.org/IService/MyOperation
    </Action>
    <ActivityId CorrelationId="7224e2a9-8f9c-4acb-a924-17cb6af67b23"
xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">
      43ffa660-a0c6-4249-bb36-648b73a06213</ActivityId>
  </s:Header>
  <s:Body>
    <MyOperation xmlns="http://tempuri.org">
      <MyValue>Some Value</MyValue>
    </MyOperation>
  </s:Body>
</s:Envelope>
```



SOAP REQUEST

SOAP RESPONSE



POST/soaprpc HTTP/1.1
SOAPAction: urn:Services#sample
Content Type:text/xml
Content Length: 570
User Agent:kSOAP/2.0
Host:soap.example.com:9090

```
<SOAP ENV:Envelope
xmlns:xsi=http://www.w3.org/XMLSchema instance
xmlns:xsd=http://www.w3.org/XMLSchema
xmlns:SOAPENC=http://schema.xmlsoap.org/encode/
xmlns:SOAPENV=http://schema.xmlsoap.org/envelope/>
<SOAPENV:Body
SOAPENV:encodingStyle=http://schema.xmlsoap.org/enco
de/>
 <samplexmln="rn:service"id="o"SOAPENC:root="1">
    <String xmlns=""xsi:type="xsd:string">
          ABCDEFG
    </String>
    <ServiceType xmlns=""xsi:type="xsd:string">
          Synchronous
    </ServiceType>
  </sample>
 </SOAP ENV:Body>
</SOAP ENV:Envelope>
```

(a)

HTTP/1.1. 200 OK
Content Type: text/xml
Content Length: 530

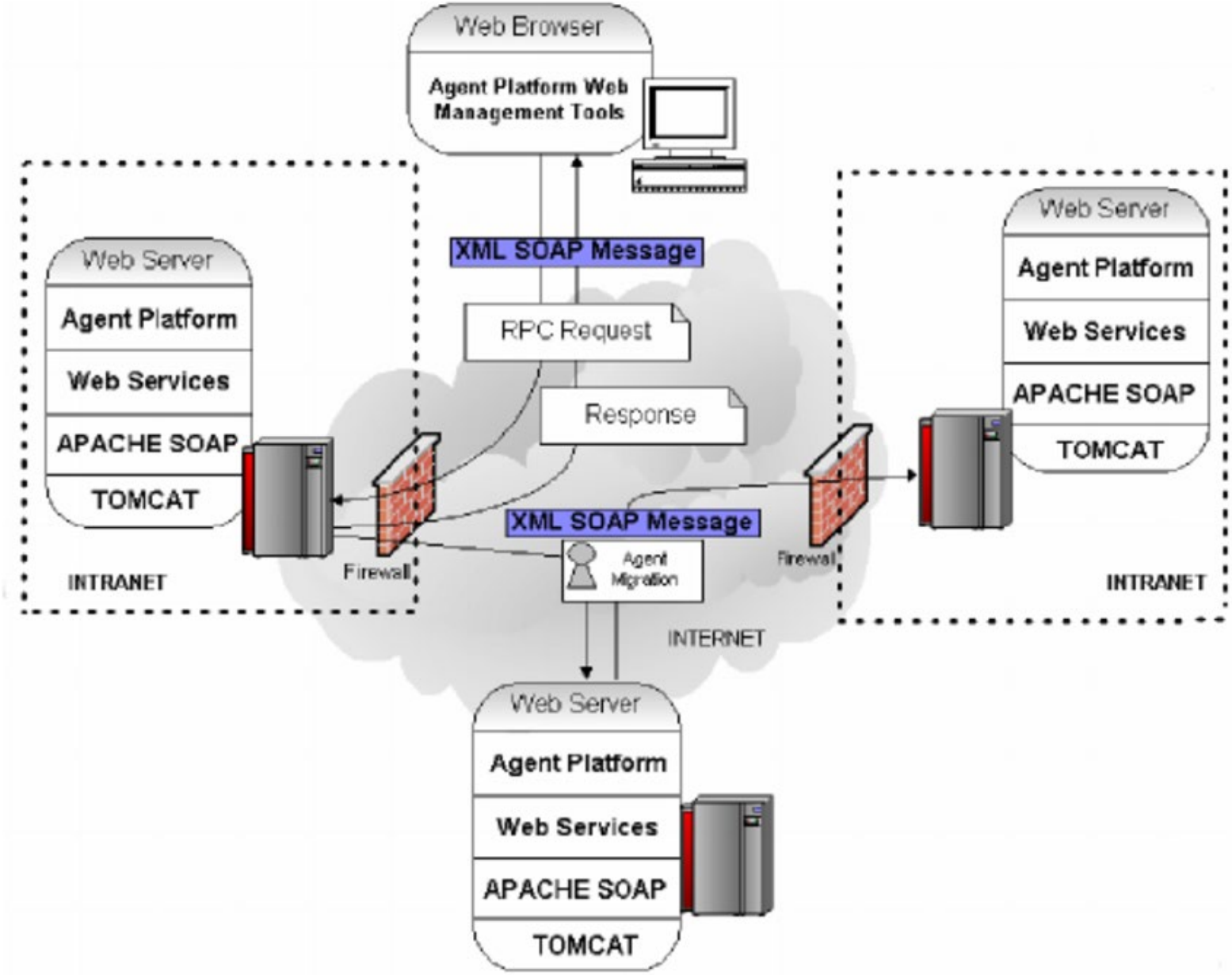
```
<SOAP ENV:Envelope
xmlns:xsi=http://www.w3.org/XMLSchema instance
xmlns:xsd=http://www.w3.org/XMLSchema
xmlns:SOAP ENC=http://schema.xmlsoap.org/encode
xmln:SOAPENV=http://schema.xmlsoap.org/envelope>
 <SOAPENV:Body
SOAPENV:encodingStyle=http://schemas.xmlsoap.org/
                    /encode/>
<sampleResponse
xmlns="rn:Services"id="o"SOAPENC:root="1">
  <return xmlns=""xsi:type="xsd:string">
        ABCDEFG
    </return>
  </sampleResponse>
 </SOAP ENV:Body>
</SOAP ENV:Envelope>
```

(b)

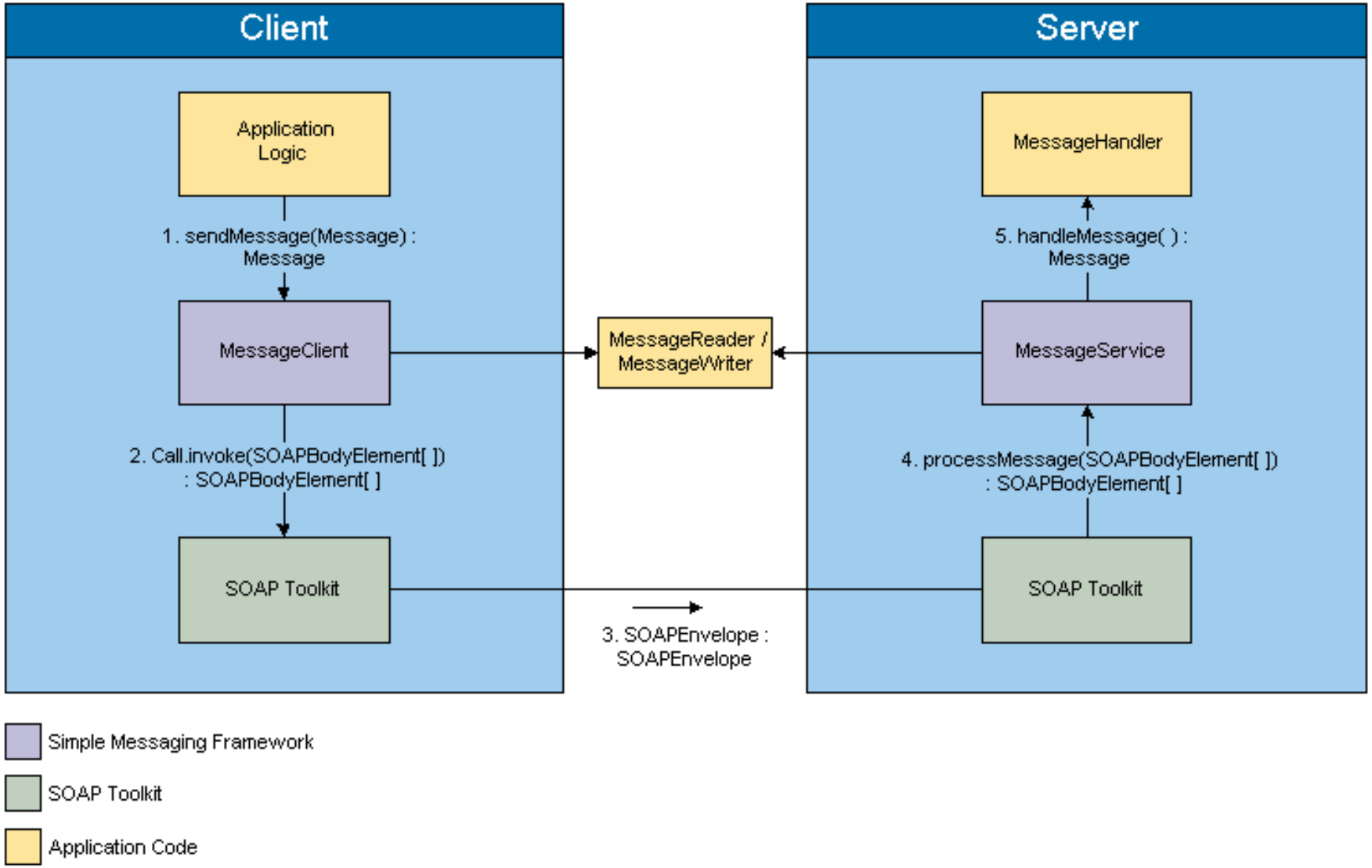
So, how do I use SOAP to communicate with a web service?

- For Java, Oracle has developed a toolkit for using SOAP:
  - http://docs.oracle.com/cd/A97630_01/appdev.920/a96616/arxml11.htm
- For C++, a number of implementations exist:
  - http://www.cs.fsu.edu/~engelen/soap.html
  - http://axis.apache.org/axis/
- For C#, SOAP support is available in the basic package:
  - https://msdn.microsoft.com/en-us/library/ff512390.aspx

A Service-Oriented Architecture using SOAP for message exchange

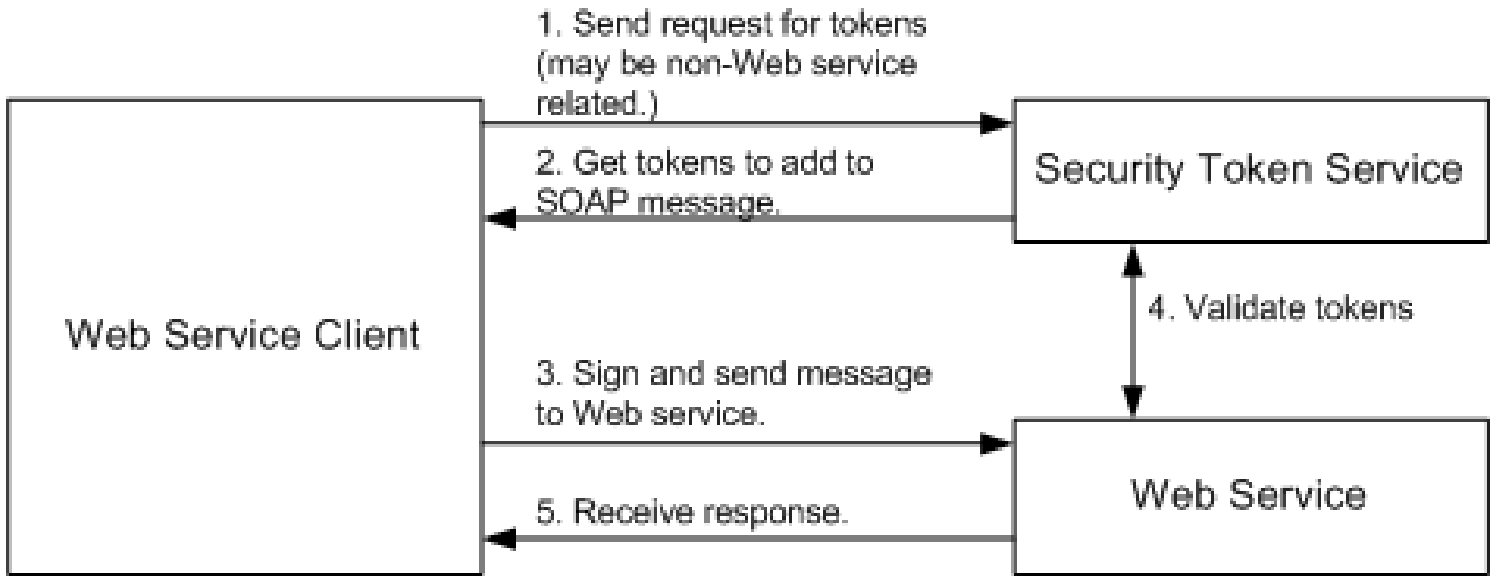


SOAP message generation and processing



Security, Transactions and Reliability

- WS-Security protocols provide ways for web services to work on the web with good privacy and security



XML – EXTENSIBLE MARKUP LANGUAGE

XML, eXtensible Markup Language, is a message format ideally suited for documents but used for many other purposes

- XML was designed for containing web documents
- The fundamental construct is the tag-value pair
- Applications can agree to abide by a particular XML *schema*
- A schema is a list of the constraints on XML documents
  - which tags are used
  - how elements are aggregated
  - etc.
- The XML standard: XML 1.0 - 5th edition (at the W3C)
  - There is an XML 1.1, only used in special situations (EBCDIC)

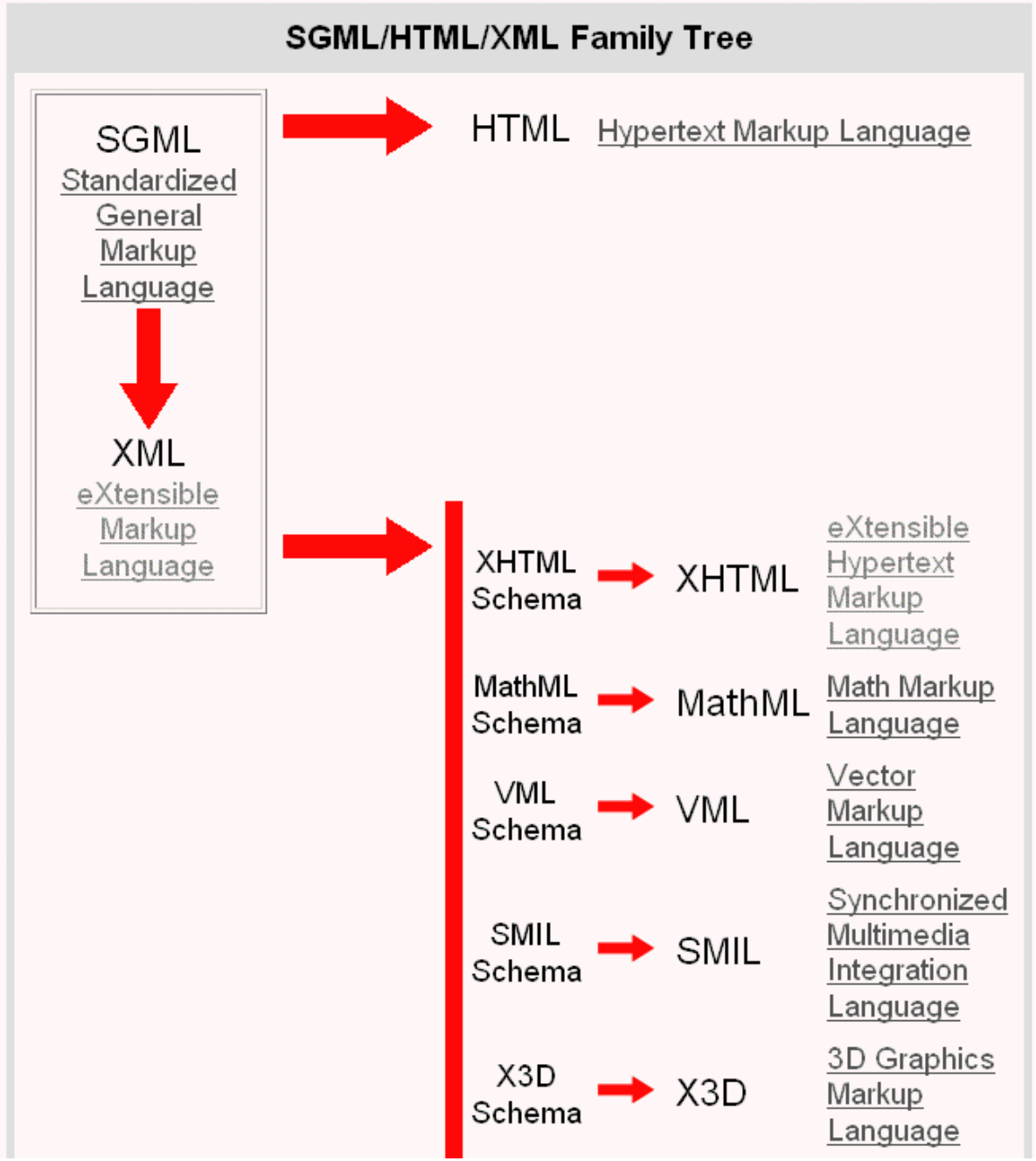
XML is based on (an eXtension of) HTML

- HTML documents are composed of one or more elements
  - Each may have elements nested within them
- An element is of this form:
  `<tag attribute1="value1" attribute2="value2">`*content*`</tag>`
- Legal tags are those defined in the HTML specification (see http://htmldog.com/reference/htmltags/)

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

XML is a form of Standardized General Markup Language;

There are many specializations of XML for graphics, math, multimedia, 3D data, etc.



In HTML, only tags defined in the specification are legal; in XML, custom tags may be defined

- The XML schema defines custom tags and constraints related to them

- A schema may be expressed in:
  - Document Type Definitions (DTDs)
  - W3C XSD (XML Schema Definitions)
  - Relax-NG
  - Schematron
  - other forms…

An XML schema may:

- provide a list of elements and attributes in a vocabulary;
- associate types, such as integer, string, etc., or more specifically such as hatsize, sock_color, etc., with values found in documents;
- constrain where elements and attributes can appear, and what can appear inside those elements, such as saying that a chapter title occurs inside a chapter, and that a chapter must consist of a chapter title followed by one or more paragraphs of text;
- provide documentation that is both human-readable and machine-processable;
- give a formal description of one or more documents.

Example: an XML file and the schema that describes it

```
<?xml version="1.0" encoding="utf-8"?>
<PurchaseOrder OrderDate="1900-01-01"
xmlns="http://tempuri.org/POSchema.xsd">
  <ShipTo country="US">
    <name>name1</name>
    <street>street1</street>
    <city>city1</city>
    <state>state1</state>
    <zip>1</zip>
  </ShipTo>
    <ShipTo country="US">
    <name>name2</name>
    <street>street2</street>
    <city>city2</city>
    <state>state2</state>
    <zip>-79228</zip>
  </ShipTo>
  <BillTo country="US">
    <name>name1</name>
    <street>street1</street>
    <city>city1</city>
    <state>state1</state>
    <zip>1</zip>
  </BillTo>
</PurchaseOrder>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="http://tempuri.org/PurchaseOrderSchema.xsd"
targetNamespace="http://tempuri.org/PurchaseOrderSchema.xsd"
        elementFormDefault="qualified">
<xsd:element name="PurchaseOrder"
type="tns:PurchaseOrderType"/>
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="ShipTo" type="tns:USAddress"
maxOccurs="2"/>
   <xsd:element name="BillTo" type="tns:USAddress"/>
  </xsd:sequence>
  <xsd:attribute name="OrderDate" type="xsd:date"/>
 </xsd:complexType>
 <xsd:complexType name="USAddress">
  <xsd:sequence>
   <xsd:element name="name"   type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="state"  type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:integer"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
 </xsd:complexType>
</xsd:schema>
```

XML syntax is based on (is an eXtension of) HTML – but there are some important differences

- In XML, all elements must be closed or marked as empty.
- Empty elements can be closed as normal, `<happiness></happiness>` or you can use a special short-form,`<happiness />` instead.
- In HTML, you only need to quote an attribute value under certain circumstances (it contains a space, or a character not allowed in a name), but the rules are hard to remember. In XML, attribute values are always quoted: `<happiness type="joy" />`
- In HTML there is a built-in set of element names and attributes. In XML, there are no built-in names (although names starting with xml have special meanings).
- In HTML, there is a list of some built-in character names like `&eacute;` for é but XML does not have this. In XML, there are only five built-in character entities: `&lt;, &gt;, &amp;, &quot; and &apos;` for <, >, &, " and ' respectively.
- In HTML, there are numeric character references, such as `&#38;` for &. ;the numbers are decimal. XML also allows hexadecimal references: `&#x26;` for example.

JSON – JAVASCRIPT OBJECT NOTATION

JSON is an alternative data format for exchange of information

- JSON stands for JavaScript Object Notation, but it's not dependent on any language
- Elements in JSON are stored as attribute-value pairs
- Attributes are strings (like field names)
- Values are numeric, string, boolean, array, collections of pairs (objects) or null values

-
```
{"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
]}
```

JSON is also written to a schema

```
{
   "id": 1,
   "name": "Foo",
   "price": 123,
   "tags": [
      "Bar",
      "Eek"
   ],
   "stock": {
      "warehouse": 300,
      "retail": 20
   }
}
```

```
{  "$schema": "http://json-schema.org/draft-03/schema#",
   "name": "Product",
   "type": "object",
   "properties": {
      "id": {
         "type": "number",
         "description": "Product identifier",
         "required": true },
      "name": {
         "type": "string",
         "description": "Name of the product",
         "required": true },
      "price": {
         "type": "number",
         "minimum": 0,
         "required": true },
      "tags": {
         "type": "array",
         "items": {
         "type": "string" }
      },
      "stock": {
         "type": "object",
         "properties": {
            "warehouse": {
               "type": "number"  },
            "retail": {
               "type": "number" }
         }
      }
   }
}
```

The JSON Specification

- Like many Internet standards, JSON is defined in a Request for Comment (RFC)
- RFC 7159
- ECMA-404 (International Standard)

A simple data record in JSON and in XML

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Comparing XML and JSON

In many ways, XML and JSON are similar -

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used in many programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest

There are important differences, however -

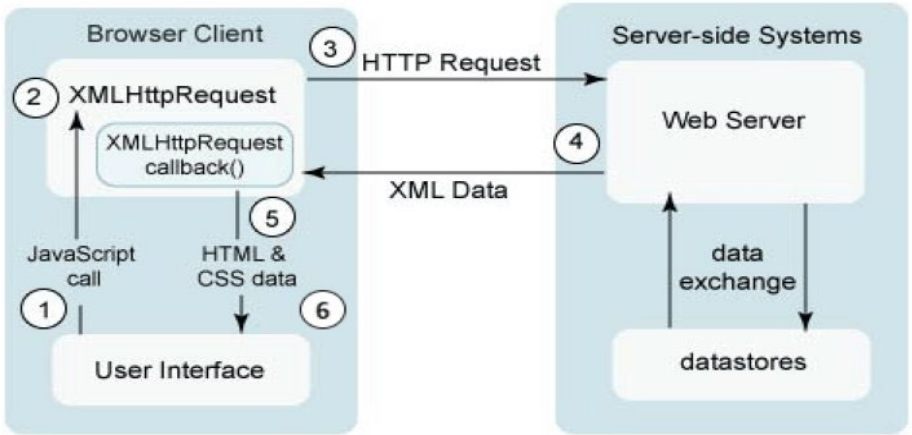
- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays

How (and why) can we create and use JSON documents in our applications?

- It's a core piece of working with AJAX
  - Asynchronous JavaScript and XML
  - An introduction is here
- JSON parsers are available in most C++ environments
- JSON works well in Java code
  - There are a number of classes and libraries freely available to create and access JSON records in Java
  - Take a look here, and here, and other places as well

Ajax is a set of technologies that work together to enable asynchronous requests and responses from web servers

- Responses are asynchronous, so often a callback function is set up to handle returned data
- Requests are made in JavaScript
- Returned data is expressed in XML
- IE supports Ajax differently than other browsers ☹
- See http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/



REPRESENTATIONAL STATE TRANSFER

REST, or Representational State Transfer, is a set of constraints and formats for web services

REST is described by four major design principles:

- **Resource identification through URI**: A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by Uniform Resource Identifiers (URIs).
- **Uniform interface**: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.
- **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. Metadata about the resource is available for control, authentication and other uses.
- **Stateful interactions through hyperlinks**: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Techniques exist to exchange state: URI rewriting, cookies, and hidden form fields.

RESTful Web Services (Representational State Transfer) is an alternative to SOAP

SOAP does not use much from existing web protocols

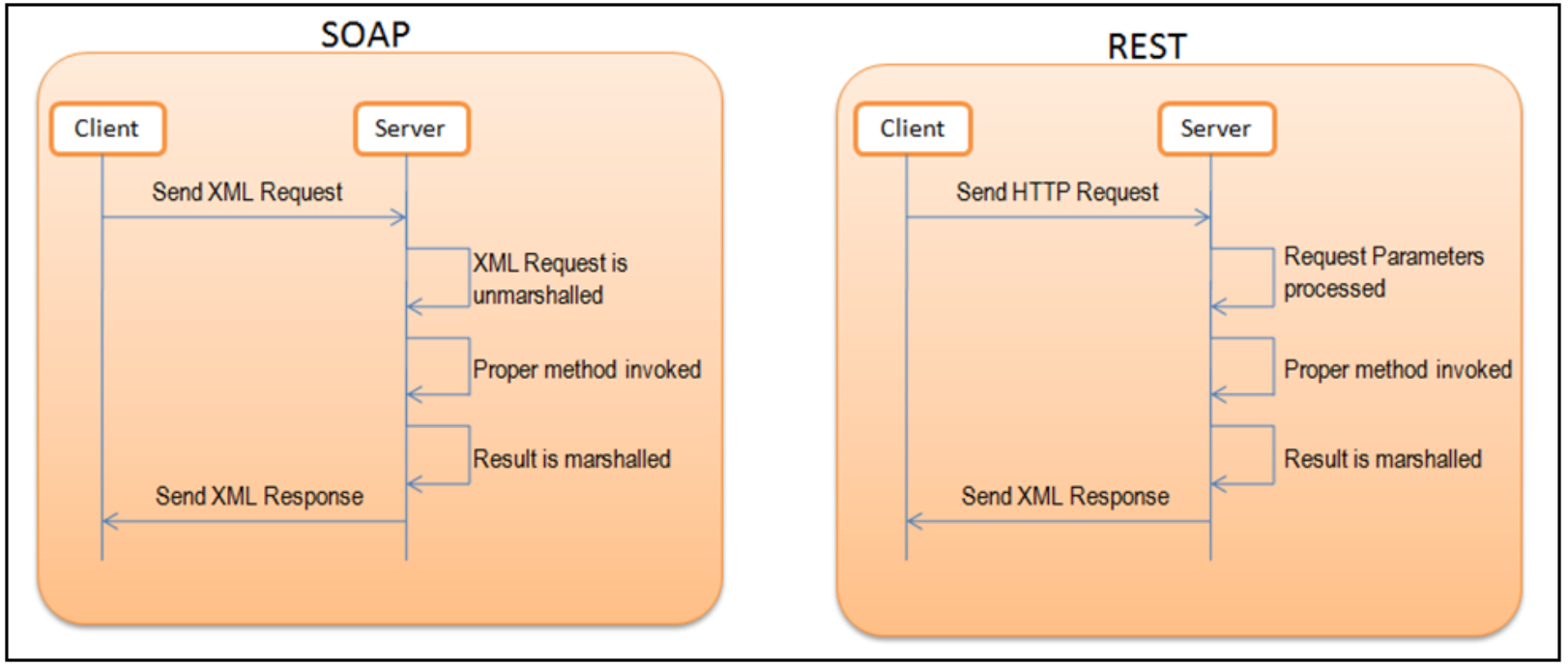
REST is developed as an alternative that is heavily dependent and compliant with other web standards

- A URI identifies a resource.
- GET requests sent to this URI return an XML document containing a list of URIs for results.
- A GET request sent to one of these URIs retrieves an XML document that contains a representation of the current state of the requested data.
- Existing resources can be updated by PUTting an XML document
- containing a the desired new state to the appropriate URI.
- New resources can be added to the database by POSTing XML documents
- containing the new resource to the collection URI.
- Resource entries can be deleted by sending a DELETE request to the appropriate URI.

REST REQUEST

REST RESPONSE

REST services respond to HTTP requests, and return results in XML or JSON (Note: this is not mandatory but most RESTful services do so)



- SOAP uses XML for both requests and replies
- Note that requests are generally simpler than replies
- REST often uses simple HTTP commands for requests and general XML or JSON for replies

Applying the design principles results in the following formal constraints on REST implementations

- Client–server
- Stateless
- Cacheable
- Layered system
- Code on demand (*optional*)
- Uniform interface
  - Identification of resources
  - Manipulation of resources through these representations
  - Self-descriptive messages
  - Hypermedia as the engine of application state

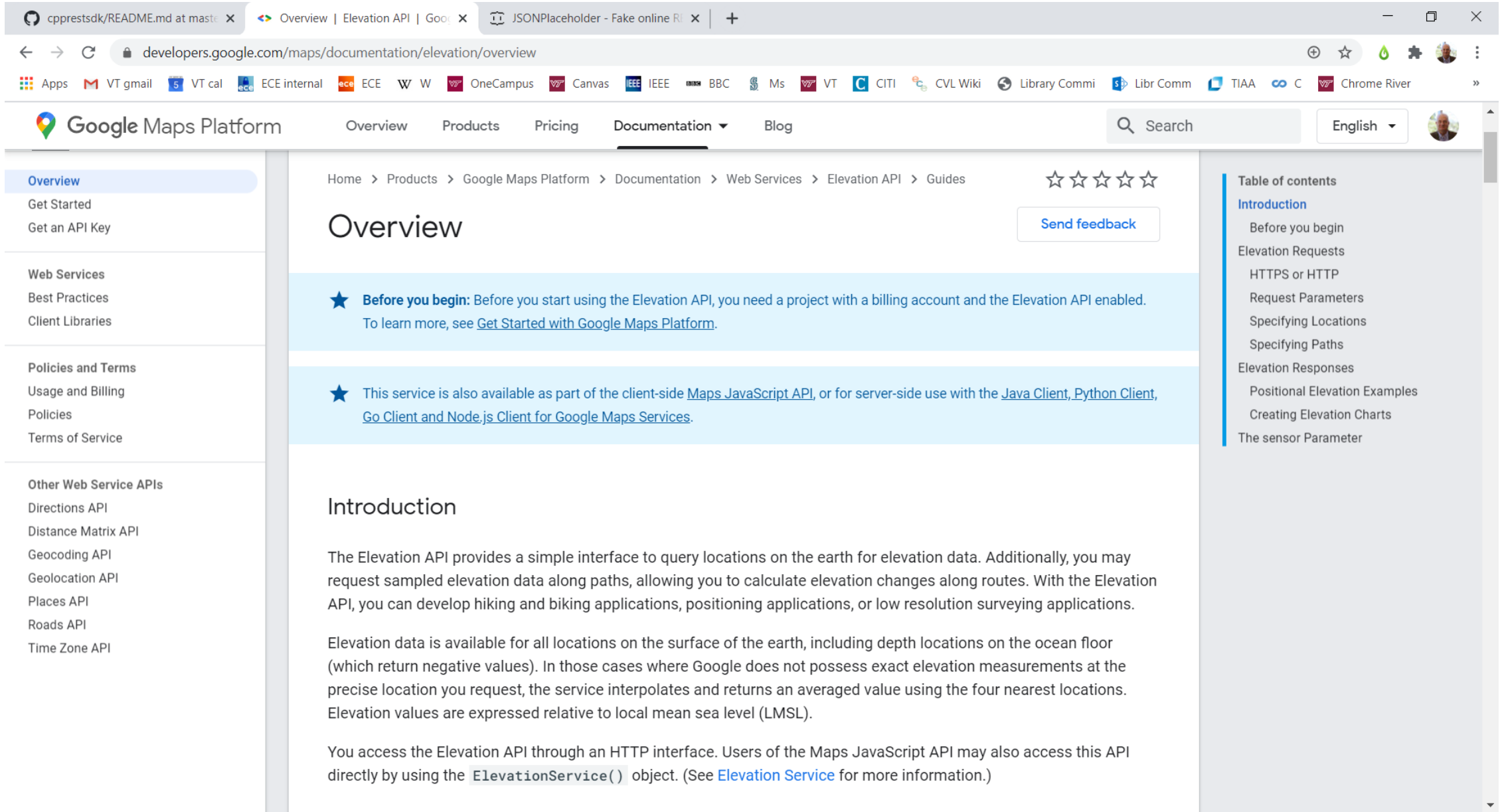
Example of a RESTful web service description: the Google Elevation API

Elevation Requests

- The Elevation API returns elevation data for locations on the earth. You specify location data in one of two ways:
  - As a set of one or more locations.
  - As a series of connected points along a path.
- Either of these approaches uses latitude/longitude coordinates to identify the locations or path vertices.
- A Google Elevation API URL must be of the following form:
  - https://maps.googleapis.com/maps/api/elevation/*outputFormat*?*parameters*
- **Note:** HTTPS is recommended for applications that include sensitive user data, such as a user's location, in requests.

Output Formats

- Outputs formats are specified using the trailing service flag in the request URL. The Elevation API currently supports the following output formats:
  - /json returns results in JavaScript Object Notation (JSON).
  - /xml returns results in XML, wrapped within a <ElevationResponse> node.


cpprestsdk/README.md at maste
Overview | Elevation API | Goo
JSONPlaceholder - Fake online R
developers.google.com/maps/documentation/elevation/overview
Apps
VT gmail
VT cal
ECE internal
ECE
W
OneCampus
Canvas
IEEE
BBC
Ms
VT
CITI
CVL Wiki
Library Commi
Libr Comm
TIAA
C
Chrome River
Google Maps Platform
Overview
Products
Pricing
Documentation
Blog
Search
English
Overview
Get Started
Get an API Key
Web Services
Best Practices
Client Libraries
Policies and Terms
Usage and Billing
Policies
Terms of Service
Other Web Service APIs
Directions API
Distance Matrix API
Geocoding API
Geolocation API
Places API
Roads API
Time Zone API
Home > Products > Google Maps Platform > Documentation > Web Services > Elevation API > Guides
Overview
Send feedback
Before you begin: Before you start using the Elevation API, you need a project with a billing account and the Elevation API enabled. To learn more, see Get Started with Google Maps Platform.
This service is also available as part of the client-side Maps JavaScript API, or for server-side use with the Java Client, Python Client, Go Client and Node.js Client for Google Maps Services.
Introduction
The Elevation API provides a simple interface to query locations on the earth for elevation data. Additionally, you may request sampled elevation data along paths, allowing you to calculate elevation changes along routes. With the Elevation API, you can develop hiking and biking applications, positioning applications, or low resolution surveying applications.
Elevation data is available for all locations on the surface of the earth, including depth locations on the ocean floor (which return negative values). In those cases where Google does not possess exact elevation measurements at the precise location you request, the service interpolates and returns an averaged value using the four nearest locations. Elevation values are expressed relative to local mean sea level (LMSL).
You access the Elevation API through an HTTP interface. Users of the Maps JavaScript API may also access this API directly by using the ElevationService() object. (See Elevation Service for more information.)
Table of contents
Introduction
Before you begin
Elevation Requests
HTTPS or HTTP
Request Parameters
Specifying Locations
Specifying Paths
Elevation Responses
Positional Elevation Examples
Creating Elevation Charts
The sensor Parameter

There are many examples of creating both RESTful clients and servers

- Here's a tutorial on creating a C++ client in REST using Visual Studio; it includes authorization
  - https://docs.microsoft.com/en-us/archive/msdn-magazine/2013/august/c-bringing-restful-services-to-c-developers
  - I will show you a simple REST client in both Java, and C++ using Qt

A simple Java routine to generate a request of a RESTful web service

jsonplaceholder.typicode.com will answer test REST requests

```
import java.net.*;
import java.io.*;

public class testRESTclient {
    public static void main(String[] args) throws IOException {
        String returnProtocol = "json";        // json or xml
        String uri = "https://jsonplaceholder.typicode.com/posts/2";
        URL url = new URL(uri);      // the web service
        HttpURLConnection connection =
                                (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Accept", "application/json");
        InputStream json = connection.getInputStream();
        BufferedReader reader =
                    new BufferedReader(new InputStreamReader(json));
        StringBuilder out = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null)
            out.append(line+"\n");
        System.out.println(out.toString());   //Prints content read
        reader.close();
        connection.disconnect();
    }
}
```

Here is a C++ class to access a REST server

```
#include "resttalker.h"

RESTTalker::RESTTalker()
{
    manager = new QNetworkAccessManager();

}

void RESTTalker::tryREST(QString hostName)
{
    QNetworkRequest request;
    connect(manager, SIGNAL(finished(QNetworkReply*)), this,
                     SLOT(replyFinished(QNetworkReply*)));

    QSslConfiguration config = QSslConfiguration::defaultConfiguration();
    config.setProtocol(QSsl::TlsV1_2);
    request.setSslConfiguration(config);
    request.setUrl(QUrl(hostName));
    request.setHeader(QNetworkRequest::ServerHeader, "application/json");
    manager->get(request);
    qDebug() << request.url();
}

void RESTTalker::replyFinished(QNetworkReply* reply)
{
    qDebug() << "REPLY: " << reply->error();
    QString qStrResponse = (QString)reply->readAll();
    QJsonDocument jsonReply = QJsonDocument::fromJson(qStrResponse.toUtf8());
    qDebug() << qStrResponse;
}
```

Today's Objectives

Service-Oriented Architectures
- Service-Oriented Systems
- Web Services
- SOAP and Messaging
- UDDI, WSDL and Metadata
- Security, Transactions and Reliability

XML
- JSON
- XML and JSON compared

RESTful web services
- What is REST?
- Concept
- Examples of RESTful clients