COMP 3322
Modern Technologies on World Wide Web

2nd semester 2017-2018

Web Service Technologies (O2)

Dr. C Wu

Department of Computer Science
The University of Hong Kong

Web service basics

- A **Web service** is a service or a piece of software offered by some service provider, which can be accessed over the Internet and used by applications running on different platforms and devices
  - to provide reusable functions, e.g., weather reports, currency conversion
  - to provide services, e.g., launching virtual machines in a public cloud
- Two major classes of Web services
  - SOAP web services
  - REST web services

SOAP Web services

- A SOAP Web service includes three roles
  - service providers
  - service requesters
  - service registry

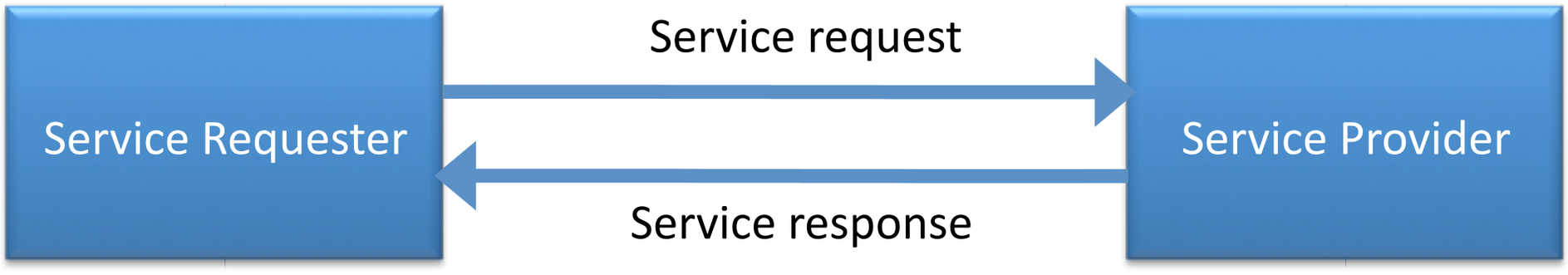
- communication between machines uses standardized XML messages
  - e.g., a service requester invokes a Web service by sending an XML message, then waits for a corresponding XML response

Service provider

- Service provider is the **provider** of the Web service; it implements the service and makes it available on the Internet.
  - Stock price querying service
  - IP geographic location lookup service
  - Weather reporting service
  - Many others: http://www.webservicex.net/

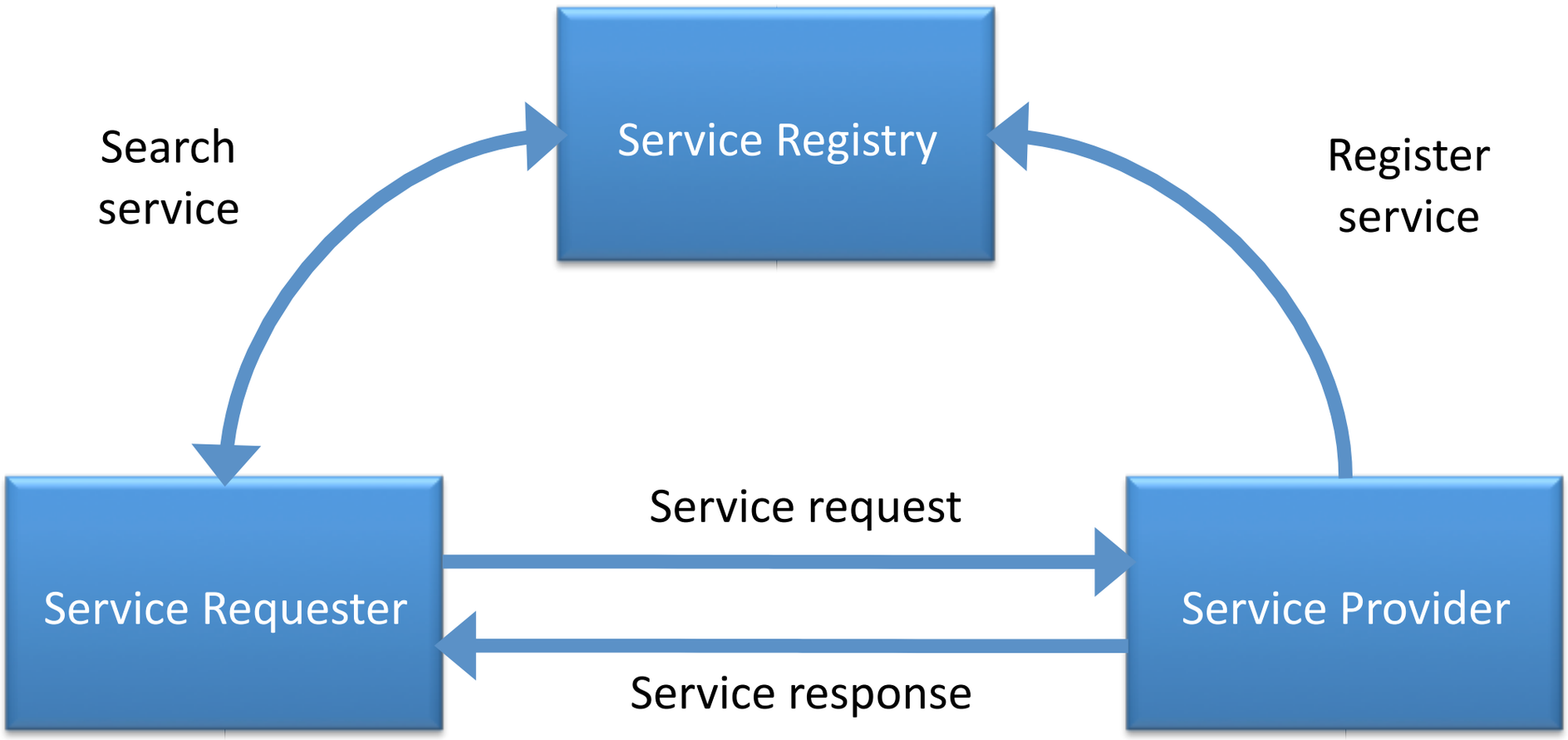
Service requester

- Service requester is any **consumer** of the Web service; it utilizes an existing Web service by opening a network connection and sending an **XML request**.



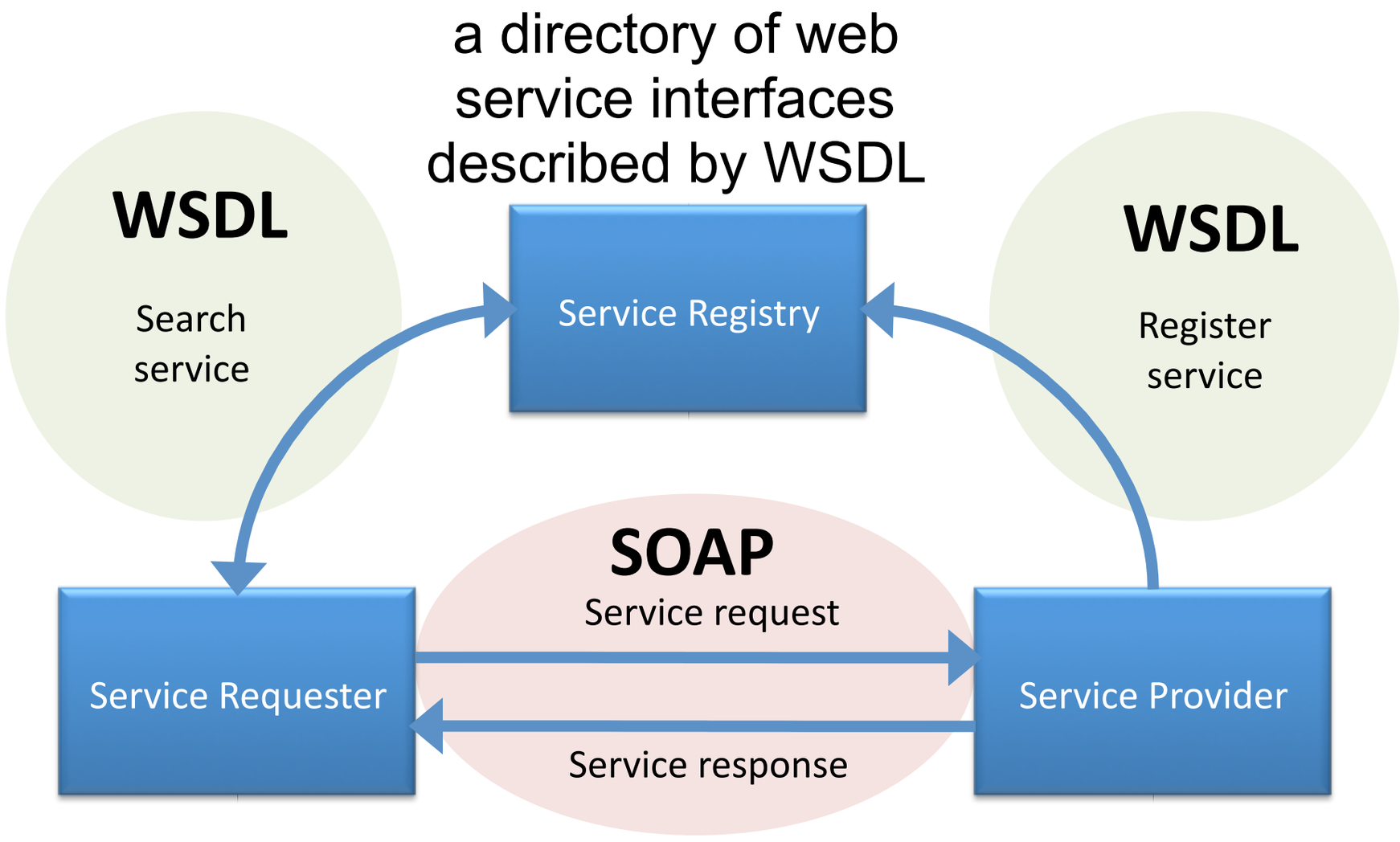
Service registry

- Service registry provides a central place where developers can publish new services or find existing ones



Main protocols

- **SOAP** (Simple Object Access Protocol) - specifies message format (in XML) between service requester and service provider
- **WSDL** (Web Services Description Language) - creates the document that describes exactly what the Web service does and how to invoke it



An example SOAP web service

- Suppose that you want to create a program that processes the stock price data and do some analysis (or even do auto trading)
  - Question: how can your program learn the stock prices?

Service Requester (You)

An example SOAP web service

A **WSDL** file

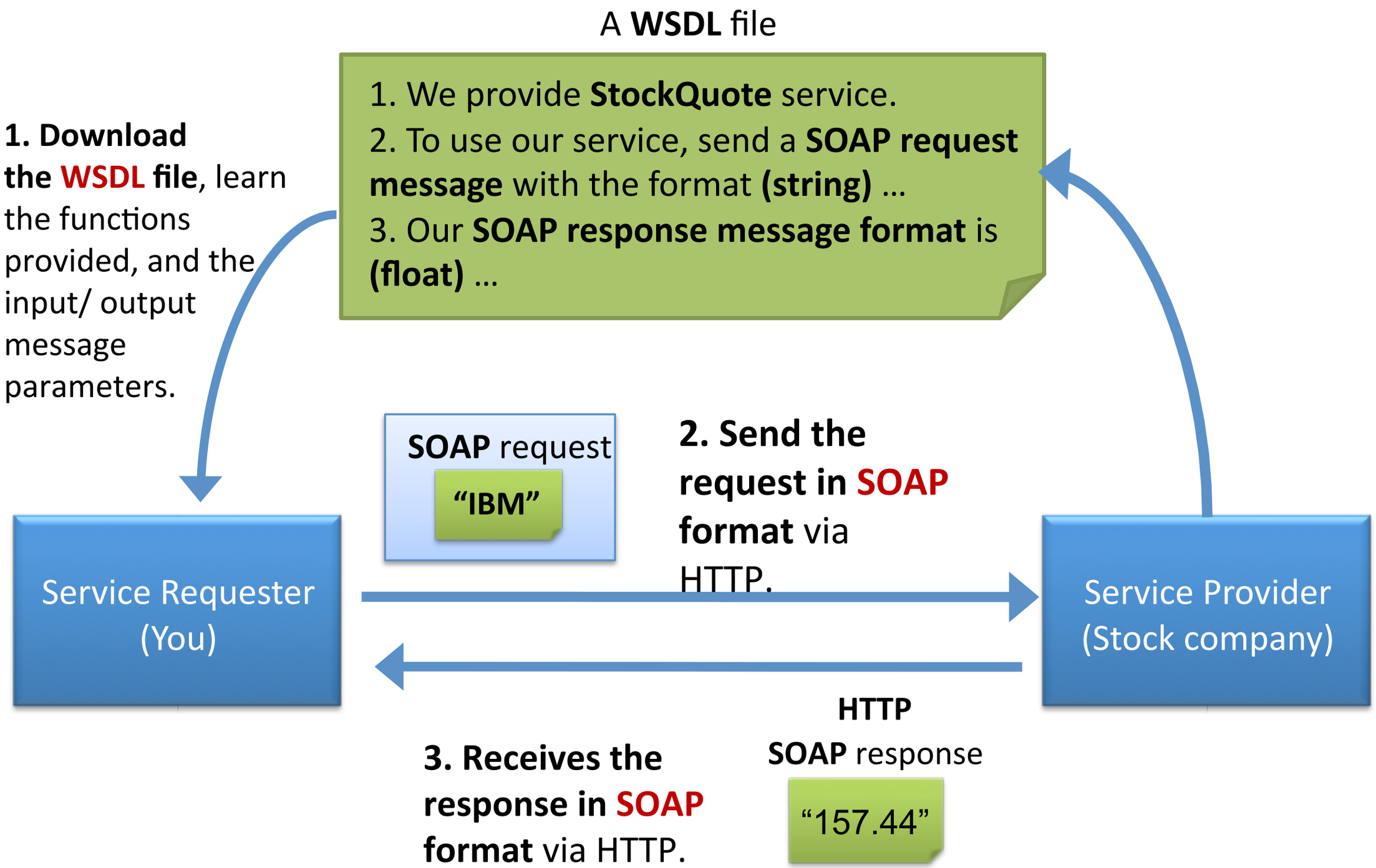
1. We provide **StockQuote** service.
2. To use our service, send a **SOAP request message** with the format **(string)** …
3. Our **SOAP response message format** is **(float)** …

Service Requester (You)

Service Provider (Stock company)

Answer:
Your program has to be able to **communicate** with “Web service providers” (e.g., Company providing stock market data service)

An example SOAP web service



An example SOAP request message

```
<?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>
      <m:getStockQuotteRequestMessage xmlns:m="http://namespaces.example.com/">
            <m:stockCode>IBM</m:stockCode>
      </m:getStockQuoteRequestMessage>
</soap:Body>


</soap:Envelope>
```

<Envelope> element identifies the XML document as a SOAP message

<Body> element contains call or response information

An example SOAP response message

```
<?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>
      <m:getStockQuoteResponseMessage xmlns:m="http://namespaces.example.com/">
            <m:stockPrice>157.44</m:stockPrice>
      </m:getStockQuoteResponseMessage>
</soap:Body>

</soap:Envelope>
```

SOAP+HTTP

- SOAP messages are typically exchanged through HTTP
  - e.g., a SOAP request sent using an HTTP POST request

```
POST /StockQuoteService.asmx HTTP/1.1
Host: www.example.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: xxx
SOAPAction: "getStockQuote"

<?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>
        <m:getStockQuoteRequestMessage  xmlns:m = "http://namespaces.example.com/">
            <m:stockCode>IBM</m:stockCode>
        </m:getStockQuoteRequestMessage >
    </soap:Body>

 </soap:Envelope>
```

SOAP+HTTP (cont'd)

- and the HTTP response message containing a SOAP response

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

<?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>
    <m:getStockQuoteResponseMessage xmlns:m="http://namespaces.example.com/">
        <m:stockPrice>157.44</m:stockPrice>
    </m:getStockQuoteResponseMessage>
</soap:Body>

</soap:Envelope>
```

RESTful Web services

- The currently more popular Web service model
- Following the principles of **Representational State Transfer** (**REST**)
  - an architecture style for designing networked applications
  - principles include client-server, stateless, cacheable, uniform interface, etc.
- As compared to its counterpart, SOAP web services, RESTful web services are simplier and easier to implement
  - there is no official standard for RESTful web services, as **REST** is an architectural style rather than a protocol
  - lightweight, highly scalable and maintainable
- Client and server typically communicate over HTTP using HTTP request/response — same as used between Web browsers and Web servers

Key components of a RESTful Web service

- Representations
- Messages
- Identification of resources (i.e., URIs)
- Uniform interface
- Caching
- etc.

Representations

- The purpose of a REST Web service is to provide a window to its clients so that they can access **resources**
  - Example resources: image/video files, Web pages, business data, APIs, or anything that can be represented in a computer system
  - Resources are identified using **URIs**
- **Representations** represent the resources and how they are related to each other
  - e.g., the server sends data as JSON or XML or HTML, which can be representations of the same resource

JSON representation of a resource

```
{
   "ID": "1",
   "Name": “Steven Lau",
   "Email": "slau@gmail.com",
   "Country": "Canada"
}
```

XML representation of a resource

```
<Person>
<ID>1</ID>
 <Name>Steven Lau</Name>
 <Email>slau@gmail.com</Email>
 <Country>Canada</Country>
</Person>
```

Messages

- The client and service communicate with each other via request/response **messages**
  - HTTP requests and responses are used in HTTP-based RESTful Web services

A POST HTTP Request

```
POST /services/persons HTTP/1.1
Host: www.myrestfulservice.com
Content-Type: text/xml; charset=utf-8
Content-Length: xxx
<?xml version="1.0" encoding="utf-8"?>
<Person>
  <ID>1</ID>
  <Name>Steven Lau</Name>
  <Email>slau@gmail.com</Email>
  <Country>Canada</Country>
</Person>
```

A HTTP Response

```
HTTP/1.1 200 OK
Date: Fri, 27 Oct 2017 13:31:04 GMT
Server: Apache/2
…
```

Uniform interfaces to access resources

- HTTP interfaces in HTTP-based RESTful Web services

Method	Operation performed on server
GET	Request a representation of the specified resource
PUT	Insert a new resource under the specified URI, or update if the resource already exists
POST	Insert a new resource as a subordinate of the resource identified by the URI
DELETE	Delete the specified resource
OPTIONS	List the allowed operations on a resource
HEAD	Return only the response headers and no response body

See HTTP request methods at https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

Uniform interfaces to access resources (cont'd)

- Examples
  - a PUT HTTP request

```
PUT /services/persons/12345 HTTP/1.1
Host: www.myrestfulservice.com
…

person 12345's data
```

  - a POST HTTP request

```
POST /services/persons HTTP/1.1
Host: www.myrestfulservice.com
…

new person's info
```

  - an OPTIONS HTTP request and its response

```
OPTIONS /services/persons HTTP/1.1
Host: www.myrestfulservice.com
…
```

```
200 OK
Allow: HEAD, GET, PUT, POST
…
```

Example: Amazon EC2 REST Web services

- An example request that launches VM instances on Amazon Elastic Compute Cloud (EC2) through an HTTP GET request

```
GET /?Action=RunInstances
&ImageId=ami-2bb65342
&MaxCount=3
&MinCount=1
&Placement.AvailabilityZone=us-east-1a
&Monitoring.Enabled=true
&Version=2014-10-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLEus-east-1%2Fec2%2Faws4_request
&X-Amz-Date=20130813T150206Z
&X-Amz-SignedHeaders=content-type%3host%3x-amz-date
&X-Amz-Signature=ced6826de92d2bdeed8f846f0bf508e8559e98e4b0194b84example54174deb456c
HTTP/1.1
Host: ec2.amazonaws.com
```

(See Amazon EC2 at https://aws.amazon.com/ec2/)

Example: Amazon S3 REST Web services

- An example request that deletes an object from Amazon S3 (Simple Storage Service) through an HTTP DELETE request

```
DELETE /puppy.jpg HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Fri, 27 Oct 2017 14:20:27 +0000
Content-Length: xxx
Authorization: signatureValue
```

(See Amazon S3 at https://aws.amazon.com/s3/)

Caching

- Clients (or a proxy server between the client and the origin server) can cache responses, so caching should be properly managed, to prevent clients from using stale data

- Caching control using HTTP headers:

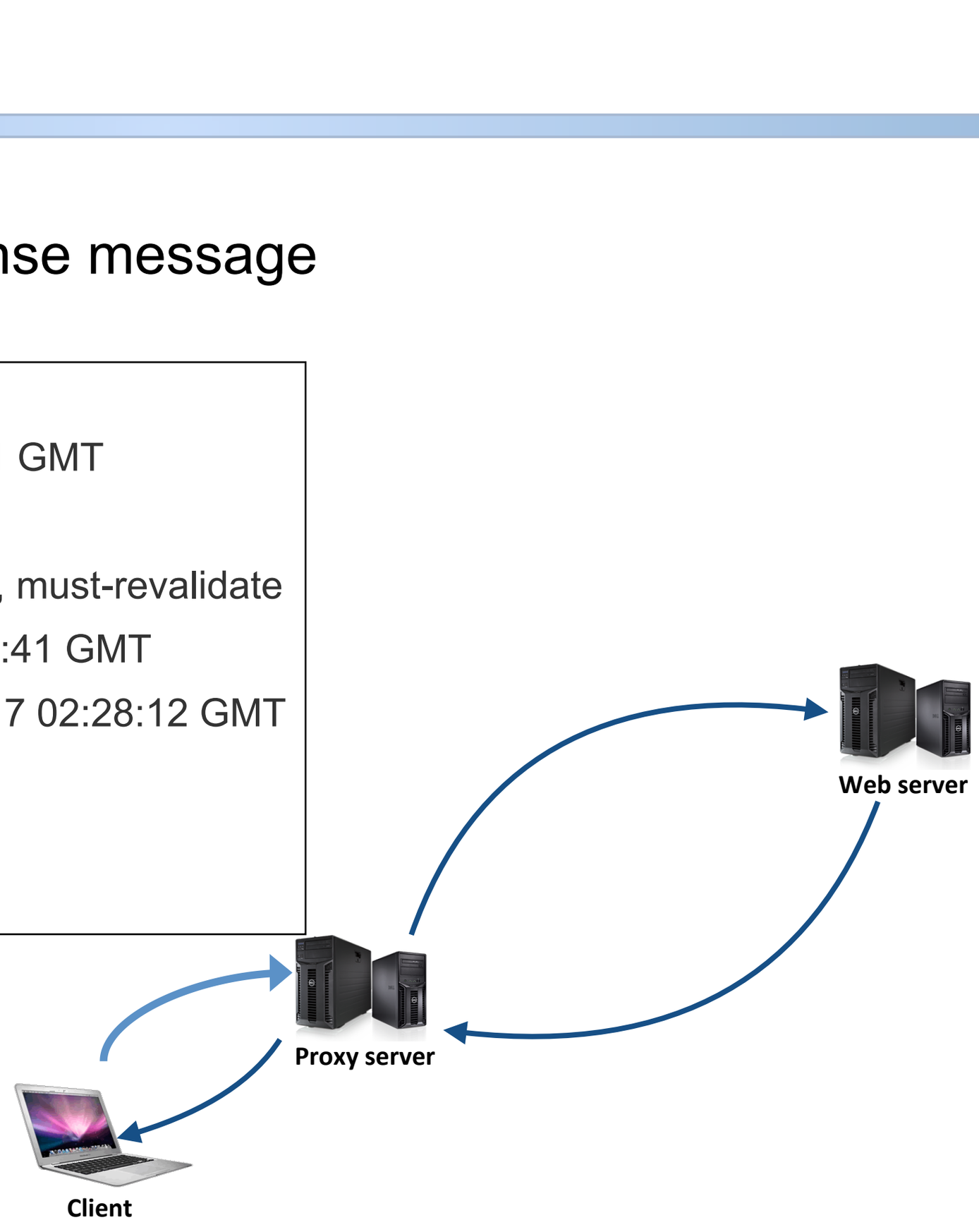
Header	Application
Date	Date and time when this representation was generated
Last Modified	Date and time when the server last modified this representation
Cache-Control	specify directives (e.g., Public, Private, no-cache/no-store) that must be obeyed by all caches along the request-response chain
Expires	Expiration date and time for this representation
Age	Duration in seconds that this has been cached

See HTTP headers at https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

Caching (cont'd)

- Example HTTP response message

```
HTTP/1.1 200 OK
Date: Fri, 27 Oct 2017 14:19:41 GMT
Server: Apache/2.4.27 (Unix)
Cache-Control: max-age=3600, must-revalidate
Expires: Fri, 27 Oct 2017 15:19:41 GMT
Last-Modified: Mon, 23 Oct 2017 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html
```



References

- **SOAP Web services:** http://www.w3schools.com/xml/xml_services.asp

- **RESTful Web services:**

  http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069?pgno=1

  http://docs.aws.amazon.com/AWSEC2/latest/APIReference/making-api-requests.html