Web/Cloud Services

CS 442: Mobile App Development
Michael Saelee <lee@iit.edu>
§ Definitions
What is a web service?
(What is *any* software service?)
a software service comprises:

- a documented set of API routines
- well defined procedure call and data exchange mechanisms

a web service is invoked “over the web”
i.e., web-enabled *remote-procedure-call* (RPC)

“web-enabled” $\rightarrow$ conducted over **HTTP**

**HTTP** = hypertext transfer protocol, lingua franca of web clients/servers
HTTP = universal language of web servers and clients
important: HTTP is a *stateless* protocol

i.e., it does not inherently track the state or progress of a client/server conversation
e.g., Client: GET /~lee/index.html
Server: (returns data)
Client: GET it again
Server: It?
Client: Which part of that was unclear?
Server: That?
resulting problems: how to track a user? how to maintain sessions across multiple page views?
additional data can be specified in HTTP request/response headers
e.g., caching behavior, last modified date, **cookies**, compression, authorization
in fact, can easily cheat and make HTTP stateful by tracking each request on server and sending tracking data in each response ... 

... disadvantages?
controversy: “hashbang” URLs

- Ajax manipulated (partial page refreshes)

- horrible for portable URLs!
only nine HTTP methods (aka “verbs”); only 4 typically used in web service APIs:
- GET
- POST
- PUT
- DELETE
akin to an API with a fixed number of procedures

… how to build rich services with this?
use a large number of different arguments!
different approaches:

- extend the “verb” vocabulary of HTTP by way of the “arguments”
- stick to just the basic verbs, but enrich their semantics
approach 1:

- service defined with web services description language (WSDL)
- XML based transport format
  - favor simple object access protocol (SOAP)
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
 xmlns:tns="http://example.com/stockquote.wsdl"
 xmlns:xsd="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">
    <input>
     <soap:body use="literal"/>
    </input>
    <output>
     <soap:body use="literal"/>
    </output>
   </operation>
  </soap:binding>
 </binding>
 <service name="StockQuoteService">
  <documentation>My first service</documentation>
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">
   <soap:address location="http://example.com/stockquote"/>
  </port>
 </service>
</definitions>
<?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
    xmlns:m="http://example.com/stockquote.wsdl">
    <m:TradePriceRequest>
      <m:tickerSymbol>IBM</m:tickerSymbol>
    </m:TradePriceRequest>
  </soap:Body>
</soap:Envelope>

<?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
    xmlns:m="http://example.com/stockquote.wsdl">
    <m:TradePrice>
      <m:price>34.5</m:price>
    </m:TradePrice>
  </soap:Body>
</soap:Envelope>
problems:

- WSDL is unwieldy and annoying to craft
  - not human readable (really)
  - “Javadoc issue”
- XML sucks
  - large, verbose, and annoying to parse
approach 2:

- RESTful services (huh?)
- typically, non-XML transport formats (e.g., JSON)
REpresentational State Transfer (REST)
REST characteristics:

- URLs identify *resources*
- clear HTTP verb semantics
- emphasizes statelessness
- no client state stored on server
GET = *fetch* the resource

POST/PUT = *create/update* the resource (with request body)

DELETE = *delete* the resource
GET = read only

PUT, DELETE = idempotent
  (i.e., can perform multiple times)

POST = may advance server state each time
http://www.fooblog.com/posts/2012/4/

- GET to retrieve list of posts
- POST to create new post


- GET to retrieve post body
- PUT to update post
- DELETE to delete post
operate on web “resources”; identified by uniform resource locators (URLs) e.g., http://www.cs.iit.edu/~lee
§Building web services
§e.g., Parse
Data Interchange & Connectivity (iOS)
JSON
HTTP request / response
NSURLConnection
Alternatives: ASIHTTPRequest, etc.
§Case Study: AWS
(Amazon Web Services)
AWS services (partial listing):
  - Simple storage service (S3)
  - Elastic Compute Cloud (EC2)
S3: on-demand, cloud-based storage
- multiple developer created “buckets”
- each bucket is a key-object map
- buckets/objects can be access-controlled
Elastic Compute Cloud (EC2)
§ Case Study: Django
Python. web. framework.
Python.

- object-oriented / procedural
- dynamically (duck) typed
- deep introspection
- lambdas & decorators
- vast standard + third-party libraries
web.

- geared towards building web services
- HTTP based API
- requires a web host/server
framework.

- libraries for building web services
- relatively low level (vs. CMS, Blogging platform)
I. URL routing & handlers
http://example.com/2012/4/ → handler_fn

II. Web view templating system
III. Web form processing
IV. Session management
V. Authentication / Authorization
VI. Persistence / Object-Relational Mapping (ORM)