Web Services

REST (Representational State Transfer)

- Servers communicate with clients using stateless connections.
  - all information about the state is encoded in the request and response
- Long term state is kept at the server as a set of identifiable resources
- Benefits
  - simplify applications
- Use verbs to act on nouns
  - HTTP methods: GET, POST, PUT, HEAD, DELETE.
  - Nouns are the resources

REST

- Tutorial video - http://bitworking.org/news/373/An-Introduction-to-REST
- We will see the tutorial and build on it.

Resources and methods in REST

http://localhost/articles defines methods on article

- GET gives you an list of articles
- POST creates an article (with post data)
- GET /articles/1 gets an article
- PUT /articles/1 updates
- DELETE /articles/1 deletes

- Rails does this via routes
  - map GET +/articles to some method like list_articles.., etc.

REST (cont.)

- The HTTP based web uses many REST principles
  - URLs – to identify webpages
  - Verbs to GET, POST to them

- and breaks other principles
  - cookies to store sessions at server
    - client side state is OK because that is what REST wants you to do

- So if no sessions are stored then
  - How would you maintain login states?
    - use keys and hashes, use http authorization headers

  - How would you store shopping carts?
    - it becomes a separate resource :
      - http://www.myamazon.com/cart/2 - put to update
      - http://www.myamazon.com/cart - post to create

Using an API – General steps

- Find if the data you want has an API
- Most APIs need keys – apply for a key on the developer page.
- A key identifies you as a developer
- Study the various queries possible using the API
- Form the queries and test them in a browser (use your key)
- See the result in the browser (JSON/XML) and write code to parse that result in your program.
- Go nuts and mashup web content!

Example – Google Places API

- Go through http://code.google.com/apis/maps/documentation/places/

- URL:
- https://maps.googleapis.com/maps/api/place/search/

  json?location=39.5765846,-76.3865142

  &radius=50000&sensor=false

  &key=AIzaSyDbI1ASxiFGVbXNHwjEcFIJ QXvHsQ1YRDg

  &name=Giant

Click here to run

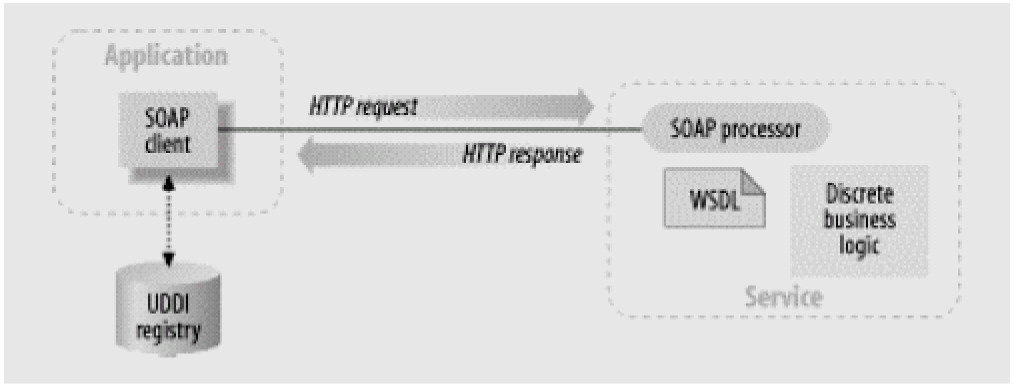
Web Services

- REST is meant to make resource access simple

- Web Services – the other end of the spectrum
  - Tools for generating and publishing full-fledged APIs

Big Web Services

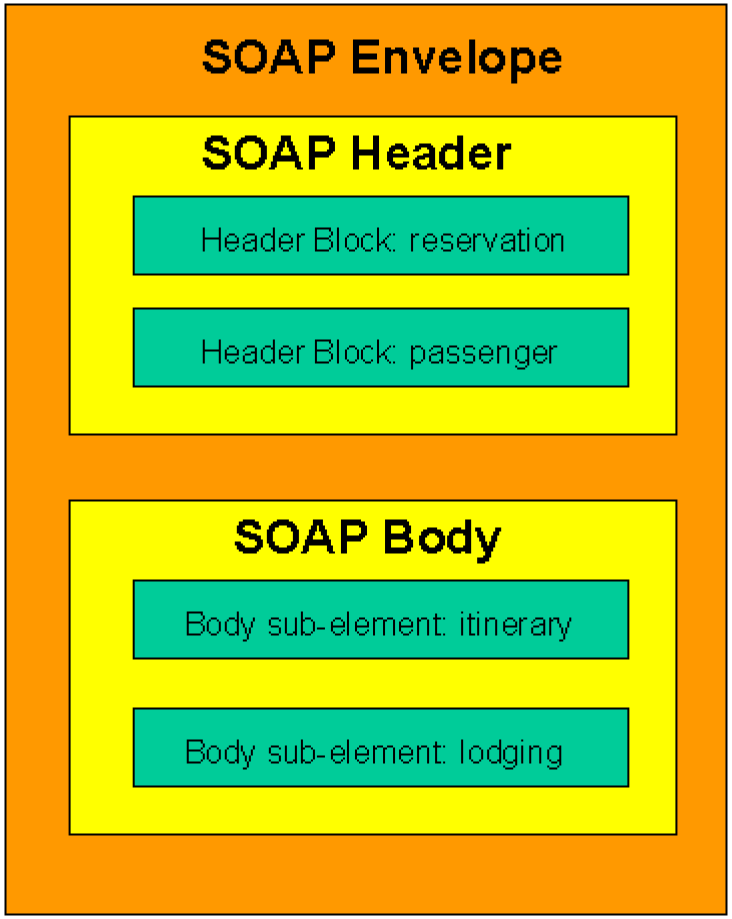
- Simple Object Access Protocol (SOAP)
- Web Service Description Language (WSDL)
- Universal Description, Discovery and Integration (UDDI)



SOAP

- Simple Object Access Protocol (SOAP)
- Service Oriented Architecture Protocol (SOAP)
- Runs over HTTP or SMTP
- Complex – many namespaces
- Systems should be writing the XML, not people

Graphical Representation of SOAP Envelope


SOAP Envelope
SOAP Header
Header Block: reservation
Header Block: passenger
SOAP Body
Body sub-element: itinerary
Body sub-element: lodging

http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

SOAP Example – from W3C SOAP Primer

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">

<env:Header>
   <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
          env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
   </m:reservation>
   <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>Åke Jógvan Øyvind</n:name>
   </n:passenger>
</env:Header>
```

http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

SOAP Example (cont.)

```
<env:Body>
<p:itinerary
xmlns:p="http://travelcompany.example.org/reservation/travel">
<p:departure>
   <p:departing>New York</p:departing>
   <p:arriving>Los Angeles</p:arriving>
   <p:departureDate>2001-12-14</p:departureDate>
   <p:departureTime>late afternoon</p:departureTime>
   <p:seatPreference>aisle</p:seatPreference>
</p:departure>
<p:return>
……
</env:Body>
</env:Envelope>
```

http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

SOAP Companion Protocols

- WSDL: Web Services Description Language
  - describe calls available in terms of inputs and outputs
  - packages of calls that are available

- UDDI: Universal Description, Discovery, and Integration
  - registry for services
  - White pages – contact information
  - Yellow Pages – industrial categorizations
  - Green – technical info about services

Web-services - thoughts

- Widely used and potentially important
  - ...but also cumbersome and potentially tricky
- Interfaces, standards, implementations change
  - Google had a SOAP interface – no new keys after 2006
- Abstract out the details – use libraries, gems
- Don't hard-code
  - Use toolkits – libraries for various services –google maps, etc...
  - Then again, you are hiding complexity, multiple toolkits..
- With the semantic web – things might get easier? more complex?

SOAP Issues

- SOAP/WSDL vs. something simpler
  - SOAP is powerful, flexible, but complicated
  - Though REST is simpler, but harder to debug/
    - if you get an error with REST? There is hardly any feedback
- XML vs. something else
  - JavaScript Object Notation (JSON) for sending structured data with less overhead?
  - XML is supposed to be written by systems, not humans
- What is the audience?
  - SOAP for composing services vs. REST for end-user web applications?