An Overview of REST

Design Concepts

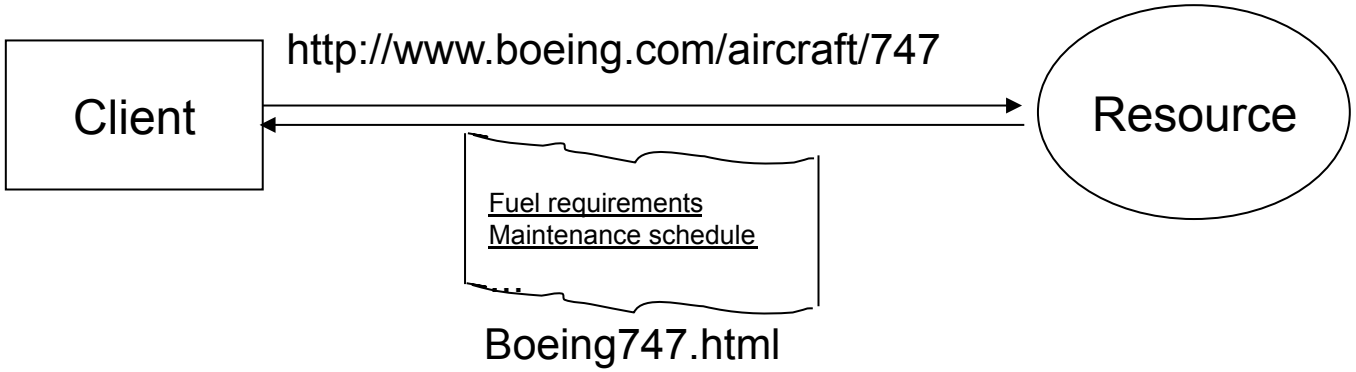
Theory (Fielding)

DESIGN CONCEPTS

REST (REpresentational State Transfer)

REST by Example:

- The Client references a Web resource using a URL (URI)
- A resource representation is returned (an HTML document)
- Representation (e.g., Boeing747.html) puts client in new state
- When the client selects hyperlink in Boeing747.html, it accesses another resource
- New representation places client into yet another state
- Client transfers state with each resource representation



REST Motivations

REST was coined and developed by Roy T. Fielding in his PhD dissertation at UC Irvine at the turn of the millenium

- In it, he provides some pretty powerful conceptual arguments for REST as an architectural style.

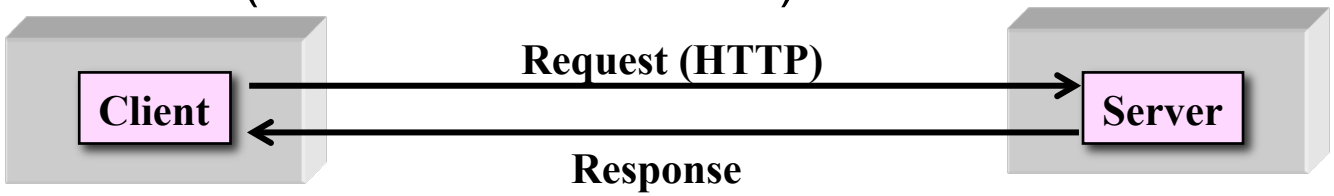
On a more practical note, there are some key motivations for the (delayed and then) growing adoption of REST:

1. Web application architectures are evolving from the primarily server-centric style (which you have done in this class) to a more client-centric style (which SER421 covers).
   - Organizations needs to have some sanity (framework) on how to manage their interfaces
2. The Web Services (WS*) stack became unwieldy, and organizations sought a faster, lighter, leaner (agile?) way of providing integration interfaces.
   - They were looking for an excuse to do it the way they wanted

Quick HTTP and URI Recap

REST is tightly coupled to HTTP, so a brief review:

- HTTP is an *application layer* **protocol** typically built on TCP/IP
- **Synchronous** (what does this mean?)
- **Stateless** (what does this mean?)



HTTP supports various "methods" (which is why it is coupled)

- **GET**: Makes a request on a resource
- **POST**: Used to pass input to the server
  - Let's change POST a little: a POST *creates a resource on the server*
- **PUT**: Creates or updates a resource on the server
- **DELETE**: removes a resource or collection of resources
- The are the main 4 to REST, but we will explore the use of **PATCH** versus PUT later

REST Definitions

Resource

- any object or concept that can be named (nouns not verbs)
- the resource does not have to exist (yet)
  - examples: books, animals, phones, unicorns
- Resources are linked to each other - navigated

Resource Identifier

- logical name and address for a resource
- implemented as a URI (URL)
- example: www.myexample.com/papers/rest.doc

Representation

- contains information about a resource's state
- in a specific format (Mime media types like text/html)
- examples: html, xml, mp4

Rajendran, Doran, Balasubramaniam, 2009

An Example of RESTful Web Service

Service: Get a list of parts

- Web service makes an available URL to a parts list resource
- A client uses URL http://www.parts-depot.com/parts to get the parts list
  - How web service generates the parts list is completely transparent to the client
- Each resource is identified as a URL
  - Parts list has links to get each part's detailed info
- Key feature of REST design pattern: Client transfers from one state to next by choosing from alternative URLs in the response

```
<Parts>
      <Part id="00345" href="http://www.parts-depot.com/parts/00345"/>
      <Part id="00346" href="http://www.parts-depot.com/parts/00346"/>
      <Part id="00347" href="http://www.parts-depot.com/parts/00347"/>
      <Part id="00348" href="http://www.parts-depot.com/parts/00348"/>
</Parts>
```

```
<Part>
     <Part-ID>00345</Part-ID>
     <Name>Widget-A</Name>
     <Description>This part is used within the frap assembly</Description>
     <Specification href="http://www.parts-depot.com/parts/00345/specification"/>
     <UnitCost currency="USD">0.10</UnitCost>
     <Quantity>10</Quantity>
</Part>
```

Response Formats of RESTful Web Services

XML or JSON

- JSON is more popular, as the primary use case for RESTful services is consumption in a browser application
- http://www.omdbapi.com/?s=titanic[&r=xml]

```
<root response="True">
<Movie Title="Titanic" Year="1997" imdbID="tt0120338" Type="movie"/>
<Movie Title="Titanic II" Year="2010" imdbID="tt1640571" Type="movie"/>
<Movie Title="Titanic: The Legend Goes On..." Year="2000" imdbID="tt0330994" Type="movie"/>
<Movie Title="Titanic" Year="1953" imdbID="tt0046435" Type="movie"/>
<Movie Title="Titanic" Year="1996" imdbID="tt0115392" Type="movie"/>
<Movie Title="Raise the Titanic" Year="1980" imdbID="tt0081400" Type="movie"/>
<Movie Title="Titanic" Year="2012" imdbID="tt1869152" Type="series"/>
<Movie Title="The Chambermaid on the Titanic" Year="1997" imdbID="tt0129923" Type="movie"/>
<Movie Title="Titanic: Blood and Steel" Year="2012" imdbID="tt1695366" Type="series"/>
<Movie Title="Titanic" Year="1943" imdbID="tt0036443" Type="movie"/>
</root>
```

```
{"Search":[{"Title":"Titanic","Year":"1997","imdbID":"tt0120338","Type":"movie"},
{"Title":"Titanic II","Year":"2010","imdbID":"tt1640571","Type":"movie"},{"Title":"Titanic:
The Legend Goes On...","Year":"2000","imdbID":"tt0330994","Type":"movie"},
{"Title":"Titanic","Year":"1953","imdbID":"tt0046435","Type":"movie"},
{"Title":"Titanic","Year":"1996","imdbID":"tt0115392","Type":"movie"},{"Title":"Raise the
Titanic","Year":"1980","imdbID":"tt0081400","Type":"movie"},
{"Title":"Titanic","Year":"2012","imdbID":"tt1869152","Type":"series"},{"Title":"The
Chambermaid on the Titanic","Year":"1997","imdbID":"tt0129923","Type":"movie"},
{"Title":"Titanic: Blood and Steel","Year":"2012","imdbID":"tt1695366","Type":"series"},
{"Title":"Titanic","Year":"1943","imdbID":"tt0036443","Type":"movie"}]}
```

REST THEORY (FIELDING)

REST Theory

- Introduced and defined by Roy Fielding in his Ph.D. dissertation (UC Irvine 2000)
  - At the time he was working w/ others on HTTP 1.1 & URI standards
- REST defined a set of architectural constraints to make a system ***anarchically stable***
  - Uh, what does that mean? Fielding 2002 (emphasis mine):

    "Most software systems are created with the implicit assumption that the entire system is under the control of one entity, or at least that *all entities participating within a system are acting towards a common goal and not at cross-purposes. Such an assumption cannot be safely made when the system runs openly on the Internet*. Anarchic scalability refers to the need for architectural elements to continue operating when subjected to an unanticipated load, or when given malformed or maliciously constructed data, since they may be communicating with elements outside their organizational control. The architecture must be amenable to mechanisms that enhance visibility and scalability.

Stable operation in the face of anarchy! Even better than graceful degradation!

Summary: Fielding's 6 principles

1. Uniform Interface
   - 4 principles we discussed: *Resources, Representations, Self-describing (navigable), and HATEOS*
2. Stateless
   - Know what state we are talking about. REST calls our conversational state *application state*, while our world model state is called *resource state*. Resource state is not client dependent. Application state has to be relayed by client in-band so the server can decide if there is a stateful decision to be made. In other words, *the client must transmit all information the server would need to make a decision on how to respond* (including state)
3. Cacheable
   - Responses should indicate whether they can be cached, as client-side caching of responses is a common optimization practice

Summary: Fielding's 6 principles

4. Client-Server
   - Each side has its own responsibility. Cleanly decoupled and can evolve independently (well, there is that versioning thing)
5. Layered System
   - Intermediary servers may be present along the route to enhance QoS perspectives such as scalability (load balancer), performance (caching proxy), security (firewall or other), etc.
   - I don't like the term *Layered* here as it obscures the topology of Web connectivity. It is more like a *Decorator* pattern
6. Code on Demand
   - Servers may temporarily extend client functionality by transferring logic (code) to the client for execution.
   - Examples include common plugin types, applets, & Javascript
   - Can't say I really understand how this agent-based style of thinking is RESTful, sounds like Fielding threw it in. But that is OK as it is optional and most folks don't discuss as part of REST

References

1. T. Fredrich. *RESTful Best Practices Guide*, Pearson eCollege, August 2013. http://www.restapitutorial.com/media/RESTful_Best_Practices-v1_1.pdf
2. Paliath, V.S. “An Overview of REST”, Available online at *vivin.net/pub/REST.pdf*. Last accessed March 2018.
3. Fielding, R.T., and Taylor, R.N. “Principled Design of the Modern Web Architecture”, *ACM Transactions on Internet Technology*, *2*(2):May 2002, pp. 115-150.
4. Fielding, R.T. “Architectural Styles and the Design of Network-based Software Architectures”, Ph.D. dissertation, UC-Irvine, 2000. Available at

   https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

* Reference 3 is basically taken straight from parts of 4