HTTP & REST

Noah Mendelsohn
Tufts University
Email: noah@cs.tufts.edu
Web: http://www.cs.tufts.edu/~noah
Architecting a universal Web

- Identification: URIs
- Interaction: HTTP
- Data formats: HTML, JPEG, GIF, etc.
Goals

- Learn why traditional architectures would not have worked for the Web
- Learn the basics of HTTP (review)
- Explore some interesting characteristics of HTTP
- Learn the REST distributed computing architecture
Question:
What would happen if we built the Web from RPC?
Why not use RPC for the Web?

- **No uniform interface**
  - You’d need an interface definition to follow a link

- **No global naming**

- **Most RPC data types poorly tuned for documents**
  - Int, float, struct *not* HTML, XML, mixed-content, etc.

We’ll revisit the RPC/Web question later and add some more
HTTP in Action
(review from first week)
The user clicks on a link

URI is http://webarch.noahdemo.com/demo1/test.html
The http “scheme” tells client to send HTTP GET msg

URI is \texttt{http://webarch.noahdemo.com/demo1/test.html}
The server is identified by DNS name in the URI

URI is http://webarch.noahdemo.com/demo1/test.html

HTTP GET

Host: webarch.noahdemo.com
The client sends an HTTP GET

URI is http://webarch.noahdemo.com/demo1/test.html

HTTP GET

demo1/test.html

host: webarch.noahdemo.com

GET /demo1/test.html HTTP/1.0
Host: webarch.noahdemo.com
User-Agent: Noahs Demo HttpClient v1.0
Accept: */*
Accept-language: en-us
The client sends an HTTP GET

- URI is `http://webarch.noahdemo.com/demo1/test.html`
- Note that HTTP is a text-based protocol
- HTTP GET
- GET `/demo1/test.html` HTTP/1.0
  - Host: `webarch.noahdemo.com`
  - User-Agent: Noahs Demo HttpClient v1.0
  - Accept: */*
  - Accept-language: en-us
- ...but it can carry binary entity bodies such as image/jpeg

`demo1/test.html`
`http://webarch.noahdemo.com`
The server sends an HTTP Response

HTTP/1.1 200 OK
Date: Tue, 28 Aug 2007 01:49:33 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html

<html>
<head>
<title>Demo #1</title>
</head>
<body>
<h1>A very simple Web page</h1>
</body>
</html>

HTTP Status Code 200 Means Success!
The server sends an HTTP Response

HTTP/1.1 200 OK
Date: Tue, 28 Aug 2007 01:49:33 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html

<html>
<head>
<title>Demo #1</title>
</head>
<body>
<h1>A very simple Web page</h1>
</body>
</html>

The “representation” returned is an HTML document
The server sends an HTTP Response:

HTTP/1.1 200 OK
Date: Tue, 28 Aug 2007 01:49:33 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html

<html>
<head>
<title>Demo #1</title>
</head>
<body>
<h1>A very simple Web page</h1>
</body>
</html>
HTTP Methods, Headers & Extensibility
HTTP Requests always have a method

URI is http://webarch.noahdemo.com/demo1/test.html

GET /demo1.html HTTP/1.0
Host: webarch.noahdemo.com
User-Agent: Noahs Demo HttpClient v1.0
Accept: */*
Accept-language: en-us
HTTP Headers

GET /demo1.html HTTP/1.0
Host: webarch.noahdemo.com
User-Agent: Noahs Demo HttpClient v1.0
Accept: */*
Accept-language: en-us

Response headers
HTTP/1.1 200 OK
Date: Tue, 28 Aug 2007 01:49:33 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html

<!DOCTYPE html>
<html>
<head>
<title>Demo #1</title>
</head>
<body>
<h1>A very simple Web page</h1>
</body>
</html>
HTTP Extension Points

- **HTTP**
  - New *headers* *(quite common)*
  - New *methods* *(rare)*
  - New versions of HTTP itself *(e.g. HTTP 1.1)* *(very rare)*
  - New status codes *(very rare)*

- **Technologies used by HTTP**
  - Media types *(text/html, image/jpec, application/...something new?...)*
  - Languages *(e.g. en-us)*
  - Transfer encodings *(e.g. gzip)*
HTTP
Protocol for a Discoverable Web
HTTP: the protocol for a discoverable Web

- HTTP provides a *uniform interface* to all resources
- You don’t have to know in advance what a resource is like to access it with HTTP...so, you or your software can explore the Web in ad-hoc ways
- **Self-describing:** when a response comes back, you can figure out what it means
- Using HTTP to access a resource doesn’t damage the resource
- For all this to work, you must use HTTP...and you must use it properly!
HTTP Requests always have a method

URI is http://webarch.noahdemo.com/demo1/test.html

GET /demo1.html HTTP/1.0
Host: webarch.noahdemo.com
User-Agent: Noahs Demo HttpClient v1.0
Accept: */*
Accept-language: en-us
## HTTP Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong></td>
<td>Retrieve a representation of the resource but <em>do not change anything!</em></td>
</tr>
<tr>
<td><strong>POST</strong></td>
<td>Update the resource or a dependent resource base on supplied representation</td>
</tr>
<tr>
<td><strong>HEAD</strong></td>
<td>Optimized GET – retrieves only headers, used for checking caches, etc.</td>
</tr>
<tr>
<td><strong>PUT</strong></td>
<td>Create or completely replace the resource base on supplied representation</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>Delete the resource identified by the URI</td>
</tr>
</tbody>
</table>

*GET is very carefully defined to make the Web scalable and discoverable.*
### HTTP Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong></td>
<td>Retrieve a representation of the resource <em>but do not change anything!</em></td>
</tr>
<tr>
<td><strong>POST</strong></td>
<td>Update the resource or a dependent resource base on supplied representation</td>
</tr>
<tr>
<td><strong>HEAD</strong></td>
<td>Optimized GET – retrieves only headers, used for checking caches, etc.</td>
</tr>
<tr>
<td><strong>PUT</strong></td>
<td>Create or completely replace the resource base on supplied representation</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>Delete the resource identified by the URI</td>
</tr>
</tbody>
</table>

GET is very carefully defined to make the Web scalable and discoverable.
HTTP GET Has A Very Particular Meaning

URI is http://webarch.noahdemo.com/demo1/test.html

It’s always safe to try a GET on any URI. In fact, that’s why search engines like Google can safely crawl the Web!

GET /demo1.html HTTP/1.0
Host: webarch.noahdemo.com
User-Agent: Noahs Demo HttpClient v1.0
Accept: */*
Accept-language: en-us
HTTP GET Has A Very Particular Meaning

It’s always safe to try a GET on any URI. In fact, that’s why search engines like Google can safely crawl the Web!

But GET is so convenient that some people use it in the wrong places...if you’re changing the state of a resource you must use PUT or POST!
Demonstration #3:
http://webarch.noahdemo.com/MisuseGet/
Using HTTP Properly

- If GET is used instead of POST, your applications may update accidentally

- If too many applications misuse GET, then it will become impossible to explore the web
  - If you click a link, you might break something…you would have to ask about each link before clicking!
  - Search crawlers couldn’t be used…there would be no Google!

Read TAG Finding
“URIs, Addressability, and the use of HTTP GET and POST”
http://www.w3.org/2001/tag/doc/whenToUseGet.html
Web Interaction: Highlights

- HTTP: a uniform way to interact with resources
- Promotes interoperability & discoverability
- Use HTTP properly: GET is special!
- HTTP has features like content negotiation and headers that let you adapt content for your users
The Self-Describing Web
The Self-Describing Web

- Everything you need to understand a response is in the response…
- …to find the instructions, read RFC 3989…
- …and follow your nose!
Follow your nose

- The Web community’s term for:
- Going step by step through a Web response and the pertinent specifications…
- …to figure out what the response *means*
- *All the specifications you need are found (transitively) from RFC 3986!*
- *Thus there is a quite rigorous sense in which the Web is browsable and explorable*

Read TAG Finding “The Self-Describing Web”

The server sends an HTTP Response

HTTP GET

HTTP RESPONSE

Host: webarch.noahdemo.com
demo1/test.html

HTTP/1.1 200 OK
Date: Tue, 28 Aug 2007 01:49:33 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html

<html>
<head>
<title>Demo #1</title>
</head>
<body>
<h1>A very simple Web page</h1>
</body>
</html>

The “representation” returned is an HTML document
Stateful and Stateless Protocols
Stateful and Stateless Protocols

- **Stateful:** server knows which step (state) has been reached
- **Stateless:**
  - Client remembers the state, sends to server each time
  - Server processes each request independently

- **Can vary with level**
  - Many systems like Web run stateless protocols (e.g. HTTP) over streams...at the packet level, TCP streams are stateful
  - HTTP itself is mostly stateless, but many HTTP requests (typically POSTs) update persistent state at the server
Advantages of stateless protocols

- Protocol usually simpler
- Server processes each request independently
- Load balancing and restart easier
- Typically easier to scale and make fault-tolerant
- Visibility: individual requests more self-describing
Advantages of stateful protocols

- Individual messages carry less data
- Server does not have to re-establish context each time
- There’s usually *some* changing state at the server at some level, except for completely static publishing systems
HTTP is Stateless (usually)

- HTTP is fundamentally stateless…response determined entirely from GET request & URI
  - (unless resource itself changes!!)

- In practice:
  - Cookies stored in browser tied to session state and login status
  - OK: determine access rights from cookie
  - NOT OK: return different content based on cookie
  - WHY?: On the Web, we identify with URIs, not cookies!

- Of course, the state of resources does change:
  - E.g. when we add something to a shopping cart
REST
REST – **REprese**n**eration**al **St**ate **T**ransfer

- **What is REST?**
  - A distributed system *architecture*
  - In practice: implemented using HTTP

- **Key features**
  - States *transferred* using *representations* (HTTP requests/responses)
  - Server is stateless
  - URIs (and request data if provided) convey state to application

- **History**
  - Described by Roy Fielding in his PhD thesis
  - Came after Tim BL invented the Web, but…
  - …Roy was very influential in development of formal specifications of and details of HTTP and URIs

Building a REST application

- All data communicated using HTTP GET / POST
  - Unit of interaction is document
  - Formats like JSON, XML, RDF, N3 used to convey data

- State of navigation typically captured in URIs:
  - URI to start application
  - URI for 2nd step
  - URI for 3rd step
  - Captures client parameters & input in URIs (e.g. search parms, flight numbers)

- States of a REST application are linkable, bookmarkable

- Examples:
  - http://webarch.noahdemo.com/MisuseGet/
  - Google search
  - Google maps (AJAX example: uses REST recursively)
A Complex REST Application
Google Maps

Warning: some details of Google’s implementation may have changed since this presentation was prepared
AJAX Application – Google Maps

Images retrieved in segments using ordinary Web HTTP Requests
AJAX Application – Google Maps

JavaScript at client tracks mouse and moves images for smooth panning… asynchronously requests new image tiles in background
AJAX Application – Google Maps

The Web is used to retrieve an ordinary file listing points of interest…. (used to be XML but could use JSON, other?)

```xml
<?xml version="1.0" ?>
<page>
  <title>hotels in hawthorne</title>
  <query>pizza in atlanta</query>
  <center lat="33.748888" lng="-84.388056" />
  <info>
    <title>Wellesley Inn</title>
    <address>
      <line>540 Saw Mill River Rd.</line>
      <line>Elmsford, NY 10523</line>
    </address>
  </info>
</page>
```
**AJAX Application – Google Maps**

...Data used to drive formatting of points of interest

```xml
<?xml version="1.0" ?>
<page>
    <title>hotels in hawthorne</title>
    <query>pizza in atlanta</query>
    <center lat="33.748888" lng="-84.388056" />
    <info>
        <title>Wellesley Inn</title>
        <address>
            <line>540 Saw Mill River Rd.</line>
            <line>Elmsford, NY 10523</line>
        </address>
    </info>
</page>
```
AJAX Application – Google Maps

...and XSLT in the browser converts that to HTML
AJAX Application – Google Maps
Question:
What would happen if we built the Web from RPC?

---
Revisited!
Why not use RPC for the Web?

- **No uniform interface**
  - You’d need an interface definition to follow a link
- **No global naming**
- **Most RPC data types poorly tuned for documents**
Why not use RPC for the Web?

- **No uniform interface**
  - You’d need an interface definition to follow a link
- **No global naming**
- **Most RPC data types poorly tuned for documents**
- **No safe methods**
Why not use RPC for the Web?

- No uniform interface
  - You’d need an interface definition to follow a link
- No global naming
- Most RPC data types poorly tuned for documents
- No safe methods
- No content negotiation
HTTP
Content Negotiation
Content negotiation for language

URI is http://webarch.noahdemo.com/demo1/test.html

HTTP GET

I prefer English, please!

Host: webarch.noahdemo.com

GET /demo1/test.html HTTP/1.0
Host: webarch.noahdemo.com
User-Agent: Noahs Demo HttpClient v1.0
Accept: */*
Accept-language: en-us

But: it is OK to also provide individual URIs for each language version
Content negotiation for device

URI is http://webarch.noahdemo.com/demo1/test.html

HTTP GET

GET /demo1/test.html HTTP/1.0
Host: webarch.noahdemo.com
User-Agent: Mozilla/4.0
Accept: */*
Accept-language: en-us
Content negotiation for device

URI is http://webarch.noahdemo.com/demo1/test.html

HTTP GET

Cell phone browser

GET /demo1/test.html HTTP/1.0
Host: webarch.noahdemo.com
User-Agent: Cell Phone Browser V1.1
Accept: */*
Accept-language: en-us
Device independence

The same URI works across devices...

...you can see my reservation from your cell phone

Even phone numbers are on the Web
<a href="tel:18005551212">
Summary
Summary

- HTTP and the REST model are designed to meet the unique needs of the Web
- Document-oriented
- Discoverable
- Stateless
- Extensible
- Global scale
- Supports unique features like content negotiation