

Web Alphabet Soup



The Big Picture

- How desktop UIs work (input / output / arch)
- How human abilities affect UI design

- The Web

- Key ideas and technologies
- Some emerging standards
- A little bit of history
- Philosophy of web architecture
- Rapid prototyping
- Mobile web

Today



Outline

- The Web Today
 - HTTP, HTML, URL
 - XML
 - DOM, SAX
 - CSS, JavaScript
 - SOAP, WSDL, UDDI
 - AJAX, others



HTTP HTML URLs
(in bed)

Hypertext Transfer Protocol (HTTP)

- Standard way of transferring content
- Often done on top of TCP, but doesn't have to be
 - For example, could have HTTP via Bluetooth
 - Just requires reliable transport of data

HTTP Request Example

GET / HTTP/1.1

Host: uma.pt

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.13) Gecko/20060414

Accept:

text/xml,application/xml,application/xhtml+xml, text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

HTTP Request Example

GET / HTTP/1.1

Host: uma.pt

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.13) Gecko/20060414

Accept:
text/xml,
text/html;q=0.9

Accept-Language: en

Accept-Encoding: gzip, deflate

Accept-Charset: utf-8, iso-8859-1;q=0.5

Keep-Alive: 300

Connection: close

- Request to <http://uma.pt>
- Text-based protocol (vs binary RPC)
- First line is the request, everything else is a header

HTTP Request Example

GET / HTTP/1.1

Host: uma.pt

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.13) Gecko/20060414

Accept:
text/xml,
text/html;q

Accept-La

Accept-En

Accept-Ch

Keep-Alive

Connection

- HTTP defines some standard methods
 - GET, POST, DELETE
 - HEAD, PUT, TRACE
- / here is what web page to get
- HTTP/1.1 is the version

HTTP Request Example

GET / HTTP/1.1

Host: uma.pt

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.13) Gecko/20060414

Accept: text/xml,
text/html;q=0.9

Accept-Language: en

Accept-Encoding: gzip, deflate

Accept-Charset: utf-8, iso-8859-1;q=0.5

Keep-Alive: 300

Connection: close

- “Host” says what web server name
 - Added to HTTP1.1 for virtual hosting

HTTP Request Example

GET / HTTP/1.1

Host: uma.pt

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.13) Gecko/20060414

Accept: text/xml,
text/html;q=0.9

Accept-Language: en

Accept-Encoding: gzip, deflate

Accept-Charset: utf-8, iso-8859-1;q=0.5

Keep-Alive: 300

Connection: close

- “User-Agent” is what web browser
- Note that it is trivial to lie here

HTTP Request Example

GET / HTTP/1.1

Host: uma.pt

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.13) Gecko/20060414

Accept:

text/xml,application/xml,application/xhtml+xml, text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

Accept-Language: en-us,en;q=0.5

Accept-En

Accept-Ch

Keep-Alive

Connection

- “Accept” is what MIME type your browser can handle
 - q-values specify preference
 - text/html, image/gif

HTTP Request Example

GET / HTTP/1.1

Host: uma.pt

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.13) Gecko/20060414

Accept:

text/xml,application/xml,application/xhtml+xml, text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

HTTP Request Example

GET / HTTP/1.1

Host: uma

User-Agent:

5.1; en-U

Accept:

text/xml,

xt/html;q

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

- “Keep-Alive” and “Connection” are optimizations for HTTP1.1

HTTP Request

- Other headers:
 - Referer (sic): what web page you were on
 - Cookies: maintaining state between transactions
 - If-Modified-Since: conditional get
- Easy to view these
- Easy for a proxy to modify headers as well
 - For example, hiding browser type

HTTP Response Example

HTTP/1.x 200 OK

Date: Sun, 01 Oct 2006 22:45:16 GMT

**Server: Apache/1.3.34 (Unix) mod_pubcookie/3.1.1
pre-beta1 (CMU-027) mod_ssl/2.8.25
OpenSSL/0.9.6m**

Last-Modified: Thu, 21 Nov 2002 15:28:50 GMT

Etag: "385dd9-6ffb-3ddcfbb2"

Accept-Ranges: bytes

Content-Length: 28667

Keep-Alive: timeout=5

Connection: Keep-Alive

Content-Type: image/gif

HTTP Response Example

HTTP/1.x 200 OK

Date: Sun, 01 Oct 2006 22:45:16 GMT

Server: Apache/1.3.34 (Unix) mod_pubcookie/3.1.1
pre-beta1 (CMU-027) mod_ssl/2.8.25
OpenSSL/0.9.6m

Last-Modified:

Etag: "385"

Accept-Ranges:

Content-Length:

Keep-Alive:

Connection:

Content-Type:

- Version, response code, message
- Response code 200 is “OK”
- Response code 404 is “not found”

HTTP Response Example

HTTP/1.x 200 OK

Date: Sun, 01 Oct 2006 22:45:16 GMT

**Server: Apache/1.3.34 (Unix) mod_pubcookie/3.1.1
pre-beta1 (CMU-027) mod_ssl/2.8.25
OpenSSL/0.9.6m**

Last-Modified:

Etag: "385"

Accept-Ranges:

Content-Length:

Keep-Alive:

Connection:

Content-Type:

- What server is being used

HTTP Response Example

HTTP/1.x 200

Date: Sun, 10 Jun 2007 12:00:00 GMT

Server: Apache/2.2.2 (Ubuntu)
pre-beta
OpenSSL/0.9.8a

Last-Modified: Sun, 10 Jun 2007 12:00:00 GMT

Etag: "385"

Accept-Ranges: bytes

Content-Length: 28667

Keep-Alive: timeout=5

Connection: Keep-Alive

Content-Type: image/gif

- How much content is being sent
- What is the MIME type

HTTP Comments

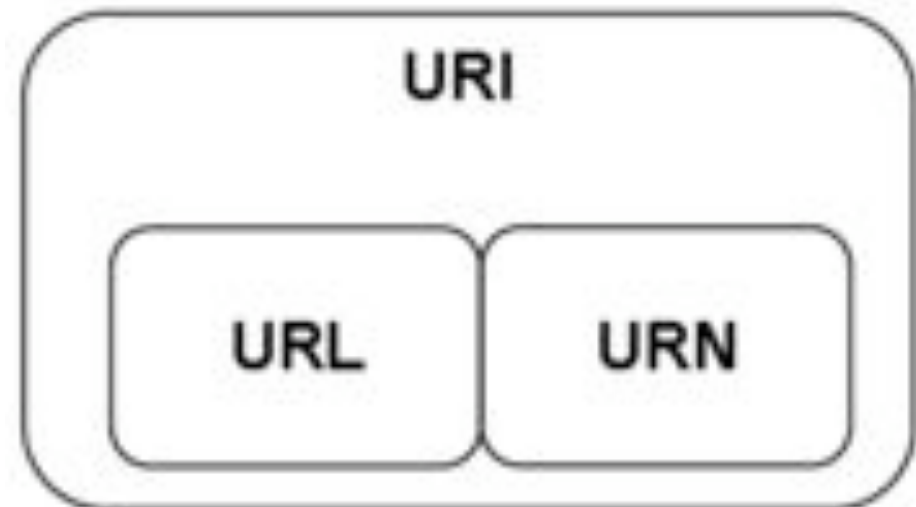
- Simple stateless protocol
 - Text-based, easy to inspect, easy for proxies
 - Few well-defined methods that can be used for resources
 - GET, POST
- Common case of just getting data is relatively basic
- Gets thornier with HTTP 1.1 though
 - Lots of hacks to make it work with persistent connections

Hypertext Markup Language (HTML)

- Standard way of representing content
 - Original version mixed content with presentation
 - Ex. Bold, Italics, Font
 - Older versions relied heavily on tables for layout
 - Now, cleaner separation with style sheets
- More about HTML and CSS in a second

Uniform Resource Locator (URL)

- URL is a standard way of addressing content
 - URL is a URI (Uniform Resource Identifier)
- URN is a standard way of naming content
 - `mailto:java-net@java.sun.com`
 - `news:comp.lang.java`
 - `isbn:096139210x`



Outline

- The Web Today
 - HTTP, HTML, URL
 - XML
 - DOM, SAX
 - CSS, JavaScript
 - SOAP, WSDL, UDDI
 - AJAX, others



XML

- Stands for Extensible Markup Language
 - Intended to facilitate data exchange
 - Rather than having many ways of representing and parsing data, create one standard extensible way
- Lots of hype, but industry adoption is strong
 - Odds are high you will have to deal with XML

XML

- Basic idea: Like HTML, but more structured and with definable tags

```
<?xml version="1.0"? encoding="UTF-8" >
<person>
  <name type="full">John Doe</name>
  <tel type="home">412-555-4444</tel>
  <tel type="work">412-268-5555</tel>
  <email>johndoe@anon.net</email>
</person>
```

XML Structure

- XML documents have no semantics
 - Up to the programs using them
 - Often defined by a committee
- Structure determined by schema
 - Fancy way of saying what tags and values are allowed
- Schema #1 – Document Type Definition
 - Currently a standard, sort of messy
- Schema #2 – XML Schema
 - Currently a recommendation, extremely complex

XML Notes

- XML absolutely must be properly formatted
 - Not sloppy like HTML
- XHTML is XML-formatted HTML
 - Currently recommended
- Subjective opinion about XML
 - Basic idea of XML is good
 - But doesn't make easy things easy
 - Quickly blowing out of control with too many changes
 - XPath, XQuery, XPointer, XSL, XSLT, etc
 - Didn't consult experts in databases

Outline

- The Web Today
 - HTTP, HTML, URL
 - XML
 - DOM, SAX
 - CSS, JavaScript
 - SOAP, WSDL, UDDI
 - AJAX, others



Document Object Model (DOM)

- A way of representing parsed HTML (or XML)
- Mostly analogous to interactor tree
 - Tree of HTML elements
 - Each element has multiple properties
- One easy way to modify web page is to modify DOM
 - Find the right element, modify it
 - Add an element
 - Remove an element

DOM Inspector

File Edit Search View Tools Window Help

<http://onu.edu/> Inspect

Document - DOM Nodes

nodeName
BODY
#text
NOSCRIPT
#text
#comment
#text
CENTER
#text
#comment
#text
FORM
#text
<input type="text"/>
#text
#comment
#text
#comment
#text

Object - Javascript Object

Property	Value
target	[object HTMLInputElement]
nodeName	"INPUT"
nodeValue	(null)
nodeType	1
parentNode	[object HTMLFormElement]
childNodes	[object NodeList]
firstChild	(null)
lastChild	(null)
previousSibling	[object Text]
nextSibling	[object Text]
attributes	[object NamedNodeMap]
ownerDocument	[object HTMLDocument]
insertBefore	function insertBefore() { [native code] }
replaceChild	function replaceChild() { [native code] }
removeChild	function removeChild() { [native code] }
appendChild	function appendChild() { [native code] }
hasChildNodes	function hasChildNodes() { [native code] }
cloneNode	function cloneNode() { [native code] }

Browser

Document Object Model (DOM)

- Many programming languages have DOM support
 - Ex. Java is `org.w3c.dom.*`
- JavaScript makes it trivial to access DOM
 - Built-in variable called `document`
 - Ex. `document.body`, or `document.head`
 - Built-in methods
 - `document.getElementById()`
 - `document.getElementsByName()`
 - `document.getElementsByTagName()`

JavaScript and DOM

```
<div id="d1" name="fname">Div #1</div>  
<div id="d2" name="fname">Div #2</div>  
<div id="d3">Div #3</div>
```

- `document.getElementById("d1")`
- `document.getElementsByName("fname")`
- `document.getElementsByTagName("div")`

Document Object Model (DOM)

- Unfortunately, DOM is not 100% cross-platform
 - Different browsers and software packages have different levels of DOM compliance, bugs
- DOM also requires you to parse and load entire web page into memory
 - May be overkill for simple things
 - Browser has to do this anyway though

Simple API for XML (SAX)

- Another way of processing web pages (and XML)
- Basic idea:
 - Instantiate a SAX Parser
 - Give it a handler
 - SAX parser incrementally parses web page / xml one element at a time
 - SAX parser makes callback to handler when appropriate

Java Example of SAX

- `org.w3c.sax.*`
- `org.w3c.sax.helpers.DefaultHandler`

```
public class MyHandler
    extends DefaultHandler {

    public void startElement(...) {
    }

    public void endElement (...) {
    }

    public void characters(...) {
    }

}
```

Simple API for XML (SAX)

Calls `startDocument()`

```
<html>
```

```
<body>
```

```
    Some text and <a href="">a link</a>
```

```
</body>
```

```
</html>
```

Simple API for XML (SAX)

Calls `startElement("html")`



```
<html>
```

```
<body>
```

```
    Some text and <a href="">a link</a>
```

```
</body>
```

```
</html>
```

Simple API for XML (SAX)

```
<html>
```

```
<body>
```

```
    Some text and <a href="">a link</a>
```

```
</body>
```

```
</html>
```



Calls startElement("body")

Simple API for XML (SAX)

```
<html>
```

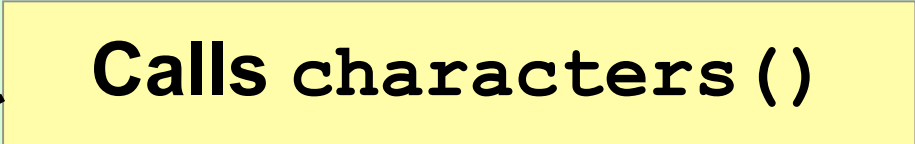
```
<body>
```

```
    Some text and <a href="">a link</a>
```

```
</body>
```

```
</html>
```

Calls characters ()



Simple API for XML (SAX)

```
<html>
```


```
<body>
```

```
    Some text and <a href="">a link</a>
```

```
</body>
```

```
</html>
```

Calls `startElement("a")`



Simple API for XML (SAX)

```
<html>
```


```
<body>
```

```
    Some text and <a href="">a link</a>
```

```
</body>
```

```
</html>
```

Calls characters ()



Simple API for XML (SAX)

```
<html>
```

```
<body>
```

```
    Some text and <a href="">a link</a>
```

```
</body>
```

```
</html>
```

Calls endElement("a")



Simple API for XML (SAX)

- SAX not built into JavaScript
- SAX has small memory footprint, in theory faster
- SAX API highly un-object-oriented
 - Passes lots of character arrays
 - Meant to be a uniform API across multiple programming languages
- When to use?
 - Simple stateless transformations, ex. remove banner ads

Recap

- Two basic ways of handling HTML (and XML):
- DOM
 - Parse document into a tree
 - Manipulate tree as needed
 - More flexible, but entire document in memory
- SAX
 - As you parse, issue callbacks
 - Only small part of document in memory at any time
 - Simple and fast, but non-object-oriented API

Administrivia

- P3 Progress?
 - Use of color?
 - Mapping / layout / grouping?

Outline

- The Web Today
 - HTTP, HTML, URL
 - XML
 - DOM, SAX
 - CSS
 - SOAP, WSDL, UDDI
 - AJAX, others



Why Cascading Style Sheets?

- Developed by Håkon Lie and Bert Bos in mid 1990s
- Basic idea: separate content from presentation
- HTML not meant to support styling information
 - But browsers started supporting inline style changes (bold, italics, centered, etc)
 - Why is this a problem?
- Inline styling information is problematic
 - Difficult to change if in multiple places
 - Harder to make consistent
 - No support for different display formats (audio, mobile)
 - Bloats pages

Why Cascading Style Sheets?

- Specify content in one file (HTML)
- Specify presentation in another (CSS)

Example Use of CSS

- Can specify styles for all of tags of a type

```
p {  
    font-family: "Garamond", serif;  
}  
h2 {  
    font-size: 110%;  
    color: red;  
    background: white;  
}
```

Example Use of CSS

- Can specify styles for a class of tags

```
.note {  
    color: red;  
    background: yellow;  
    font-weight: bold;  
}
```

```
<div class="note">Div #1</div>
```

```
<div class="note">Div #2</div>
```


Example Use of CSS

- Can specify styles for a specific id

```
#homeAddress {  
    color: black;  
    font-weight: bold;  
}
```

```
<div id="homeAddress">blah blah</div>
```

CSS Notes

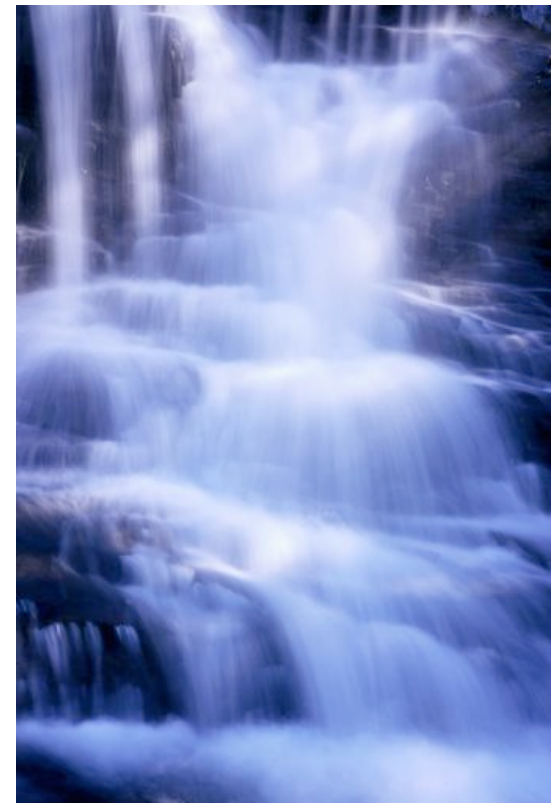
- Note: different syntax than XML
 - Would have been in XML format except didn't exist then
- Can specify how to apply a style many ways:
 - **By tag type**
 - **By class**
 - **By id**
 - By arbitrary attribute
 - By proximity
 - ...

CSS Notes

- Can provide several style sheet options
 - Give titles to each style sheet
 - One preferred (default) style, the rest are alternates
 - Ex. An online magazine
 - Screen
 - Aural, braille, embossed
 - Print
- Makes it easy to switch visual appearance

Cascading?

- Multiple styles can be combined
 - In theory, can be a CSS associated with site, page, browser, user
 - Have some way of handling conflicts
- In practice, cascading hasn't taken off
 - Usually web site specifies only style sheet



CSS Notes

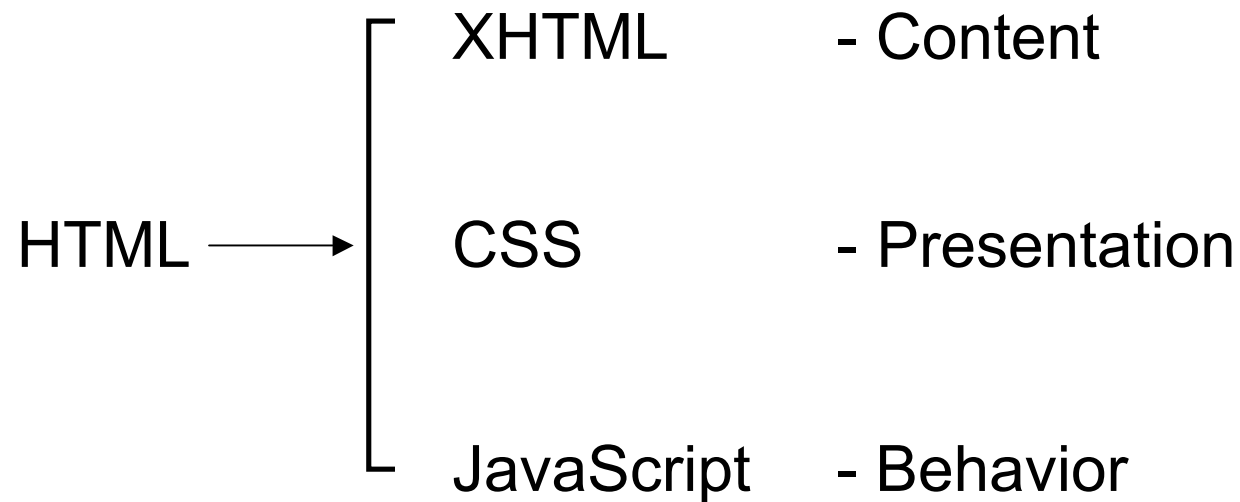
- Mostly integrated with JavaScript
 - `element.style.borderColor`
- Compatibility issues (like JavaScript)
 - Makes it a real pain to support multiple browsers
- Style in DOM reflects fully processed style sheet
 - Led to interesting history stealing hack
 - <http://jeremiahgrossman.blogspot.com/2006/08/i-know-where-youve-been.html>
 - Create a list of links to popular sites
 - Iterate through those links
 - Keep links whose color is “visited”

Cascading Style Sheets

- Good for general maintenance
 - Good principle of large-scale systems is to factor out common things
- Good for accessibility
 - Can apply an appropriate style sheet
 - Don't have to wade through tables for layout
- Good for mobile web
 - Almost everything good for accessibility is good for mobile
 - However, style sheets haven't taken off for mobile
 - Typically, re-target key parts of web site for mobile



Stepping Back, Big Picture



- Sort of like evolution of Model-View-Controller

A glowing yellow lightbulb is centered on a teal background. The text "2 Minute Break" is written in a bold, dark blue font across the middle of the lightbulb.

2 Minute Break

Outline

- The Web Today
 - HTTP, HTML, URL
 - XML
 - DOM, SAX
 - CSS
 - SOAP, WSDL, UDDI
 - AJAX, others



Web Services

- Huge standardization and interoperability effort
- Basic idea: standard formats, protocols, and programming model for accessing a service
 - Ex. Google search `code.google.com`
 - Ex. Ebay search `developer.ebay.com`
 - Ex. Amazon book search `aws.amazon.com`
- Works for all programming languages
 - Just need to send a formatted message to the right place



Google Web APIs (beta)

[Home](#)

[About Google](#)

Google Web APIs

[Overview](#)

[Download](#)

[Create Account](#)

[Getting Help](#)

[API Terms](#)

[FAQs](#)

[Reference](#)

[Release Notes](#)

Google Desktop API

Write handy plug-ins for Google Desktop Search.

Google AdWords API

Manage your accounts, build new tools, pull reports, and more.

Develop Your Own Applications Using Google

With the Google Web APIs service, software developers can query billions of web pages directly from their own computer programs. Google uses the SOAP and WSDL standards so a developer can program in his or her favorite environment - such as Java, Perl, or Visual Studio .NET.

To start writing programs using Google Web APIs:

1 Download the developer's kit

The Google Web APIs developer's kit provides documentation and example code for using the Google Web APIs service. The [download](#) includes Java and .NET programming examples and a WSDL file for writing programs on any platform that supports web services.

2 Create a Google Account

To access the Google Web APIs service, you must [create a Google Account](#) and obtain a license key. Your Google Account and license key entitle you to 1,000 automated queries per day.

3 Write your program using your license key

Your program must include your license key with each query you submit to the Google Web APIs service. Check out our [Getting Help](#) page or read the [FAQs](#) for more information.



With Google Web APIs, your computer can do the searching for you.

Program Ideas

- ▶ Auto-monitor the web for new information on a subject
- ▶ Glean market research insights and trends over time
- ▶ Invent a catchy online game
- ▶ Create a novel UI for searching
- ▶ Add Google's spell-checking to an application

<http://code.google.com/apis/soapsearch/>

Web Services Acronyms

- SOAP
- WSDL
- UDDI

Simple Object Access Protocol (SOAP)

- Protocol for message exchange
 - Specifies how to call a remote service
 - Can pass some XML objects
 - Can return some XML objects
 - RPC over HTTP using XML
- Similar to Java RMI or CORBA RMI



“Simple” SOAP Request

```
<soap:Envelope xmlns:soap =  
"http://schemas.xmlsoap.org/soap/envelope/">
```

```
  <soap:Body>  
    <getProdDetails xmlns =  
      "http://warehouse.example.com/ws">  
      <productId>827635</productId>  
    </getProdDetails>  
  </soap:Body>
```

```
</soap:Envelope>
```

“Simple” SOAP Response

```
<soap:Envelope xmlns:soap="http://... ">
<soap:Body>
<getProdDetailsResponse xmlns =
"http://warehouse.example.com/ws">
  <getProductDetailsResult>
    <productName>Toptimate 3-Piece Set
    </productName>
    <productId>827635</productId>
    <description>
      3-Piece luggage set Polyester
    </description>
    <price>96.50</price>
    <inStock>true</inStock>
  </getProductDetailsResult>
</getProductDetailsResponse>
</soap:Body>
</soap:Envelope>
```

“Simple” SOAP Response

```
<getProductDetailsResult>
```

```
<productName>
```

```
    Toptimate 3-Piece Set
```

```
</productName>
```

```
<productId>827635</productId>
```

```
<description>
```

```
    3-Piece luggage set Polyester
```

```
</description>
```

```
<price>96.50</price>
```

```
<inStock>true</inStock>
```

```
</getProductDetailsResult>
```


SOAP Notes

- Somewhat messy and confusing, but good tools will handle ugly details for you transparently



Web Services Description Language (WSDL)

- **Problem:** How to know what services a web site offers? And what data to send via SOAP?
- **Solution:** Describe web service's API using WSDL
 - Ex. in Java, we have:
 - `Interfaces for specifying APIs`
 - `public int getValue(char ch)`
 - WSDL does the equivalent

Web Services Description Language (WSDL)

- WSDL document describes:
 - Data formats
 - Valid messages
 - **Ports**, a collection of operations you can do (like library)
 - Each port has a **binding**
 - Binding is how to contact (ex. http)
 - Binding is how to format messages (what encoding)

Web Services Description Language (WSDL)

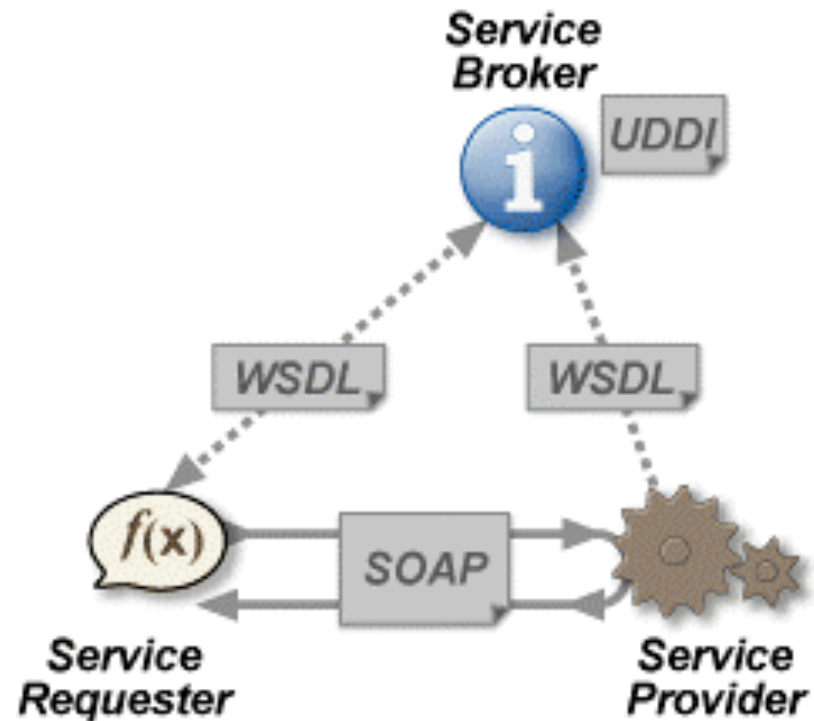
- Example:
 - `http://api.google.com/GoogleSearch.wsdl`
- Like SOAP, somewhat messy and confusing, but good tools handle these for you

Universal Description, Discovery, & Integration (UDDI)

- **Problem:** How to find WSDLs?
- **Solution:** A universal registry of WSDL services
- Let businesses list services they provide
 - White pages – real address and contact information
 - Yellow pages – industrial categorization
 - Green pages – technical information on exposed services
- Originally envisioned to be one global UDDI registry
 - Could dynamically bind to, ex. credit card authentication
 - In practice, UDDI for a company
 - Much easier integration

Web Services Recap

- SOAP
 - Protocol for exchanging messages (request / response)
- WSDL
 - A description of the web service's API
- UDDI
 - How to find available web services



Will Web Services Take Off?

- Provides an explicit way of opening up a system
 - Consider alternative, screen-scraping Amazon prices
- Strong interoperability
- Early buy-in from Google, EBay, Amazon, others

- Different APIs for different services
 - Hard to switch (since APIs aren't uniform)
- Barrier to entry pretty high

- Conjecture: very likely to be adopted, but only incremental improvement on top of existing web

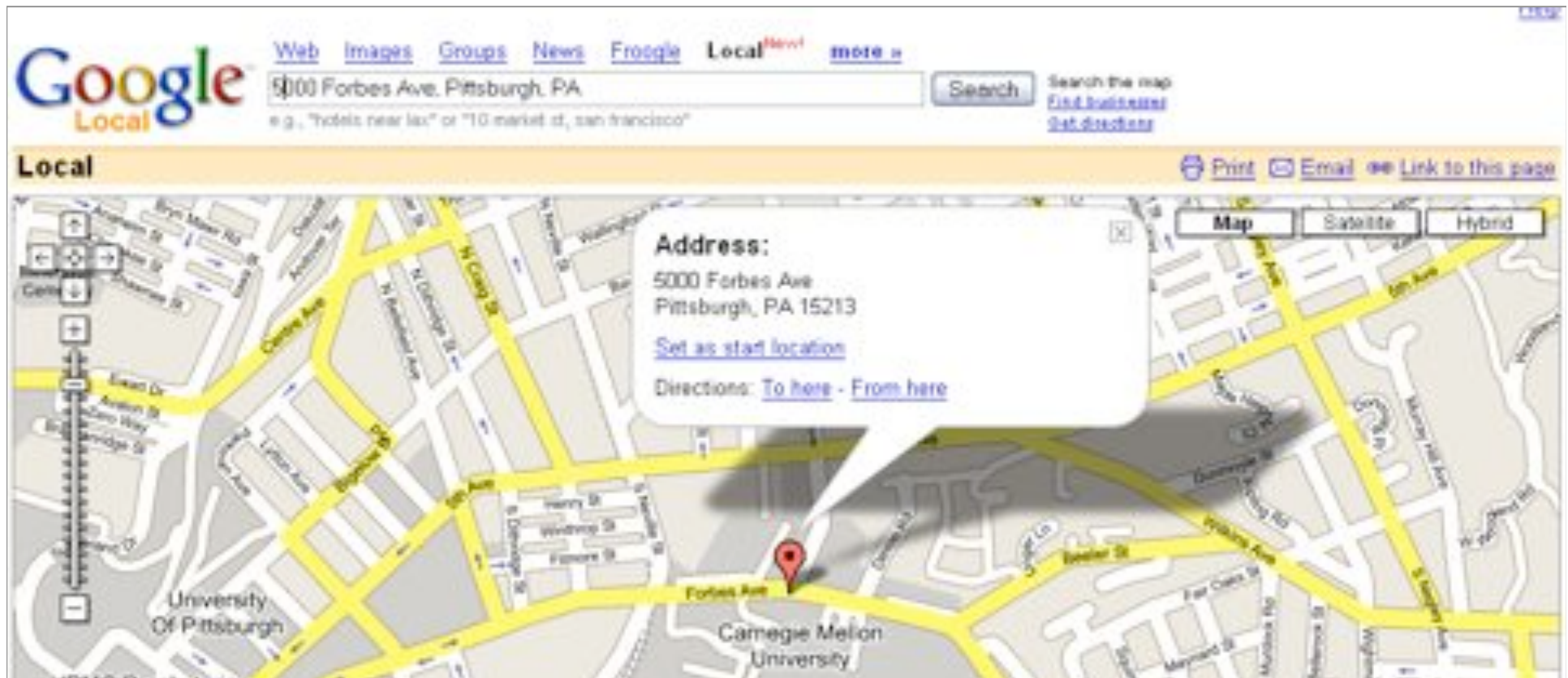
Outline

- The Web Today
 - HTTP, HTML, URL
 - XML
 - DOM, SAX
 - CSS, JavaScript
 - SOAP, WSDL, UDDI
 - AJAX, others



AJAX

- Asynchronous JavaScript and XML
 - Highly interactive web apps
 - Examples: Google Maps



Rich Internet Applications (RIA)

- Yet another buzzword without a proper definition
 - Basically, make web apps more like desktop GUI
 - Many would consider AJAX part of this
 - Might include Macromedia Flash, Java applets, ActiveX
 - More on pros and cons next time
- Will stick with RIA since not tied to specific technology

Two Kinds of Rich Internet Applications

- Two basic forms (not mutually exclusive)
- #1 – Rich interactions by cleverly manipulating DOM
 - Ex. <http://www.openrico.org/rico/demos.page>
 - Ex. <http://dojotoolkit.org>



Rich Internet Applications (RIA)

- Two basic forms (not mutually exclusive)
- #2 – Update content without explicitly reloading
 - `XMLHttpRequest` object in JavaScript
 - Can asynchronously request information from a server
 - Can load that information into page on-demand (DOM)

XMLHttpRequest Example

```
var req;
```

```
function xmlOpen(method, url,  
                 toSend, responseHandler) {  
    req = new XMLHttpRequest();  
    req.onreadystatechange = responseHandler;  
    req.open(method, url, true);  
    req.setRequestHeader("content-type",  
        "application/x-www-form-urlencoded");  
    req.send(toSend);  
}
```

```
function myResponseHandler() {  
    if (req.readyState == 4) {  
        // do something with req.responseText  
    }  
}
```

Networked RIAs

- Pros
 - Smoother and more fluid interaction
 - Don't have to wait for entire page to reload
 - Easier to maintain things in context
- Some Issues
 - How to bookmark?
 - Broken back button?
 - Easier to spy on people
 - Breaks user model of when data gets transferred (forms)
 - How to handle server load? Network latency?

Summary

- The Web Today
 - HTTP, HTML, URL
 - XML
 - DOM, SAX
 - CSS, JavaScript
 - SOAP, WSDL, UDDI
 - AJAX, others
- Reading for next time
 - Printed and at beginning of class
 - Principled design of the modern Web architecture, by Fielding and Taylor, ICSE2000
<http://doi.acm.org/10.1145/337180.337228>
 - Accessible via ACM Digital Library

Extra slides

The Web Tomorrow? – Summary

- Future very hazy —●
- Lots of opportunities
 - Semantic web
 - AJAX / RIAs
 - Customization of web pages



GreaseMonkey

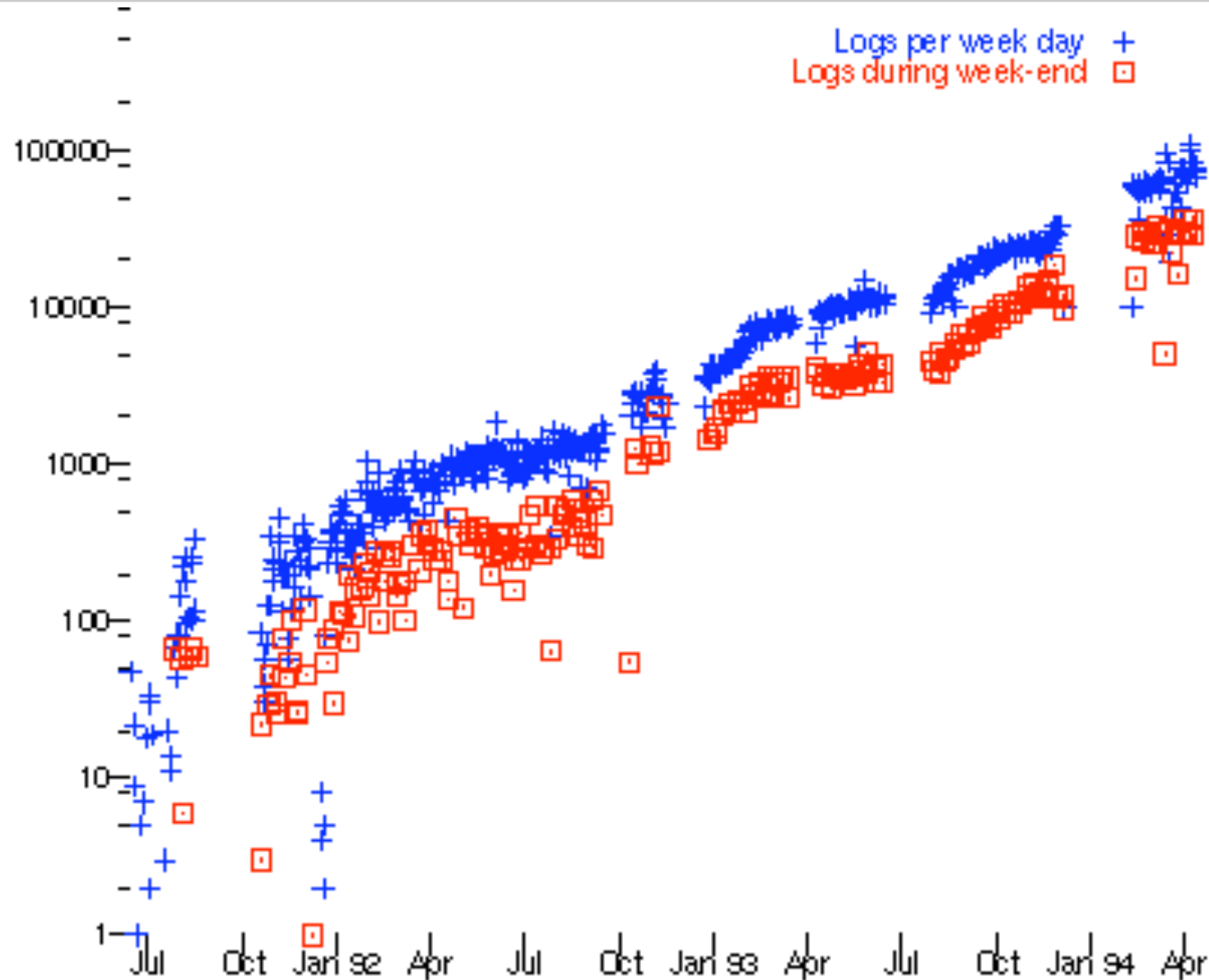
- Customize web presentation on the client-side
- Examples:
 - Make all URLs clickable links
 - For every book on Amazon, show if your local library has it
 - Add a link to Google Maps views to see the nearest geocaches to the current center of the map
- Basic Idea:
 - Site-specific JavaScript which manipulates the contents of a webpage via the DOM

ChickenFoot







- Research Project at MIT
- Same basic idea as GreaseMonkey, with easier end-user programming
 - No JavaScript
- Easier page manipulation
 - `go("google.com")`
 - `enter("sau")`
 - `click("google search")`
 - `pick('gif')`
 - `isbn = find(new TC('number just after isbn'))`



Early Web Growth



How is CSS applied?

-  Source document is parsed into a DOM tree
-  Media type is identified
-  Relevant stylesheets obtained
-  DOM tree annotated with values to every property
-  Formatting structure generated
-  Formatting structure presented (rendered)

JavaScript / ECMAScript

- Most common scripting language
 - Originally supported by Netscape, eventually by IE
- Typically embedded in HTML page
 - Executable computer code within the HTML content
 - Interpreted at runtime on the client side
- Can be used to dynamically manipulate HTML
 - Has access to DOM
 - Can react to events (ex. `onmouseover`)
 - Can be used to dynamically place data in the first place
 - Often used to validate form data

JavaScript Syntax

- Code written within `<script>` element
 - e.g., `<script type="text/javascript">`
`document.write("Hello World!")`
`</script>`
 - Use `src` attribute for scripts in external files
 - `<script type="text/javascript"`
`src="http://.../global.js">`

JavaScript Syntax

- HTML Elements have script-specific event attributes
 - e.g., `<body onmousedown="whichButton() ">`
 - e.g., `<input type="button" onclick="uncheck() ">`
- Three important things:
 - JavaScript supported on nearly all browsers
 - Direct access to the HTML DOM model
 - Makes this the language of choice for client-side web
 - XMLHttpRequest feature
 - Basis of Google Maps and other similar apps

Semantic Web

- Put everything in a machine-understandable format
 - Eliminate screen-scraping
 - Would allow for faster and more effective searches
 - Would allow us to apply logical operators to data
- Basic Idea: Artificial Intelligence meets web and XML
 - RDF (Resource Description Framework)
 - OWL (Web Ontology Language)

Semantic Web

- RDF (Resource Description Framework)
 - Metadata describing content
 - Ex. Author, homepage, price, etc
 - Primarily a standardization effort

Semantic Web

```
<?xml version="1.0"?>
```

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-  
  rdf-syntax-ns#"  
  xmlns:cd="http://www.recshop.fake/cd#">
```

```
  <rdf:Description rdf:about=  
    "http://www.recshop.fake/cd/Empire Burlesque">  
    <cd:artist>Bob Dylan</cd:artist>  
    <cd:country>USA</cd:country>  
    <cd:company>Columbia</cd:company>  
    <cd:price>10.90</cd:price>  
    <cd:year>1985</cd:year>  
  </rdf:Description>
```

```
</rdf:RDF>
```

Semantic Web

- OWL (Web Ontology Language)
 - Lets you formally specify a knowledge domain
 - Richer (and more complex!) vocabulary for describing properties and classes
 - Basically first-order predicate logic
 - Ex. Uncle is brother-of parent
 - Other example features:
 - Disjointness
 - Cardinality (“exactly one”)
 - Equality

Semantic Web

- Some Problems:
 - Primarily academic, industry doesn't seem interested
 - Industry effort more on web services right now
 - Huge amount of effort with (thus far) little benefit
 - Example problems don't seem interesting
 - Friend of a Friend
 - Not clear how many people need to buy-in before semantic web gets useful