From Web Documents to Web Applications

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

- Explore the differences between static documents and “live” applications
- Learn the *Rule of Least Power*
- Learn how AJAX applications:
  - Increased the power of the Web
  - Stretched the architecture of the traditional document-oriented Web
Sharing Documents, Data, & Applications
Sharing documents, data & applications

- Since early days of computing: all have been shared

- Programs & data/documents on:
  - Punched cards, paper tape, magnetic tape
  - Floppy disks, CD ROMS, thumb drives, etc.

- Networking early days:
  - Sharing via FTP and e-mail

- Model
  - Copy the data/documents
  - Manually install the programs
What makes applications different?

- **Documents don’t do anything**

- **Programs can:**
  - Perform arbitrary computations for you
  - Alter your machine
  - Run for a long time, with multiple interactions
  - Masquerade as other programs (spoofing)
  - Do much more (open network connections, etc.)!

- **The Web:**
  - Allowed instantaneous “activation” of documents
  - Now doing the same for applications
What makes applications different?

- **Documents don’t do anything**

- **Programs can:**
  - Perform arbitrary computations for you
  - Alter your machine
  - Run for a long time, with multiple interactions
  - Masquerade as other programs (spoofing)
  - Do much more (open network connections, etc.)!

- **The Web:**
  - Allowed instantaneous “activation” of documents
  - Now doing the same for applications
  - *A new model: applications built from smart documents*
JavaScript & Web Pages
Javascript

- A general purpose programming language
- *Runs on Web pages*
- Turing complete – this is a big deal!
  - Check out the amazing: [http://bellard.org/jslinux/](http://bellard.org/jslinux/)
  - *Remember, that’s running entirely in a Web page in your browser!*
- **Proving the power of JavaScript**
  - The JavaScript is emulating an Intel CPU and controller chip *at the instruction level*
  - A PC-compatible Linux is booted on that!

See tech notes at: [http://bellard.org/jslinux/tech.html](http://bellard.org/jslinux/tech.html)
JavaScript is a *scripting language*. Even a single statement can be used in some cases.

```html
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Demo #1</title>
</head>
<body>
<h1>JavaScript Demo</h1>
<p>Try moving your mouse over the word below:</p>
<p id="hellopara" style="font-size:large"
    onmouseover='this.style.color="red";'
    onmouseout='this.style.color="black";'>Hello!</p>
</body>
</html>
```

The ability to write single statement programs tends to be a distinguishing characteristic of scripting languages.
JavaScript is *integrated with the Web page*

```html
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Demo #1</title>
</head>
<body>
<h1>JavaScript Demo</h1>
<p>Try moving your mouse over the word below:</p>
<p id="hellopara" style="font-size:large"
    onmouseover='this.style.color="red";'
    onmouseout='this.style.color="black";'>Hello!</p>
</body>
</html>
```

What is "this"?
JavaScript is *integrated with the Web page*

```html
<html>
<head>
<script type="text/javascript">
function makeColor(elem,color) {
   elem.style.color=color;
}
function loadfunction() {
   helloPara = document.getElementById('hellopara');
   helloPara.onmouseover = function() {
      makeColor(this,"red");
   };
   helloPara.onmouseout = function() {
      makeColor(this,"black");
   };
}
</script>
<title>JavaScript Demo #2</title>
</head>
<body onload='loadfunction()'>
<h1>JavaScript Demo #2</h1>
<p>Try moving your mouse over the word below:</p>
<p id="hellopara" style=font-size:large>Hello!</p>
</body>
</html>
```

We can dynamically find elements by their id’s!
Uses for JavaScript on the Web

- **Making Web pages smarter**
  - Form field input checking
  - Dynamic highlighting & selection
  - *In this case, we still have a document*

- **Using HTML as an application container**
  - The user interacts with an application, not a document
  - Example: the Linux system we just saw
  - GMail, Yahoo Mail

- **AJAX**
  - Building smart documents *and* applications by *integrating content from multiple sources dynamically at the client*
  - The original mashup: [http://www.housingmaps.com/](http://www.housingmaps.com/) (no longer active 😞)

- **HTML Games and Animations**
Summary: Integrating JavaScript with Web Pages

- Web applications are HTML pages w/JavaScript smarts

- HTML serves as the *display list* for the application
  - JavaScript can manipulate the HTML tree & CSS at anytime
  - *Page is dynamically re-rendered*

- History footnote:
  - 1997 or so: Netscape had invented JavaScript (which has nothing to do with Java, but was named to sound like it did…)
  - Microsoft was behind in the “browser wars”, but…
  - …Microsoft had in an internal product a rendering engine that can incrementally re-render a page when parts of a document change
  - They repurposed that engine to make Internet Explorer’s dynamic HTML rendering better than Netscape’s
  - Result: Internet Explorer became the leading browser (until Firefox, based on a reworked Netscape engine, became an attractive open-source alternative)
Mashups 101
Warning!
This explanation was researched a few years ago. Details of Google Maps may have changed since.

Google Maps + craigslist = www.housingmaps.com
How Google Maps Work

Images retrieved in segments using ordinary Web HTTP Requests
How Google Maps Work

JavaScript at client tracks mouse and moves images for smooth panning... asynchronously requests new image tiles in background
How Google Maps Work

The Web is used to retrieve an ordinary XML file listing points of interest....

<?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>
How Google Maps Work

...and XSLT in the browser converts that to HTML

<?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>
How Google Maps Work

…and XSLT in the browser converts that to HTML
How the Mashup works

- Javascript opens a connection to Craigslist
- Craigslist information converted to Google-Maps compatible form
- Information is “injected” into Google Maps client side runtime.
How Google Maps Work

...now we’ve got real estate listings instead of restaurants

<?xml version="1.0" ?>
<page>
<title>Housing from Craigslist</title>
<info>
<title>Cute apartment</title>
<address>
  <line>101 5th Avenue.</line>
  <line>New York, NY 10001</line>
</address>
</page>
How the Mashup works

- Javascript opens a connection to Craigslist
- Craigslist information converted to Google-Maps compatible form
- Information is “injected” into Google Maps client side runtime.
Challenge
What if Everything on the Web was Javascript?
The Rule of Least Power

- Javascript is powerful, but...you can’t extract much information from JavaScript source
- If all Web content were JavaScript...we wouldn’t have Google!
- Where possible, use declarative languages
  - HTML
  - CSS
  - Etc.

Read TAG Finding “Rule of Least Power”
http://www.w3.org/2001/tag/doc/leastPower.html
Architecture Challenges: Naming
How do we name application states

- URI fragment IDs name elements in an HTML document
- What to do when the document hosts an application?
- This proved a subtle problem…read the TAG finding
- What would you want to link in that soccer game?
- One useful idiom: if your application is serving documents, give each one a URI
  - Maps, Office Documents, E-mail
- Remember: keep naming orthogonal to access control

*Read TAG Finding “Identifying Application State”*

http://www.w3.org/2001/tag/doc/IdentifyingApplicationState
Architecture Challenges: Security
Mobile Code is Powerful but Dangerous

- **Advantages**
  - Can do “anything” – build powerful applications
  - Supports very fluid interactivity
  - Zero install: we’re always running the latest version of Google Maps, Yahoo! Mail, etc.

- **Dangers:**
  - Can do “anything” – at least if security controls fail
  - Hard to check what a Turing-complete program is going to do: virus detection unreliable
  - Be especially suspicious of code that manipulates other code

Read Ken Thompson’s Turing Award Lecture “Reflections on Trusting Trust”

http://cm.bell-labs.com/who/ken/trust.html
Summary
Summary

- With JavaScript, the Web evolved
- Now: documents + applications
- Least Power: don’t use powerful languages unnecessarily
- Model applications as documents where practical
- Think hard about naming things with URIs
- Mobile code: powerful but dangerous!
- Is the Web really working the way Tim Berners-Lee planned it? (We’ll discuss that next time)