The State of Ajax

Douglas Crockford

Yahoo!
1890 Census
Mainframes
Hollerith Card
Console
Control Data 6600
Timesharing
Timesharing
Timesharing
Timesharing

• A timesharing system was a community.

• File sharing.

• Email.

• Chat.

• Blogs.
Douglas Engelbart
Personal Computing
Personal Computing
Personal Computing
Personal Computing

• Personal computing killed timesharing.

• A big step forward in affordable computing and interactivity.

• A big step backward in social computing.

• It would take 20 years for networks to fill the gap.
Welcome to HyperCard

HyperCard is a unique software tool that allows you to do more with your computer.

With HyperCard, you use documents called stacks. Stacks can help you do many different things—for example, you could use a stack to keep track of your appointments, manage your expenses, learn a new language, or record and play sounds.

To learn more, click “What is HyperCard?,” to the right.

©1993-1995 Apple Computer, Inc. All Rights Reserved.
Hypercocard

- Didn't anticipate color.
- Didn't anticipate text links.
- Didn't anticipate networking.
ARPANET became the Internet.
Information Retrieval

- FTP.
- Archie.
- Gopher.
- WAIS.
- WWW.
The Web

3270 for the Twenty-First Century
Interactivity
Interactivity

- Java was a huge failure.
- Very popular. High acceptance.
- The "Write once, run everywhere" promise not kept.
- Unworkable "blame the victim" security model.
- Tedious UI model.
- Successful as a server technology.
Interactivity

- **Netscape:**
  
  JavaScript

  Programming model similar to HyperCard
Interactivity

• Microsoft:
  
  JScript
  
  High fidelity clone of JavaScript
  
  Generalized document model
  
  All elements are scriptable
  
  XMLHttpRequest
Five Years Later

Jesse James Garrett
discovers Ajax
Ajax

• Applications without installation.
• Highly interactive.
• High social potential.
• Easy to use.
• Great network efficiency.
• But it is too damn hard to write applications.
Mashups

The most interesting innovation in software development in 20 years.
Mashups are insecure.

Mashups must not have access to any confidential information.
Security is a big issue.

The web is an exploit waiting to happen.
Ajax Assets

• HTML
• CSS
• JavaScript
• XML for data transfer.
Ajax Liabilities

• Inadequate application model.

  Ajax libraries make a significant difference.

• Insecurity.

• Offline operation.
JavaScript

- Deeply flawed, unpopular model.
- It works in an environment where Java failed.
- The web requires unusually levels of compatibility.
- Worst feature: Insecurity.
- Standards activity: Heavy, complex, more Java-like, insecure.
Genesis Demo
HTML

- Low graphical ability.
- Low interactivity.
- Missing a compositing model. Accessibility.
- Is it document format or an application-delivery format?
- Survived termination.
- Standards activity: Confusion.
CSS

• Slow.
• Complex.
• Fragile.
• Incomplete.
• Should be replaced.
• Standards Activity: Unimplementable.
XML

- Complicated.
- Inefficient.
- Model mismatch.
- It has been replaced:
- JSON: The x in Ajax.
Data Access

- XMLHttpRequest.
- Same origin policy. Useless for mashups.
- Circumvention is possible, but is unsafe.
- Standards Activity: Relaxation of same origin policy, Confusion.
- JSONRequest: www.JSON.org
Offline

- Reliability.
- Mobility.
- Google Gears. Good stuff.
- How do we get universal adoption?
Security

• The biggest single problem with the browser is its security model.

• The browser security model is inadequate to deal with the current generation of Ajax applicants.

• It blocks things that should be allowed, and it allows things that should be blocked.

• Urgent: This needs to be fixed.
If there is script from two or more sources, the application is not secure.

Period.
Web Time used to mean "really fast."


HTML 4.01: 1999.
Stages of Maturity

1. Proposal
2. Standard
3. Academic Freeplay
4. Proprietary Extention
5. There is no 5
Advancing

• Advances in standards are useless if the browser makers cannot/will not implement.

• Advancements by any browser maker are useless unless adopted by all browser makers.

• Advancements by all browser makers are useless unless all users upgrade.
The worst thing we can do to the web is destabilize it.

There is a lot at risk.
Why did it take 5 years for Ajax to happen?

It took that long to stabilize the web after the browser war.
Bugs

• Bugs in the web do not go away. A new edition of a browser may fix a bug, but not every user will upgrade.

• Every new edition creates new bugs.

• Developing and testing can become extremely difficult.

• It can take years to flush bugs out.
It could take as long as five years to flush IE6 out of the market.

- Most people upgrade their browsers by replacing their computers.
- Some companies do not allow their employees to upgrade.
- In the world market, a lot of people will be using less capable machines.
End of the Web?

• If the web is unable to repair itself, it could be replaced with a proprietary system, such as Microsoft's Silverlight or Adobe's AIR.

• There are advantages to proprietary systems: Only one source of new bugs. A much simpler upgrade story.

• We have come to love open systems.
I used to say that the browser was the most hostile programming environment every devised.

That was before I found out about mobile.
Going Mobile

• Mobile industry had its own failed experiments with proprietary systems and Java.

• Now they are going to try Ajax.

• Despite all of its problems, Ajax works.

• Mobile devices are replaced more frequently than PCs.
Mobile has an even more urgent need to solve the offline and security problems.

Perhaps mobile leads the next wave of web technology.
If personal computer-based browsers are unable to keep up with mobile, perhaps mobile will obsolete PCs.

Cellphones with 14" screens.
Moral:

Don't take any crap.
Email

• Research-grade email became the standard.
• Foreseeable problems with security.
• Plague of spam and phishing and viruses.
Insist on quality.

So far, the future has been a disappointment.
Ajax

It floats your dreams right down the drain!