Managing State for Ajax-Driven Web Components

John Ousterhout and Eric Stratmann
Stanford University
Introduction

- Problem: Ajax complicates Web applications
- Solution: reusable Ajax components that hide complexity
- Problem: components require state that spans Web requests
  
  2 possible solutions:
  - Reminders: store state on browsers
    - Security issues
  - Page properties: store state on servers
    - Garbage collection issues

Overall, better to store state on servers
Ajax Basics

- Incremental updates to existing Web pages
- Enables richer interactions:
  - Draggable maps
  - Auto-complete
  - Live feeds and updates

Initial page fetch

Browser

HTML

Ajax requests

Server

HTML, JSON, Javascript ...
Ajax Adds Complexity

- Every Ajax request must pass through controller
- Exposes controller to internals of views
- Scales poorly as pages get more complex
Goal: Use Components

- Hide complex Ajax behavior in components
- Fiz: component-based framework
  - E.g., tree component manages:
    - HTML tree layout
    - Javascript event handlers
    - Ajax calls for incremental expansion
    - Communication with external data source
- Controller provides:
  - Data source
  - Formatting (icons, etc.)

new TreeSection("FS.fileInDir", "code/Fiz");
Ajax Component with MVC

- Bypass controller during Ajax requests
  - Change URL routing rules
Problems with State

- **Bypass controller during Ajax requests**
  - Change URL routing rules

- **During Ajax requests, state from original rendering no longer available**
Reminders

- Reminder: collection of name/value pairs
- Embedded in page by server-side component
- Returned selectively in Ajax requests
- Similar to ASP.net ViewState, except granular
Reminder Evaluation

✓ Allows Ajax encapsulation: state visible only to components

✗ Additional overhead for transmitting reminders
  ✓ Reminders typically small

✗ Security implications:
  ✗ Reminders store internal server state; potentially sensitive
  ✗ Must protect integrity (MACs)
  ✗ May need encryption also
  ✗ Granularity enables mix-and-match replays

Unlikely that developers will recognize security risks
Page Properties

- **Page properties**: name-value pairs specific to a page
  - Stored in session
  - Created during initial page rendering
  - Accessible/modifiable during later Ajax requests

- **Page identifier**: 
  - Stored with page on browser
  - Included in Ajax requests to find appropriate properties
Page Property Evaluation

- Allows Ajax encapsulation
- No security issues: all state kept on server
- Overhead for saving page properties
  - Must be persisted with session data
- Garbage collection
  - When is it safe to delete old page properties?
  - Can return to old pages at any time (tabs)
  - Potential for "broken pages" (forced refresh)

How much state must be retained to keep users happy?
Trace-Driven Simulation

- Data from 30 Firefox users over 2 months (200,000 page views)
Per-Tab LRU Lists

- Unfortunately, browsers don't identify tabs/windows
- Would be a useful and simple addition
Conclusions

• Managing Web application state is hard
  ▪ State distributed between browser and server
  ▪ Servers don't maintain state between requests
  ▪ Ajax makes things even worse
    ● Finer-grain interactions
  ▪ Components make things worse
    ● Need more state to maintain modularity

• Neither reminders nor page properties ideal
  ▪ Reminders: security problems
  ▪ Page properties: garbage collection problems

• Overall, page properties are probably better

• Browsers should provide tab identifiers
Related Work

- **ViewState in Microsoft ASP.net**
  - Monolithic: all state for entire page sent with each request

- **Javascript-driven applications (e.g. Gmail):**
  - Application exists as Javascript in browser
  - Server provides data only
  - All page state kept in Javascript
  - Security issues simpler/more obvious
  - Overheads for downloading Javascript
  - Business rules must still be enforced on the server