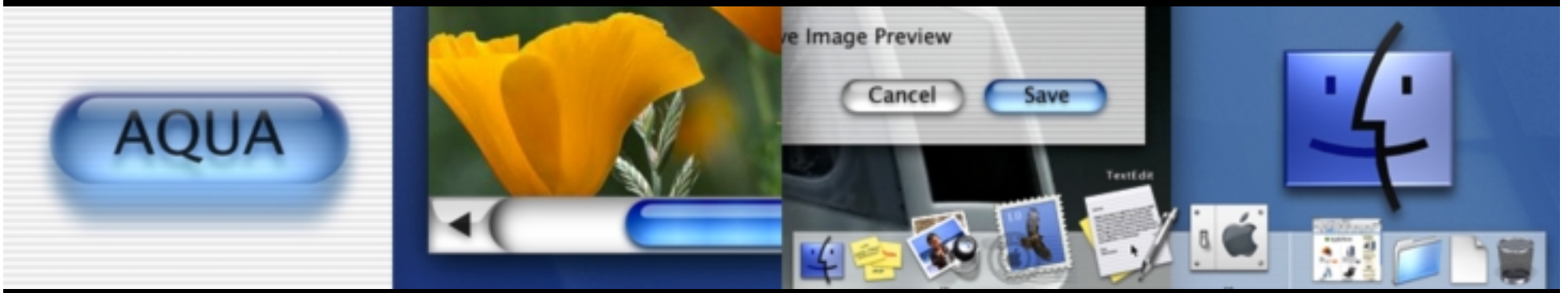
Session 410

WebObjects: Optimizing Applications



Alex Cone and Bill Bumgarner
Partners, CodeFab, Inc.

Introduction

- Developing a scalable, high throughput site requires special planning, careful execution and lots of analysis and tuning. Learn how to build WebObjects apps that can take the punishment and come back for more!

What You'll Learn

- Techniques to write leaner and faster WebObjects applications
- How to optimize your applications before you write them
- How to overhaul an existing application

What Goes Wrong?

- Bound by CPU
  (app server is running at CPU = 100%)
- Bound by memory
  (app server is swapping too much)

What Goes Wrong? (Cont.)

- Bound by network
  (network connection is saturated)
- Bound by implementation
  (responses require too much processing time)
- Bound by database
  (DB server CPU = 100%, too many calls/response)

How Can We Fix It?

- First three by spending $$
  CPU bound? Buy more boxes or CPUs!
  Memory bound? Buy more RAM!
  Network bound? Buy bigger pipe (add bursting)!
- Last two require optimization
  Do less work to generate a response
  Make more efficient use of database

Good Rules to Code By

- Make it work, make it right, make it fast
- Don’t optimize without analysis
- Optimize in small steps and test results after each step
- If it ain’t broke don’t fix it!

Design Optimization

- Understand usage patterns
  - Optimize most used areas first
  - Make entry page fast!
- Plan business logic around response generation
  - Avoid repeating “expensive” calculations
  - Retain and reuse data—and know when it is out of date
  - Manage cached data carefully

Design Optimization (Cont.)

- Minimize memory footprint (smaller application == more instances running)
  - Share data across sessions
  - Clean up thoroughly (do not rely on GC!)
  - Clear transient ivars when no longer needed
  - Use stateless components
  - Use shared sessions if appropriate
  - Set the right session timeout value

Design Optimization (Cont.)

- Plan data access—queries, caching, and cache updating
  - Understand data latency issues
  - Try for 0 queries per response
  - Use in-memory searches where possible
  - Manage faulting–plan relationship population
  - Manage caching–explicitly update stale data
  - Use shared editing context for reference data

Design Optimization (Cont.)

- Use time outside request-response loop for housekeeping
  - Load reference data at app startup (register for **applicationWillFinishLaunching**)
  - Use timers or **performAfterDelay** to do database access between requests
  - Serialize and lock request handling while performing housekeeping tasks to avoid threading/reentrancy issues

Design Optimization (Cont.)

- Partition functionality into multiple applications
  - Separate data maintenance from presentation
  - Move expensive operations from live site to data entry application
  - Use optimized object models for each application
  - Complex object model for data entry app
  - Simplified model for live site query/display
  - Maximize reuse through frameworks

Design Optimization (Cont.)

- Minimize use of frames in UI
- Use Direct Actions
- Beware of mixing Java and Obj-C
  - Crossing language bridge is expensive
  - Use all Java, Java + WebScript, or Obj-C + WebScript

Improving Performance

- OK, the app is done, but it is…
  - Too slow
  - Using too much memory
  - Using too many CPU cycles
  - Occasionally very slow
- Now what?

Don’t Be Silly!

- Make sure **WOCachingEnabled** is on
- Make sure **WODebugging** is off (and use **debugWithFormat**!)
- Have action methods that stay on the same page return **self.context.page**

Start With the Most Frequently Used Bits

- Know the actual usage patterns
  - Log user activity
  - Use WOStatisticsStore logging
  - Capture DirectAction activity
- Tune most visited areas first

Optimize DB Usage

- Change app functionality to avoid pathological behavior
  - Prevent unrestricted user searches by requiring at least one qualifier
  - Use fetch limits—nobody really wants to scroll through 100s of rows!
  - Cache search results
  - Use in-memory searches whenever possible (leverage the cache!)

Optimize DB Usage (Cont.)

- Optimize fetching
  - Use shared editing context for reference data that will not be edited
  - Use session editing contexts only for data that will be edited by session user

Optimize DB Usage (Cont.)

- Optimize fetching
  - Use inter-app messaging to update caches to avoid stale cached data
  - Use time between requests for reference data updates
  - Use raw rows and custom queries to get non-object-based data from the database

Optimize DB Usage (Cont.)

- Look for unexpected fetching
  - Use **EOAdaptorDebugEnabled** to monitor activity
  - Beware of excess faulting
  - Do not fetch data for pop-ups, browsers, etc. in the components; Manage such reference data at the application level and filter as needed for component display

Optimize DB Usage (Cont.)

- Look for unexpected fetching
  - Avoid refaulting shared reference data into session's editing context
  - Manage movement of objects between editing contexts—use **localInstanceOfObject**

Optimize DB Usage (Cont.)

- Optimize eomodels
  - Simplify object model
  - Avoid deep inheritance hierarchies (and deep fetches!)
  - Build simplified read only entities (based on business object tables) with flattened attributes to support user queries

Optimize DB Usage (Cont.)

- Optimize eomodels
  - Build views in DB for queries (DB-sidflattening!)
  - Use batch faulting
  - Use prefetching
  - Watch for excess back pointers

Optimize DB Usage (Cont.)

- Optimize queries
  - Create indices
  - Use “explain plan” to make sure indices are being used
  - Check ratio of cache hit to disk access for common queries
  - Make sure DB is tuned to use available processors
  - Make sure DB is tuned to use available RAM

Optimize DB Usage (Cont.)

- Look at the generated SQL
  - Does it suggest additional indexes?
  - Can it be “hand” optimized? Put tuned SQL in the eomodel. (This is a last resort if EOF insists on generating sub-optimal SQL)
  - Use stored procedures

Optimize Components

- Simplify component nesting
- Define your own (compiled) subclass of **WOComponent**, put common functionality there and make components inherit from that instead of **WOComponent**
- Consider caching pages or using new “stateless” components
- Make static content static!

Refactor Software

- Compile anything that does serious calculations
- Simplify **Application** and **Session** objects, move functionality to singleton “manager” classes (such as a configuration manager or a cached object manager)

Is Your WebServer Doing Its Share?

- Tune configuration
- Use mixture of static (served by web server) and dynamic (served by app server) content
- Offload all serving of content that you can (images, files, multimedia)

Optimize for Fast Browser Display

- Check total size of generated pages
  - Smaller pages display faster
  - Batch displays of long sets of data
  - Generate short URLS (i.e., /images vs. /I)
- Do better things with images!
  - Smaller images
  - Common image names
  - Use less images

Optimize for Fast Browser Display (Cont.)

- Improve the structure of your HTML
  - Use HTML code checker (such as WebLint) on generated pages
  - Simplify table structures
  - Watch for nesting problems (especially nested forms! Don’t work!)

Optimize for Fast Browser Display (Cont.)

- Watch for overlap problems
  (<form><table>...</form></table>, etc.)
- Look at the generated HTML—some problems are within a single component template, others span components

For More Information

http://www.apple.com/webobjects

Visit the WebObjects lab downstairs!
Everyday from 11:00 a.m.–2:00 p.m.

Try out your WebObjects 4.5 Evaluation CD!

Who to Contact

Toni Trujillo Vian
Director, WebObjects Engineering
wofeedback@group.apple.com

Ernest Prabhakar
Product Line Manager, WebObjects
webobjects@group.apple.com


WWDC
Worldwide Developers Conference 2000

Think different.