9th Sakai Conference

This page contains notes from various sessions attended at the conference. They revolve primarily around UI/UX development for Sakai (JavaScript, GWT, etc).

Table of Contents

- Sakai & GWT - Toward Improved UX & Easy Web 2.0 Development all in Java
- Fluid in Your Institution
- Better, Faster and Lighter Sakai UX
- Client-side Rendering in Sakai

Sakai & GWT - Toward Improved UX & Easy Web 2.0 Development all in Java

1430-1530, 1 July 2008

OpenSyllabus is a project that has developed a new Syllabus tool using GWT. It is a dynamic tool that allows a tree view of a syllabus and split frame access to the contents of each syllabus item (table-of-contents view).

Reasons to go with GWT over straight JavaScript:

- it's Java (they cite JavaScript gurus hard to find)
  - it's similar to Swing/AWT and they had developers familiar with Swing - it was easy for them to pick up GWT.
- can use well-established Java tools (Eclipse, JUnit, etc.)
  - JavaScript can be difficult to debug/refactor
- JavaScript is not Object Oriented in the same way that Java is (cannot follow the same design patterns)

GWT features

- similar to Swing/AWT
- rich widgets
- never have to write JavaScript - the compiler converts to JavaScript and is compatible with IE 6, 7; FF 1.5, 2, 3
- built-in internationalization
- can maintain OO design patterns
- powerful Java IDE can be used
  - edit/test/debug/refactor
  - debugging tools
- "Hosted Mode" can simulate various browsers within Eclipse - just need to edit code and refresh to see the changes
- Can move from "Hosted Mode" to "Web Mode" at the click of a button (compilation takes a few minutes)

GWT integration with Sakai

- Sakai used as a stateless server
  - Presentation logic is moved entirely to the client (just one page: index.jsp)
  - Server needs data services and an RPC server (RPC Servlet, extends RemoteServiceServlet GWT class)
  - Client contains the MVC code
- GWT Design Patterns
  - Big paradigm shift: client oriented instead of server based
  - Main client design pattern is MVC + Observer + Event driven
  - Asynchronous programming
  - Minimize JSON, XML - just use POJOs for data models (? RPCs take care of data transfer ?)

GWT Challenges

- Spring & GWT wiring; solution: ApplicationContext
- I18N is not dynamic out of the box - for more complex strings, "i.e. Hello, JOHN", they needed to develop their own mechanism for resource bundle key lookup
- CSS is still browser dependent, need to check with different browsers
- more... see presentation notes (ask for link to ppt)

Summary/Thoughts (with notes from discussion after the presentation)

- Their development team was split between "front-end" and "back-end"
  - there was only ever one person working full time on the project and up to eight people on the project at its height
  - their UI designers were actually developers
  - they were unable to use GWT's UI design tools because of unfamiliarity (learning curve). They did use it for mockups, but never used it to hook up the actual application behavior.
- They really like GWT because they are strictly a Java shop. They don't know or like JavaScript and are more accustomed to other UI technologies like Swing. It was real easy for them to pick up GWT and use all their familiar tools.
- Claude did mention that if they had a JavaScript guru, things may have been different and they may have continued with JavaScript. But, given Sakai's Java environment, GWT is clearly advantageous.
Fluid in Your Institution

1330-1430, 2 July 2008

Fluid Project

- Direct Technical Fluid questions to Colin Clark

UX Challenges

- Open source projects reward code generation, not designs... there's not a lot of formal process for UI designers to participate
- how can we build a common language between UI designers and developers to work together efficiently and happily
- poor UX an impediment to adoption
- it’s tackled at the end of development or even once apps are in production
- range of roles not well understood or invested in:
- Academic communities are very diverse: preference, needs, habits, concepts, comforts, convictions, language, etc...

Fluid Design Methods

- Participatory design (user-centered design) - recruit individuals who are going to be using the tool help design it
  - in community source, we do not have the "traditional" limits in access to users that exist in industry (no marketing /sales departments, etc.) - you can just walk out onto the Diag and observe user behavior and/or have them test your designs and tools. You can have very fast iterations of the design-develop-user test-design-... cycle.

Fluid Resources

- UX Toolkit (user research tools, contextual inquiry, persona development and library...)
- add link here soon...

Discussion

- Tool metaphor is big in Sakai. So, if you know how to use a tool, you're good, but not a lot of people think about it that way. For example, faculty come in thinking about a learning experience, not about tools to create that experience.
- Tools do not have a standard look-and-feel or standard development design patterns. Resources for developers are scattered, but Fluid is working to provide a one-stop-shop for both:
  - developing Sakai applications with a consistent, accessible, relevant UX
  - getting up and running with a Sakai deployment including local customizations and skinning

Better, Faster and Lighter Sakai UX

1430-1530, 2 July 2008

- Sakai is very developer driven
- UI Designers/Developers/User Stakeholders are trying to get in, but they are essentially blocked
  - trying to change this: Web 2.0, OpenSocial, etc.
  - would change team structure, development cycles: faster!
- scalability bonus: client is doing more work, server less
- it's more efficient to drop an HTML file and refresh rather than building maven, starting Tomcat...
- traditional: Java driven- get some requirements, go off for a long time... build the tool... (briefly) test the tool... the "test interface" usually becomes the final product.
- UI dev driven: user interviews, user personas/scenarios, interaction design with mockups/wireframes, pass to Java developer (they have focused goals to accomplish), take back to the user (user test), cycle through this a couple of times refining the UI /functionality, finalize

Questions

- HOW FAST... what did the iterations look like? how many iterations?
  - what were challenges in moving faster?
  - how could you go even faster?

Demo

- MyCamTools looks good, works fast
  - to get access, email Ian Boston - public demo might be coming soon

Client-side Rendering in Sakai

1600-1700, 2 July 2008

Shift towards client-side

- users want more interactive apps
- increasing

Client-side rendering
• pros (reduced cycle times, portability of "components" across different server-side frameworks and languages)
• cons (harder to test, greater risk of client-side performance
  harder to degrade gracefully to a "plain view", loss of document idom: links, Google, save etc.)

Why are we building a client-side template rendering system?

• need a way to reuse markup without cut & paste: "Here's a component that you can move around, server agnostic."
• template rendering that
  • is unobtrusive/customizable
  • allows for choice: seamless client or server rendering
  • fully independent from RSF

Separate Behavior from Markup

• goal: highly customizable HTML
• building stuff that will not break if elements/structures change
• Fluid components fit in with any rendering engine (client or server) but a default engine built in to Fluid is what we're working on... just another choice (WHAT are advantages/disadvantages?)

Simple example

• table heading - JSON data
  Unknown macro: {"header}
• RSF.selfRender - does the rendering, very simply
• we can reference actual rsf:ids in the template or use "pure selector rendering" and use a jQuery selector to find what needs to be replaced. This is cool because it gets rid of ALL RSF/JavaScript from the template. Designers only need to work with HTML and CSS.
• "Duck compression" is used in JSON - packages data in a certain way and inspects the type of component tree - "automatic" generation/mapping to UI elements

Prototyping

• you can do everything in JavaScript
• use JSON to render dummy data
• then, just use JSON for real data later on...

Separation of Concerns

• html (template)
• JSON (data)
• JavaScript (transformation of static template into dynamic component)

RSF Data Binding

• RSF.explode
• takes a JSON object and for every member, creates another JSON object with the information of where/how it fits into the DOM
• kind of uses EL paths...
• functionally equivalent (though much better performance) to RSF Server-controlled fossilization.

Where are we going?

• refine selector-based approach
• build a jQuery plugin
• more performance tuning
• Fluid community support: put powerful new techniques within reach of ordinary JavaScript developers

Monthly release cycle: Fluid 0.5, Aug 2008; Fluid 0.6 Fall 2008

RSF 0.7.3 - Out next week

• client rendering in preview mode
• data views (JSON, XML and other formats)
• RSF self-testing