Building Real-World Dojo Web Applications

Andrew Ferrier
IBM Software Services for WebSphere

Agenda

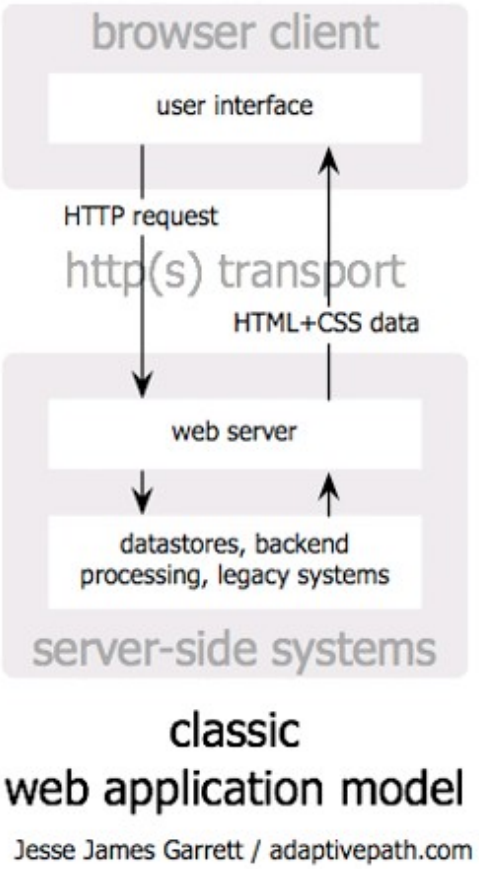
- How has building web applications changed?
  - Web 1.0 to Web 2.0
  - AJAX / REST / JSON
- What is Dojo?
- Why is IBM interested and how does it fit in to WebSphere?
- A flavor of Dojo
- How do we design applications for Dojo?
- Some Dojo Best Practices

Web 1.0 Model

- Static HTML content, little-to-no-dynamicity
- Most folks know this already
- Server-side-driven content
  - Page per state
  - Perhaps with a small amount of JavaScript for effects or form validation
  - Traditionally written with a variety of technologies – Servlets, JSPs, etc.

(1) User actions trigger HTTP request to a web server

(2) Server does some processing and returns an HTML page to client



classic web application model

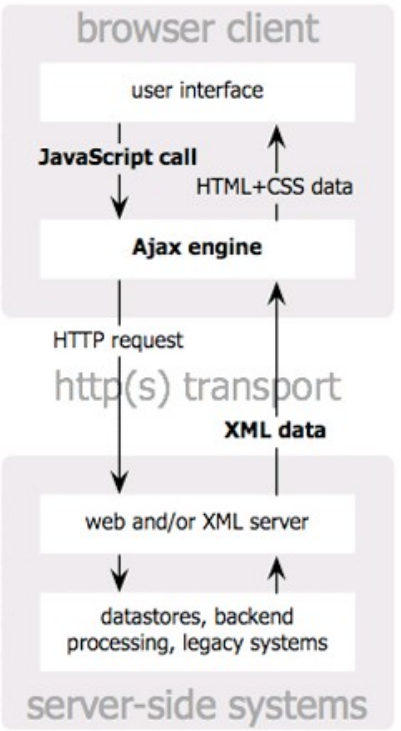
Jesse James Garrett / adaptivepath.com

But this has disadvantages...

- Interaction is awkward
- Slow response time – every action requires a entire page refresh
  - W3C states that 1 second is highest acceptable response time for interactive action
- Often ugly, flickery, content

Web 2.0 Model

- Browser using AJAX to communicate with server
- Lightweight RESTful Service Calls
- Service Gateway or other technology to proxy all service invocations



Ajax
web application model

(1) User actions trigger JavaScript call to Ajax engine

(2) Ajax engine Invokes asynch request

(3) Server does processing and returns XML to the Ajax engine

(4) Ajax engine Displays the XML in the browser

What is AJAX?

- Asynchronous JavaScript and XML
  - Doesn't have to be about XML, and often isn't
- Uses XMLHTTPRequest feature of browser to make an HTTP invocation without refreshing page
  - Used to retrieve data
  - 90% of the time this is formatted in JSON (JavaScript Object Notation)

JSON

```
{
    “customerName”: “Andrew Ferrier”
    “customerHeightInCm”: 100
    “customerAddresses”: [
        { “addressType”: “home”,
          “addressLine1”: “123 Anytown”,
          “addressPostcode”: “AB12 3CD”
        }
    ]
}
```

What is REST?

- Use resource-driven URLs:
  - e.g. /rest/customer/1234
- Leverages HTTP Verbs
  - GET, PUT, POST, DELETE
- Leverages HTTP Response Codes
  - 200, 401, 403, etc.
- Typically uses lightweight data format
  - JSON, plain text
- Particularly well-suited to CRUD

Web 2.0 Browser Model

- Thick-ish client
- JavaScript code runs in browser
- Events on page are used to drive code (onLoad, onClick, etc...)
- Extra data and code are requested from server using XHRs, with async callbacks in code

Web 1.0 Page

- Static HTML with mixed layout and content:

```
<html>
	<head>
		<title>Customer Page</title>
	</head>

	<center>
		<h1>Hello, and welcome to the page all about <b>International Shipping</b></h1>
	<center>

	<p>Some of the key facts about <b>International Shipping</b> are:</p>
	<ul>
		<li>They spend over <em>$5.5</em> every year on paint.</li>
		<li>They use an aged fleet.</li>
		<li>We can sell them more paint.</li>
	</ul>
</html>
```

- Little to no JavaScript

Web 2.0 Page

- 3 primary components:
  - HTML (DOM Model)
    - Tree-like structure of HTML tags
    - These days we use a lot, we avoid 'styled' tags like <b>.
  - JavaScript (including Dojo)
  - CSS for Styling

Browser – Separation of Concerns

- Model (DOM) / View (CSS) / Controller (JavaScript) allows for:
  - Decoupling of styling and layout (CSS) from content (HTML)
  - Decoupling of code (JavaScript) from page model (DOM)
  - Event-driven programming model

Separation of Styling

```
<html>
	<head>
		<title>Customer Page</title>
	</head>

	<div class="header">Hello, and welcome to the page all about <span class="customername">International Shipping</span></div>

	<p>Some of the key facts about <span class="customername">International Shipping</span> are:</p>
	<ul>
		<li>They spend over <span class="money">$5.5</span> every year on paint.</li>
		<li>They use an aged fleet.</li>
		<li>We can sell them more paint.</li>
	</ul>
</html>
```

```
div.header {
	font-size: 24pt;
}

span.customerName {
	font-weight: bold;
}

span.money {
	font-style: italic;
}
```

Let's not forget the Service Gateway...

- Facades RESTful services to UI, primarily to resolve cross-domain security restrictions
- Can be implemented in a variety of technologies:
  - JAX-RS
  - Servlets
  - WebSphere ESB and Process Server
  - … virtually anything else that can host HTTP

What's wrong with JavaScript?

- Traditional JavaScript weak at:
  - Structured code – no classes, modules
  - Cross-browser support – incompatibilities, especially with the dreaded IE6 (http://www.ie6countdown.com/)
  - No support for complex or rich form controls
  - Hard to handle JavaScript context and other awkwardnesses

So what is Dojo?

- Dojo is a set of common JavaScript libraries used for creating Ajax and DHTML web applications
- http://dojotoolkit.org
  - Open Source
  - Large widget collection (“dijits”)
  - Powerful I/O (XHR)
  - Data abstraction layer
  - Event management
  - Logging and debugging
  - Extensible modular architecture
  - Declarative and programmatic

Why is this exciting and important?

- Rich, powerful user experience, bridging interactivity gap
- Low barrier to entry in many environments
  - Main requirement is a modern web browser
    - Firefox 3.5+, Chrome, Safari
    - IE 7+ also OK-ish
- With some kind of service gateway technology, can connect to most types of backend

Why Is IBM Interested?

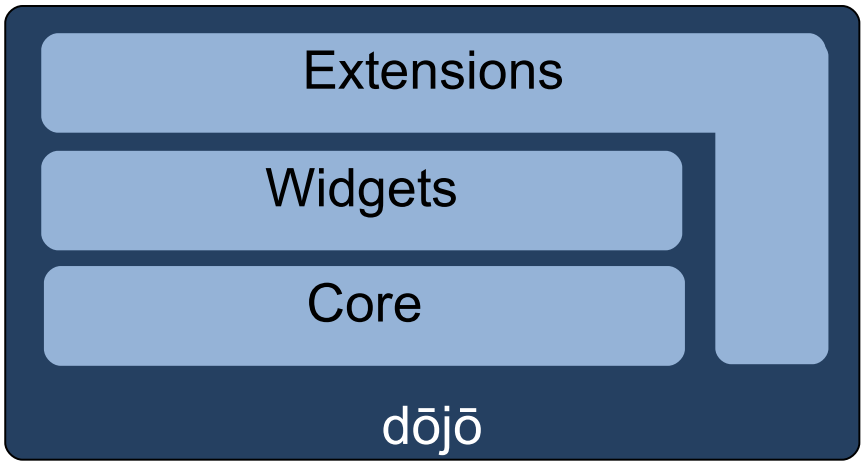
- Part of an overall business solution
- IBM does more than middleware
- **Web 2.0 Feature Pack for WAS contains Dojo**
- Building Modern Rich Applications

Why use Dojo over the alternatives?

- Numerous competing JavaScript toolkits
- JQuery, Prototype, YUI, Mootools, GWT

dōjō

- Dojo is IBM's strategic choice for a number of reasons
- More than just DOM access, CSS effects and AJAX
- Enterprise-grade toolkit and feature set (internationalization, accessibility, etc.)
- Actively supported and used by IBM



A Flavor of Dojo

Dojo Base

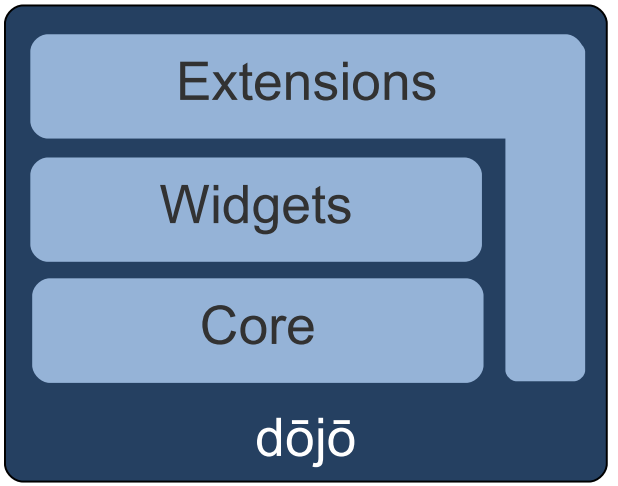
- Dojo “Base” libraries are Dojo functions that are always available inside the base dojo.js bootstrap

- Module Loader
- Lang Utils & Array Extras
- Cookie functions
- Query, Node & Style Utils
- I/O (Ajax)
- JSON serialization
- Fast CSS Style manipulation
- Deferreds
- Events (simple connect)
- Color functions
- Browser detection
- URL functions
- Doc Load/Unload Hooks
- Animation & Effects
  - Fade, slide
  - CSS colors

Dojo Components

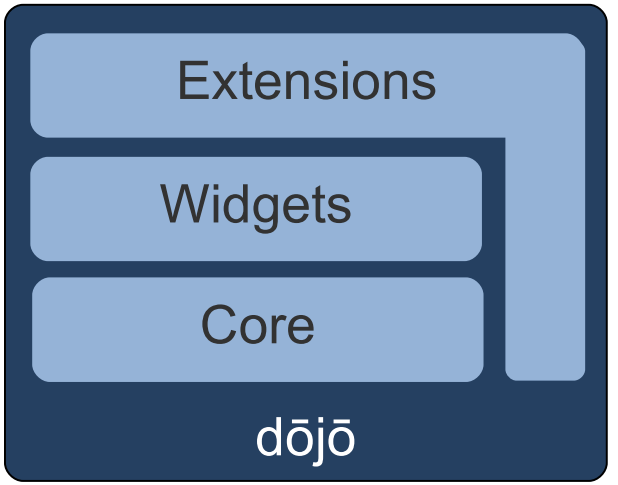
- Base
  - Extremely common functions that are always available no matter what else is included
  - Provided within the dojo.js (<**60kb**, <**24kb** zipped)

- Core
  - Common functions
  - Loaded externally
  - Extended animation effects
  - Date parsers
  - Additional IO above **xhr**
  - and more



Other Functions

- Dijit
  - Internationalized, accessible form, layout and specialized widgets built upon a powerful set of widget functions
- Dojox
  - Everything not yet included in dojo / dijit
  - Quality varies widely
  - Read author notes!
- Utils
  - Tools to create custom builds
  - Reduce load time and size



Loading Dojo

- Import any stylesheets and themes you wish to use in your application
- Link to core **dojo.js**

```
<!-- Import themes / styles -->
<style>
  @import "/dijit/themes/soria/soria.css"
</style>

<!-- Load the base Dojo -->
<script type="text/javascript"
  src="/dojo/dojo.js"
  djconfig="parseOnLoad:true, isDebug:true">
</script>

<!-- Define required modules -->
<script type="text/javascript">
  dojo.require("dojo.parser");
  dojo.require("dijit.form.Form");
  dojo.require("dijit.form.Button");
</script>
```

Adding Styling, Modifying DOM, Adding an Event

```
dojo.addOnLoad(function(){
  dojo.query("div.thinger > [someAttr]")
    .addClass("thingerItems")
    .style({
      backgroundColor: "blue",
      opacity: 0.5

    .onclick(function(e){
      dojo.anim(e.target, { opacity: 1 });
    });
});
```

Credits: Excerpt from Dojo Fundamentals Ajax Experience presentation by Alex Russell

AJAX with XHR

```
var deferred = dojo.xhrGet( {
        url: "/services/subscriptions",
        handleAs: "json",
        Timeout: 10000, // 10 sec (Time in milliseconds)

        // The LOAD function will be called on a successful response.
        load: function(response, ioArgs) {
                        var feed = response.feed;
                        // ... Do something with the data ...
                        return response;
        },
        // The ERROR function will be called in an error case.
        error: function(response, ioArgs) {
                console.error("HTTP status code: ", ioArgs.xhr.status);
                return response;
        }
} );
```

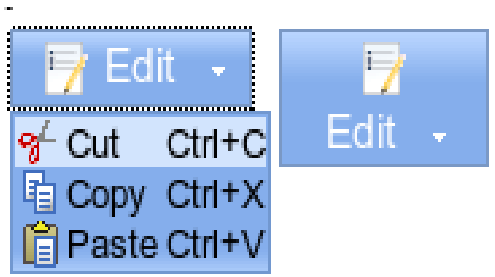
Using Dijits

Declarative

```
<div dojoType="dijit.form.DropDownButton" menuId='editMenu'>
  <img src="images/note.gif" width="20" height="20" />
  Edit
</div>
```

Programmatic

```
<div id="button2"></div>
<script type="text/javascript">
  dojo.addOnLoad( function() {
    var button = new dijit.form.DropDownButton({
      caption: "<br>Edit",
      img:     "images/note.gif",
      menuId:  "editMenu"
      }, dojo.byId("button2") );
  });
</script>
```



Some Dijits

- Uniform Input visuals on all browsers
- dojo.form.Form provides easy access to all form values as a single JSON object

•DateTextBox

Text Inputs

- TextBox
- ValidationTextBox
- TextArea
- DateTextBox
- TimeTextBox
- SimpleTextArea
- NumberTextBox
- NumberSpinner
- CurrencyTextBox

Selectors

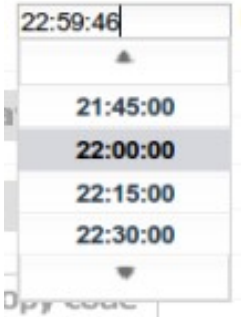
- ComboBox
- MultiSelect
- FilteringSelect

Others

- Checkbox
- Radio
- Slider

Buttons

- Button
- DropdownButton
- ComboButton



•TimeTextBox

Writing Dijits

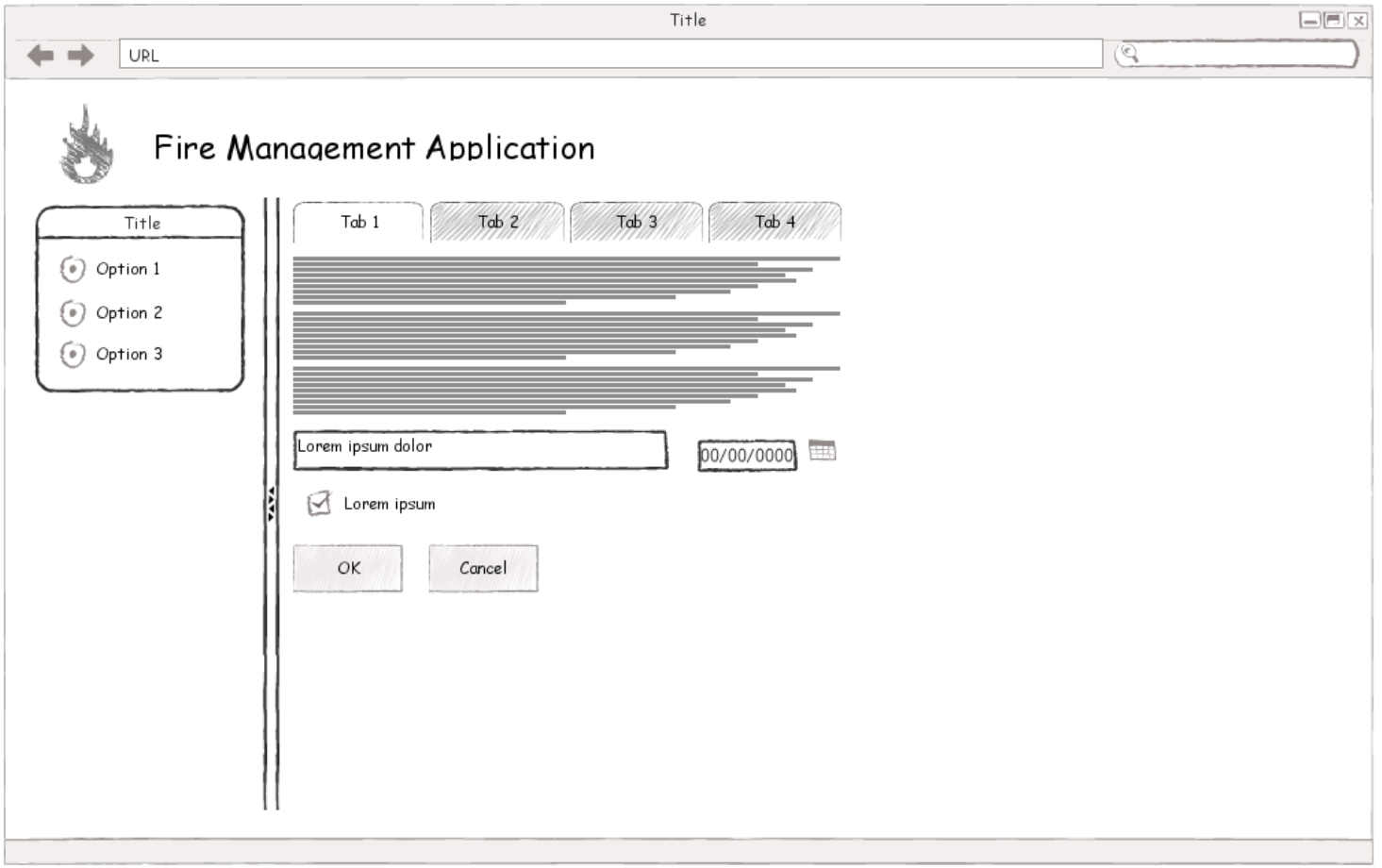
- Can write own dijits
- 3 basic components:
  - Template HTML (use dojoAttachPoints)
  - JavaScript with certain call-in functions, such as *constructor()*, *buildRendering()*, *postCreate()*, etc.
  - CSS File

So how do we design?

- Wireframing to mock up UI
- Break UI into Bijits, specify each
- Service Catalog for Gateway
- Other definitions for backend services
  - Web Services
  - Legacy Services
  - Database DDLs
  - etc...

Wireframe page

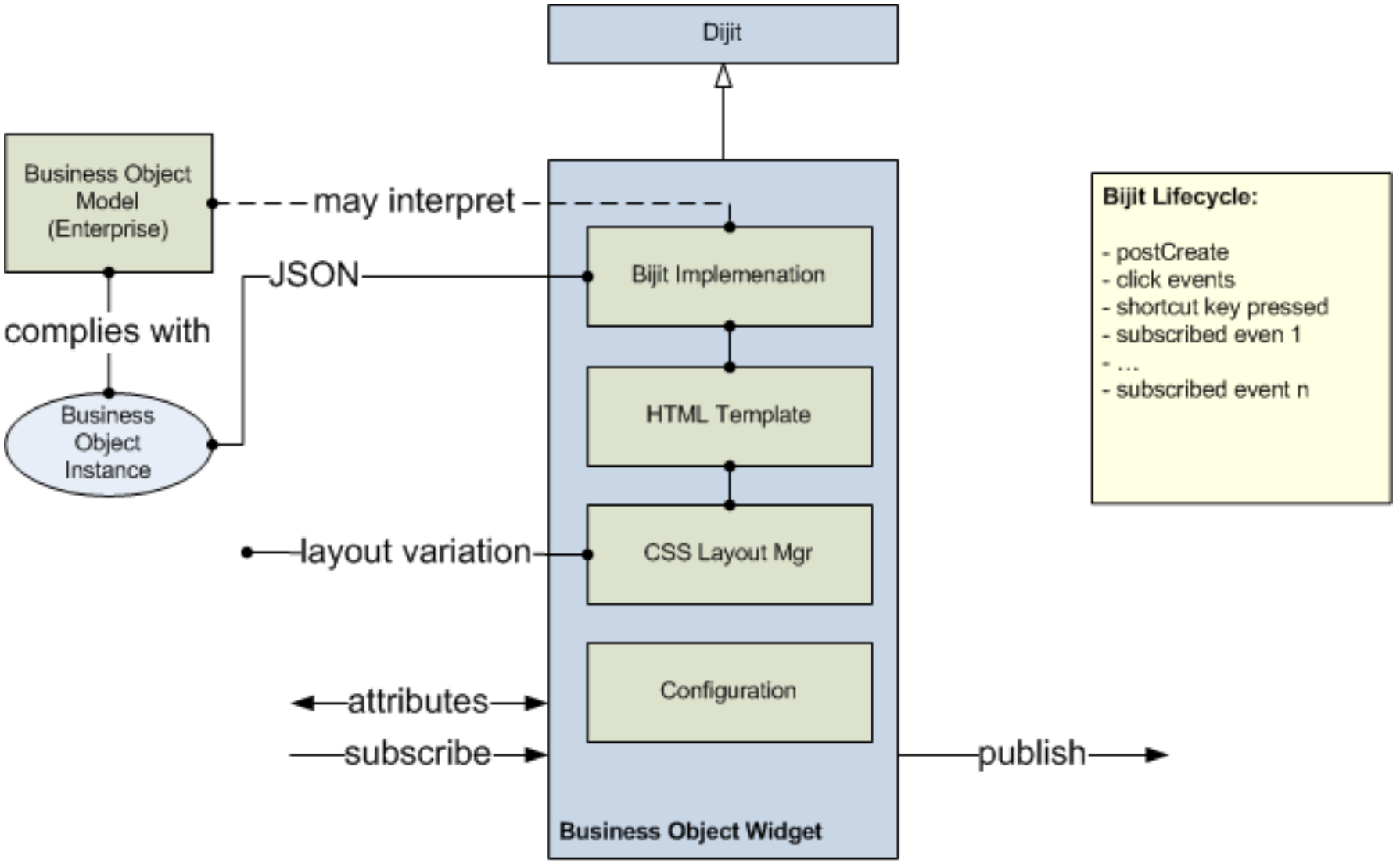
- I did this using iplotz.com, more mature tools available



Designing pages

- Break page into re-usable bijits
- Each is effectively a dijit with a business function
  - Has well-defined properties, events, etc.
- Nothing special about bijits – you don't need explicit support
- Tie then together using a page controller (e.g. page controller concept from IBM's Kuba)
  - Just a class that does wiring

Bijits


Dijit
Business Object Model (Enterprise)
may interpret
JSON
complies with
Business Object Instance
Bijit Implemenation
HTML Template
CSS Layout Mgr
Configuration
layout variation
attributes
subscribe
publish
Business Object Widget
Bijit Lifecycle:
- postCreate
- click events
- shortcut key pressed
- subscribed even 1
- ...
- subscribed event n

Example Bijit – Customer

Service Catalog

- Find a way to describe JSON-based services
- Decouples implementation of UI from services
- Allows implementation by separate teams
- No formal templates or definitions
- No widely-accepted JSON schema description
  - JSON is so simple, doesn't need one

Service ID	HTTP Verbs				URL	JSON Request (PUT/POST)	JSON Response
	GET	*PUT*	*POST*	*DELETE*			
1	X	X	X	X	/rest/customer/${customerid}	{	{
						"customerName": string	"customerName": string
						"customerAddress": string	"customerAddress": string
						}	}
2	X				/rest/order/${orderid}		{
							"ordered": date
							"delivered": date
							}
3				X	/rest/product/${productid}		…

Service Catalog Best Practices

- Keep services fine-grained
  - Coarse-grained is a warning sign
- If you must do coarse-grained, always describe JSON envelope in detail
- Services should always be short-running

Best Practices

- Develop in Chrome or Firefox with Firebug
- Don't Forget Internet Explorer (esp. IE6). To debug:
  - Use IE 8 in backwards-compat mode with debugging console
  - Use JSLint...

JSLint

- By Doug Crockford
- www.jslint.com
- Javascript developer's “honest” friend
- Examine the entire output!
  - Do all those globals look right?
- Play with options
- Works with HTML and JSON data too

- Install JSLint Eclipse plugin from Rock Star Apps

Best Practices

- Do Unit Testing - Dojo DOH framework: http://www.sitepen.com/blog/2008/04/15/unit-testing-custom-code-with-the-dojo-objective-harness/
- Continuous Integration
  - Use a continuous integration server such as Hudson, Bamboo, etc...
- Need CSS experience on project – an often over-looked skill
- Learn how to use HTTP headers to leverage caching

Best Practices

- Do Logging

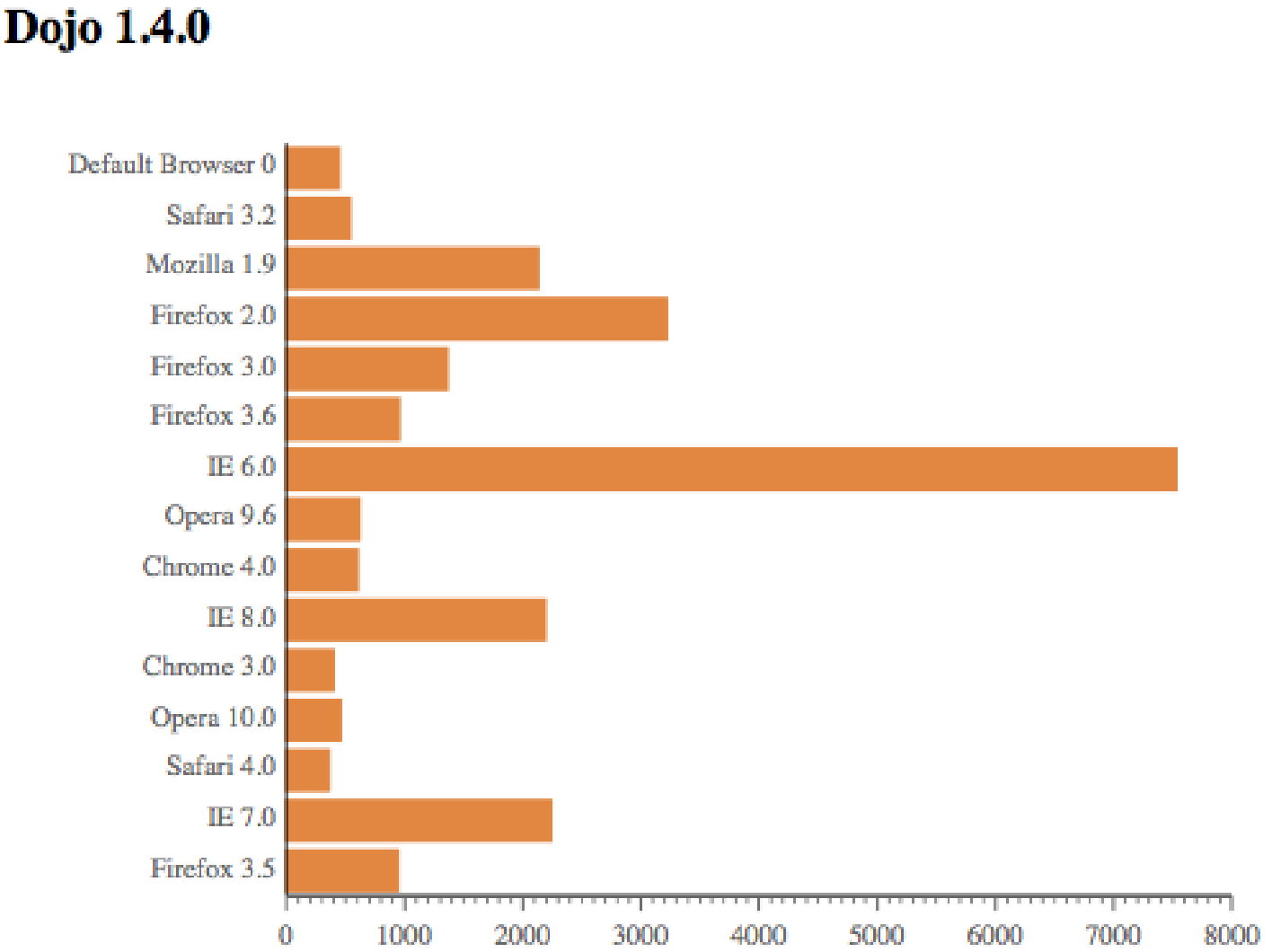
```
dojo.provide(“my.mod.Foo”);
dojo.declare(“my.mod.Foo, null, {

  constructor: function my_mod_Foo_constructor(args) {
    // Do any initialization here
    var F = this.declaredClass + “contructor(): ”;
    console.log(F,”Starting: ”, args);
  }

});
```

- Make sure developers understand JavaScript (context and async behavior in particular)
- Generate 'production' code using an automated build: http://www.ibm.com/developerworks/web/library/wa-aj-custom/index.html

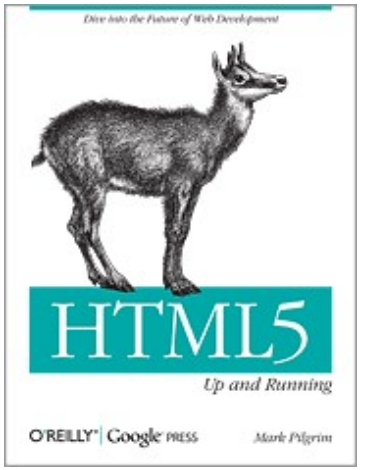
Use an up-to-date browser

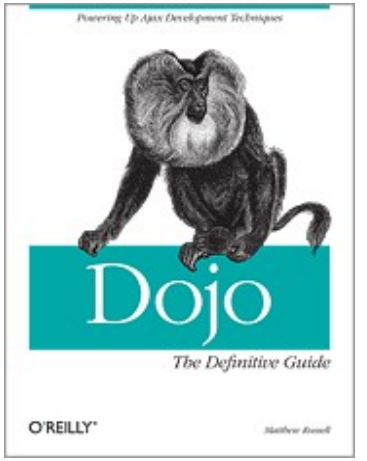


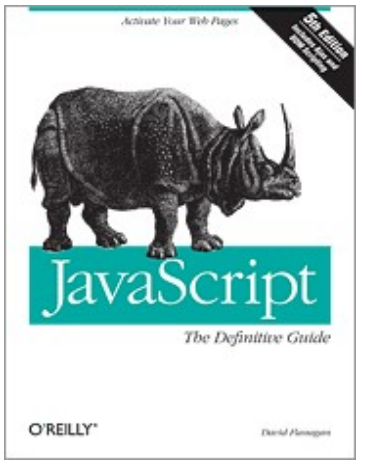
Taken from TaskSpeed benchmarks - http://dante.dojotoolkit.org/taskspeed/report/charts.html

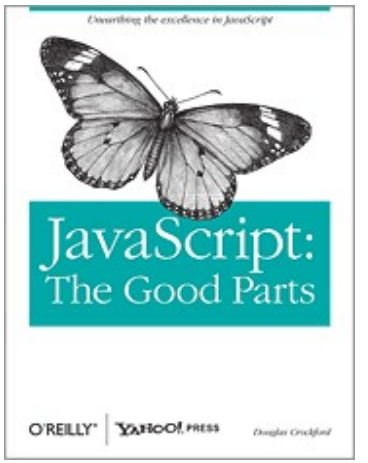
Further Learning

- http://dojotipsntricks.com/
- O'Reilly Books
- http://dojocampus.org/
- http://dojotoolkit.org/









Summary

- Modern, interactive, Web 2.0 applications are the future
- Dojo helps you build these quickly, efficient, and to a high quality
- To design:
  - Wireframe pages
  - Break them into bijits
  - Write a service catalog

andrew.ferrier@uk.ibm.com