
# Ajax.framework: Under the Hood

Chuck Hill
Global Village Consulting, Inc.
WOWODC 2011

# Session Overview

- My Goals
- Outline:
  - Ajax vs Classical WebObjects
  - How Things Work vs Just Works
  - Building Blocks
  - Case Study
- No JSON

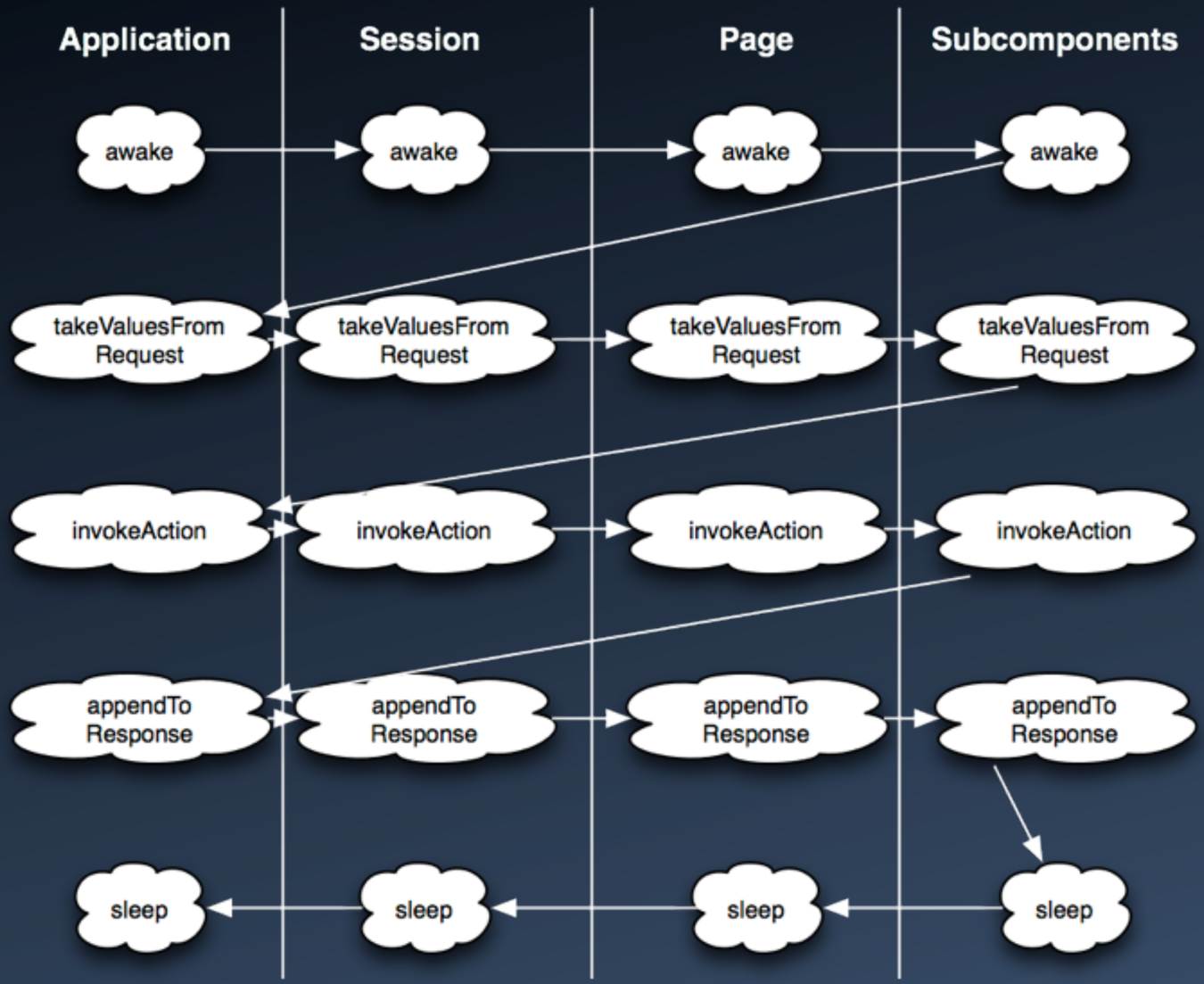
# A Word on our Sponsors

- Jean-Francois Veillette
- Anjo Krank
- Mike Schrag
- the rest of us
- you

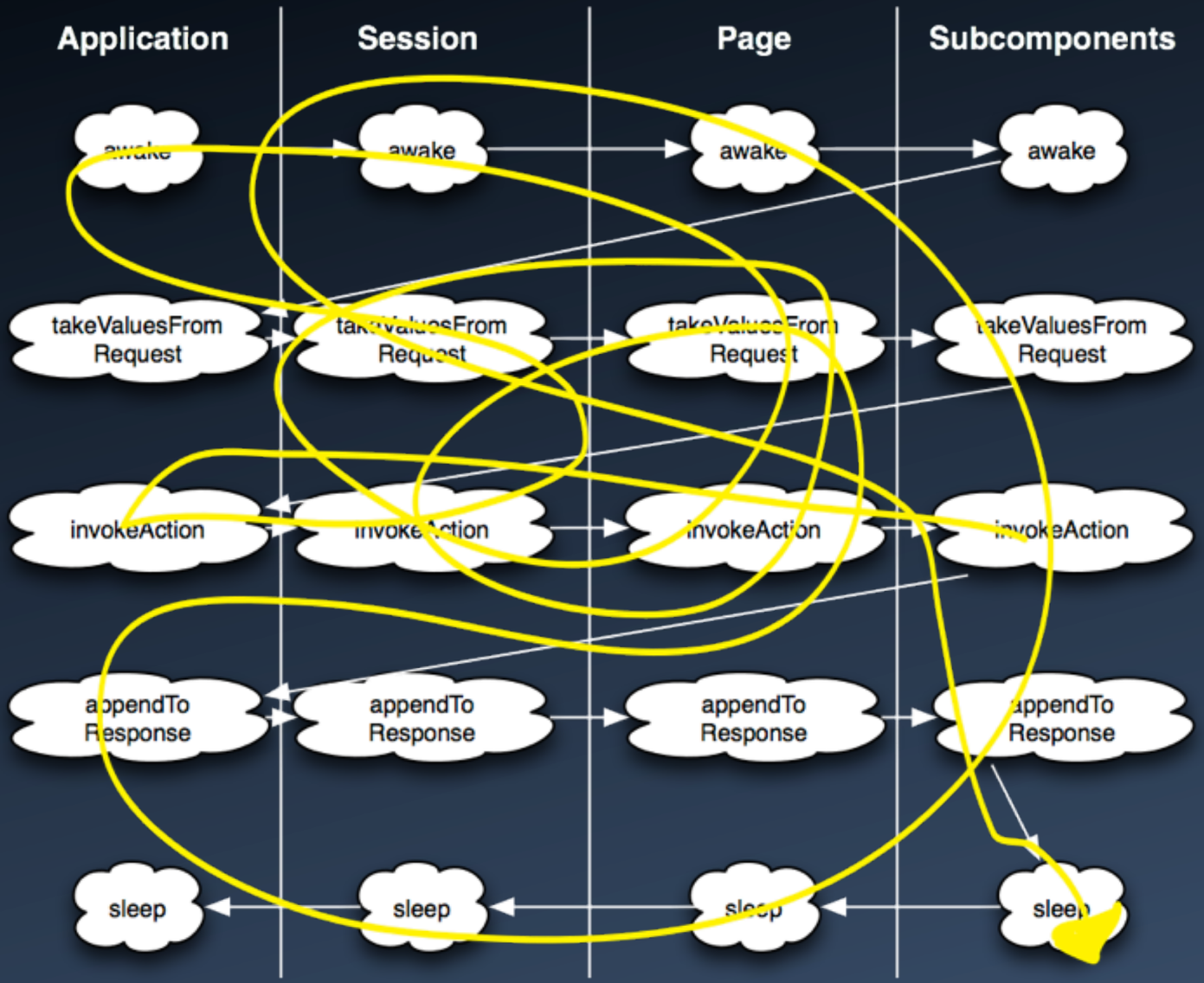
# A Word on JavaScript

- Not a toy scripting language
- NOT Java, "Lisp in C's clothing"
- functional programming
- loose "duck" typing and prototypal inheritance
- dynamic objects
- Douglas Crockford and Yahoo

# Classical Request/Response Loop



# Ajax R-R in a Nutshell



# New R-R Loop Phase

- handleRequest
- replaces action method
- returns response for Ajax request
- called by invokeAction
  - ... at the right time!
- takeValues→invokeAction→handleRequest→appendToResponse?'
- returns AjaxResponse (important!)

# Ajax Action R-R Loop

- “The Basic”
- takeValues
- invokeAction
- handleRequest
- whatever
- can return JS, HTML, or nothing
- client-side result varies

# Ajax Update R-R Loop

- “The Meat Lover’s”
- ...?_u=MyContainerID&1239887663
- invokeAction
- handleRequest
- AjaxResponse.generateResponse
- invokeAction (yes!  again!)
- handleRequest (on AUC)
- appendChildrenToResponse (AUC)

# Ajax Submit, Action, & Update

- “All Dressed”
- takeValuesFromRequest
- invokeAction
- handleRequest
- AjaxResponse.generateResponse
- invokeAction (yes! again!)
- handleRequest (on AUC)
- action method (result discarded)
- appendChildrenToResponse (AUC)

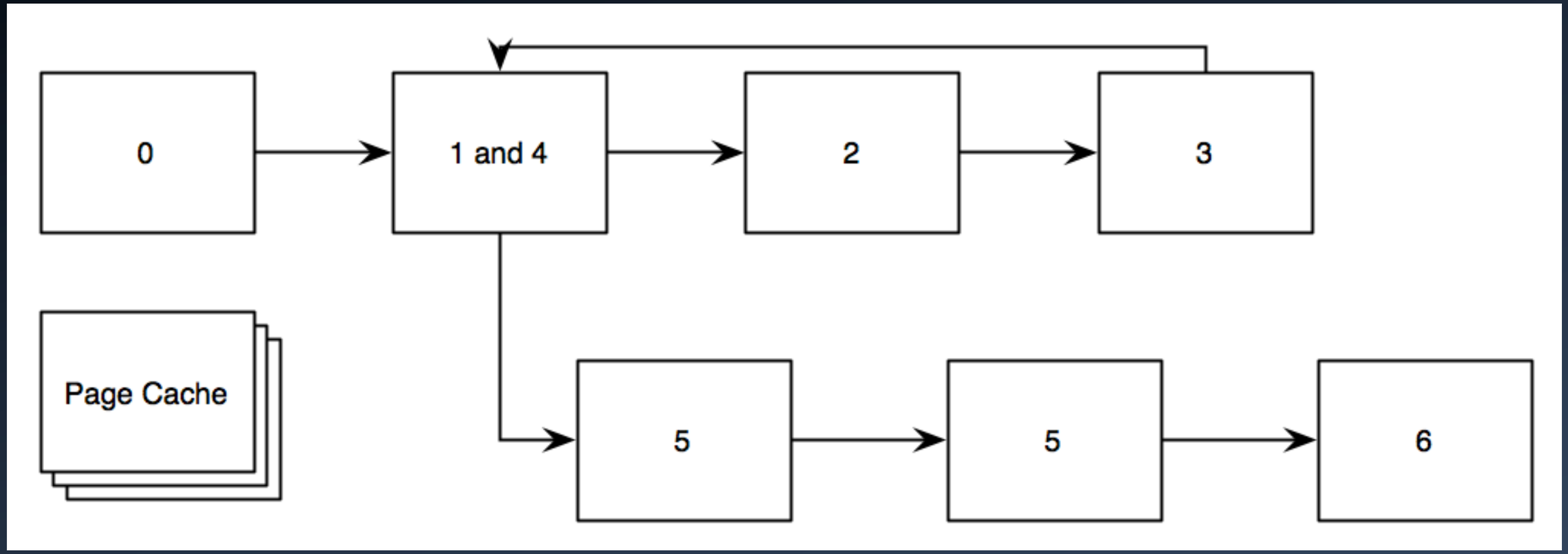
# Ajax Replace R-R Loop

- “Poutine Pizza”
- ...?_r=true&1239887663
- takeValuesFromRequest
- invokeAction
- handleRequest
- action method (component) returned as replacement)
- intuitive, but may be dangerous

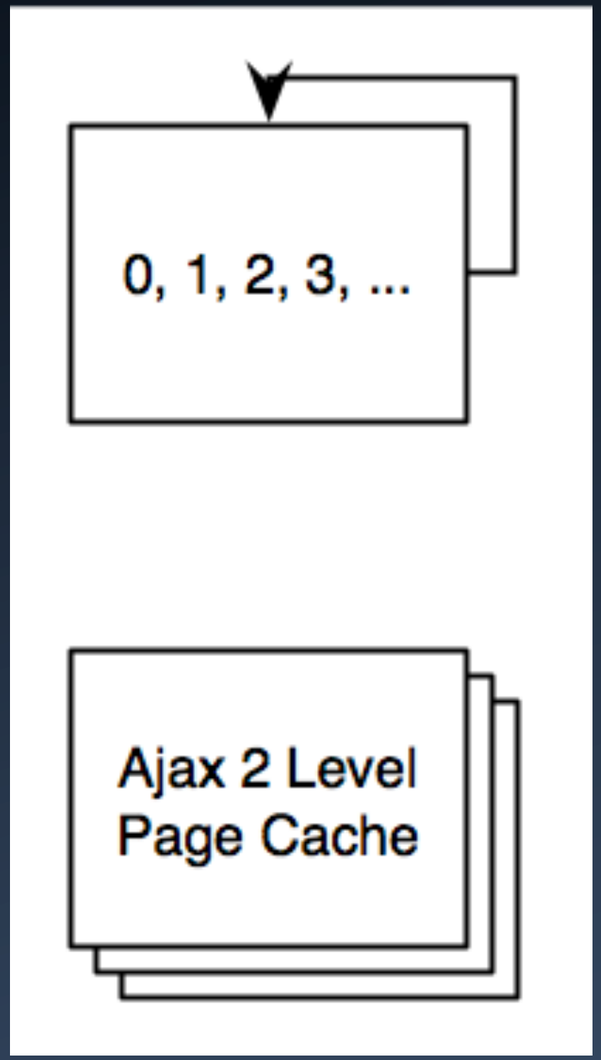
# Special Ajax Classes

- AjaxRequestHandler: don't store page
- ERXAjaxApplication: no takeVlaues for ?_u=xxx&2309482093
- ERXAjaxSession: Ajax Page Caching
- ERXAjaxContext: partial form submits
- AjaxResponse: generateResponse→invokeAction for AUC

# Classical Page Caching



# Ajax Page Caching



# Support Classes

- AjaxComponent
- AjaxDynamicElement
- AjaxUtils
- AjaxOptions, AjaxOption, AjaxConstantOption, AjaxValue

# AjaxComponent

- WOComponent replacement
- Stateful
- Mostly familiar usage
- handleRequest
- addRequiredWebResources

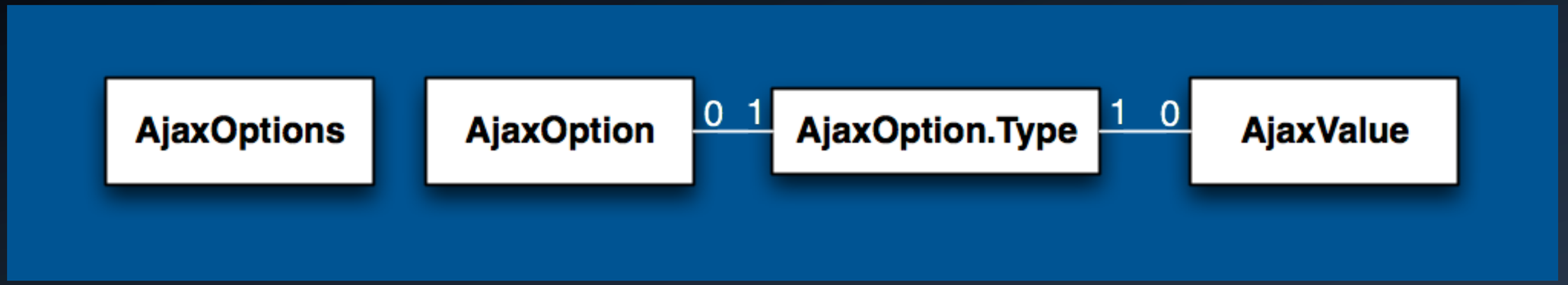
# AjaxDynamicElement

- WODynamicGroup replacement
- Stateless
- Harder to use than WOComponent
- Binding and template access
- handleRequest
- addRequiredWebResources

# AjaxUtils

- createResponse
- javascriptResponse
- Add JS and CSS resources (via ERXResponseRewriter)
- shouldHandleRequest
- binding convenience with JSON parsing
- response generation
- updateDomElement
- redirecting
- etc. seek and ye shall find

# JavaScript/JSON Value Encoding




```
new Ajax.Request('/some_url', {
    method: 'get',
    parameters: {company: 'example', limit: 12}
  });
```

# AjaxValue

- converts to JavaScript/JSON format
- type-aware value
- type guessing ability
- `javascriptValue`
- [a, b, ...], {a:b; ...}, escaping  in strings etc.

# AjaxOption.Type

- simple enumeration
- AjaxValue data types
- update if you extend AjaxValue

# AjaxOption

- named for Prototype parameter "`options`"
- bridge from bindings to JavaScript
- component or dynamic element
- Parameters
  - JavaScript Name
  - Type
  - Binding Name
  - Default Value

# AjaxConstantOption

- Used for computed values

```
public AjaxConstantOption(
    String   javaScriptName,
    Object   constantValue,
    Type     type)
```

# AjaxOptions

- with an S
- collection of key-value pairs
- used as component* or class
- *component usage likely buggy
- works with StringBuffer and WOResponse
- works best with

  `AjaxOption.createAjaxOptionsDictionary`

# AjaxOptions Usage

```
  public NSDictionary createAjaxOptions(WOComponent component) {
  NSMutableArray<AjaxOption> list = new NSMutableArray<AjaxOption>();
  list.addObject(new AjaxOption("frequency", AjaxOption.NUMBER));
  list.addObject(new AjaxOption("onLoading", AjaxOption.SCRIPT));
  list.addObject(new AjaxOption("evalScripts", Boolean.TRUE, AjaxOption.BOOLEAN));
  ...
  NSMutableDictionary<String, String> options =
   AjaxOption.createAjaxOptionsDictionary(list, component, associations());
  return options;
 }

response.appendContentString("AUC.registerPeriodic('" +
id + "'," + canStop + "," + stopped + ",");
AjaxOptions.appendToResponse(options, response, context);
response.appendContentString(");");
```

# Part II

- Being the Second Part

# Twenty Steps to WO-Ajax

- Who loves ya baby?
- From discovery to commit
- slowly
- in 20 steps

# Step 1: Review and Plan

- Find something
- See if it is worth using
- Plan how you will use it

# Step 2: Check that License

- Licenses like the GPL are best avoided
- For Wonder: BSD, MIT or Apache 2 are preferred
- For your own projects consider the terms

# Step 3: Download Source Files

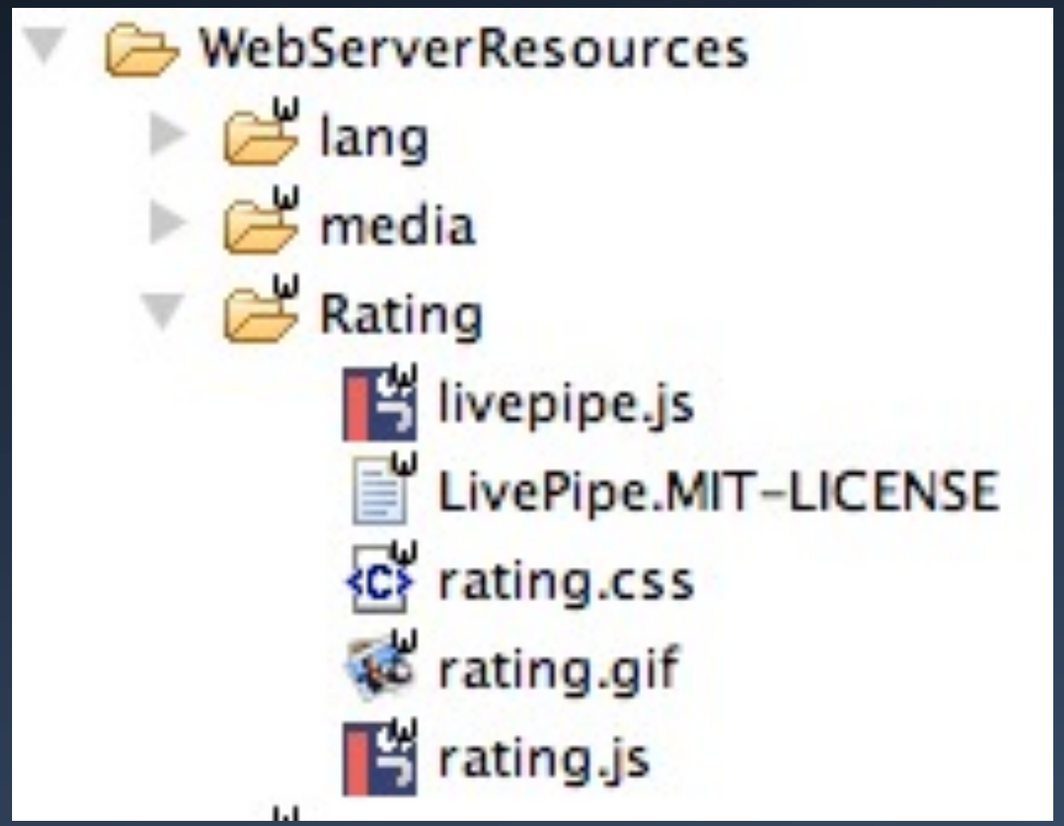
- Determine JavaScript files needed
- Often component specific and supporting

# Step 4: Download Resources

- Get example HTML from demo source
- CSS files
- Image resources
- License file
- Other resources?

# Step 5: Start Small

- AjaxComponent or AjaxDynamicElement?
- Ajax, ERCoolComponents, or private?
- Create class and/or .wo
- Add JS, CSS, images, etc to WebServerResources



# Step 6: addRequiredWebResources

```
protected void addRequiredWebResources(WOResponse response, WOContext context) {

    addScriptResourceInHead(context, response, "Ajax", "controls.js");
    addScriptResourceInHead(context, response, "Ajax", "prototype.js");

    addScriptResourceInHead(context, response,
        "ERCoolComponents", "Rating/livepipe.js");
    addScriptResourceInHead(context, response,
        "ERCoolComponents", "Rating/rating.js");
    addStylesheetResourceInHead(context, response,
        "ERCoolComponents", "Rating/rating.css");

}
```

# Step 7: Static appendToResponse

```
public void appendToResponse(WOResponse response, WOContext context) {

  super.appendToResponse(response, context);

  response.appendContentString("<div id=\"rating_one\" class=\"rating_container\">");
  response.appendContentString("</div>");

  response.appendContentString("<script type=\"text/javascript\">");
  response.appendContentString("var rating_one = new Control.Rating('rating_one');");
  response.appendContentString("</script>");

}
```

# Step 8: handleRequest

```
public WOActionResults handleRequest(WORequest request, WOContext context) {

    System.out.println("CCRating " + request.headers());

    return null;
}
```

# Step 9: Initial Testing

- Smoke Test Time!
- Is control rendered correctly?
- Does CSS load?
- Are images visible?
- Does it “work” client side?

# Step 10: Check Client Side Errors

- Use FireBug or Safari Web Inspector
- Look for errors in JavaScript console
- Look for resources that fail to load

# Step 11: Update Paths in CSS

Resource paths in CSS are likely wrong:

```
background-image:url(/stylesheets/rating.gif);
```

These are resolved (by Wonder) relative to the CSS file's location:

```
background-image:url(rating.gif);
```

# Step 12a: First Dynamic Steps

```
public void appendToResponse(WOResponse response, WOContext context) {

  super.appendToResponse(response, context);

  String id = (String) valueForBinding(
      "id",
      ERXWOContext.safeIdentifierName(context, false),
      context.component());

  response.appendContentString("<div ");

  appendTagAttributeToResponse(response, "id", id);
  appendTagAttributeToResponse(response, "class", "rating_container");

  response.appendContentString("></div>");
```

# Step 12b: First Dynamic Steps

```
appendToResponse continued...


response.appendContentString("<script type=\"text/javascript\">");
response.appendContentString("var ");
response.appendContentString(id);
response.appendContentString(" = new Control.Rating('");
response.appendContentString(id);
response.appendContentString("'); </script>");
}
```

# Step 13a: Rating Options

```
<script type="text/javascript">

  var e_1_0_0_1_3_7 = new Control.Rating('e_1_0_0_1_3_7',
          {value: 5,
           min:   2,
           max:   8}
        );

</script>
```

# Step 13b: Rating Options

```
    response.appendContentString("'); </script>");

    response.appendContentString("', ");
    AjaxOptions.appendToResponse(createOptions(context), response, context);
    response.appendContentString("); </script>");
}

protected NSMutableDictionary createOptions(WOContext context) {
    NSMutableArray ajaxOptionsArray = new NSMutableArray();
    ajaxOptionsArray.addObject(new AjaxOption("min", AjaxOption.NUMBER));
    ajaxOptionsArray.addObject(new AjaxOption("max", AjaxOption.NUMBER));
    ajaxOptionsArray.addObject(new AjaxOption("value", AjaxOption.NUMBER));
    return AjaxOption.createAjaxOptionsDictionary(
            ajaxOptionsArray, context.component(), associations());
}
```

# Step 14: Rest of Bindings

```
   ajaxOptionsArray.addObject(new AjaxOption("capture", AjaxOption.BOOLEAN));
ajaxOptionsArray.addObject(new AjaxOption("classNames", AjaxOption.DICTIONARY));

ajaxOptionsArray.addObject(
    new AjaxOption("input", "inputId", null, AjaxOption.STRING));

ajaxOptionsArray.addObject(new AjaxOption("multiple", AjaxOption.BOOLEAN));
ajaxOptionsArray.addObject(new AjaxOption("rated", AjaxOption.BOOLEAN));
ajaxOptionsArray.addObject(new AjaxOption("reverse", AjaxOption.BOOLEAN));
ajaxOptionsArray.addObject(new AjaxOption("updateOptions", AjaxOption.DICTIONARY));

ajaxOptionsArray.addObject(
    new AjaxOption("updateParameterName", "formValueName", null, AjaxOption.STRING));
```

# Step 15a: Server Interaction

```
   ajaxOptionsArray.addObject(
  new AjaxOption("updateParameterName", "formValueName", null, AjaxOption.STRING));

ajaxOptionsArray.addObject(
    new AjaxConstantOption("updateParameterName", "formValueName",
        formValueName(context), AjaxOption.STRING));

ajaxOptionsArray.addObject(new AjaxConstantOption("updateUrl",
    AjaxUtils.ajaxComponentActionUrl(context), AjaxOption.STRING));

protected String formValueName(WOContext context) {
    return (String)valueForBinding(
    "formValueName",
    id(context) + "_value",
    context.component());
```

# Step 15b: Server Interaction

```
public WOActionResults handleRequest(WORequest request, WOContext context) {
  System.out.println("CCRating " + request.headers());
  System.out.println("CCRating " + request.uri());
  System.out.println("CCRating " + request.formValues());

  Object ratingValue = request.formValueForKey(formValueName(context));

  if (ratingValue instanceof String) {
      ratingValue = Integer.valueOf((String)ratingValue);
  }

  setValueForBinding(ratingValue, "value", context.component());

  return null;
}
```

# Step 16: WO Integration: action Binding

```
public WOActionResults handleRequest(WORequest request, WOContext context) {
    Object ratingValue = request.formValueForKey(formValueName(context));
    if (ratingValue instanceof String) {
        ratingValue = Integer.valueOf((String)ratingValue);
    }
    setValueForBinding(ratingValue, "value", context.component());

    // Nothing gets returned to the client from  the CCRating action so we
    // discard any result from firing the action binding
    if (hasBinding("action")) {
        valueForBinding("action", context.component());
    }

    return null;
}
```

# Step 17a: WO Integration: Form Input

```
  protected boolean actAsInput(WOContext context) {
    return booleanValueForBinding("actAsInput", true, context.component());
}

protected NSMutableDictionary createOptions(WOContext context) {
...
    if ( ! actAsInput(context)) {
        ajaxOptionsArray.addObject(new AjaxConstantOption("updateUrl",
            AjaxUtils.ajaxComponentActionUrl(context), AjaxOption.STRING));
    }
    else {
        ajaxOptionsArray.addObject(new AjaxConstantOption("input",
            id(context) + "_input", AjaxOption.STRING));
    }
```

# Step 17b: WO Integration: Form Input

```
   public void appendToResponse(WOResponse response, WOContext context) {
...
if (actAsInput(context)) {
    response.appendContentString("<input ");

    appendTagAttributeToResponse(response, "type", "hidden");

    appendTagAttributeToResponse(response, "id", id + "_input");
    appendTagAttributeToResponse(response, "name", formValueName(context));

    appendTagAttributeToResponse(response, "value",
        valueForBinding("value", context.component()));

    response.appendContentString("/>");
...
}
```

# Step 17c: WO Integration: Form Input

```
    public void takeValuesFromRequest(WORequest request, WOContext context) {
        if (actAsInput(context)) {
            setValueFromFormValue(request, context);
        }
        super.takeValuesFromRequest(request, context);
    }

    protected void setValueFromFormValue(WORequest request, WOContext context) {
        Object ratingValue = request.formValueForKey(formValueName(context));
        if (ratingValue instanceof String) {
            ratingValue = Integer.valueOf((String)ratingValue);
        }
        setValueForBinding(ratingValue, "value", context.component());
    }
```

# Step 18: Clean and Document

- Remove debugging code
- General code clean-up and refactoring
- JavaDocs for class
- JavaDocs for methods, @param, @return

# Step 19: @binding JavaDocs

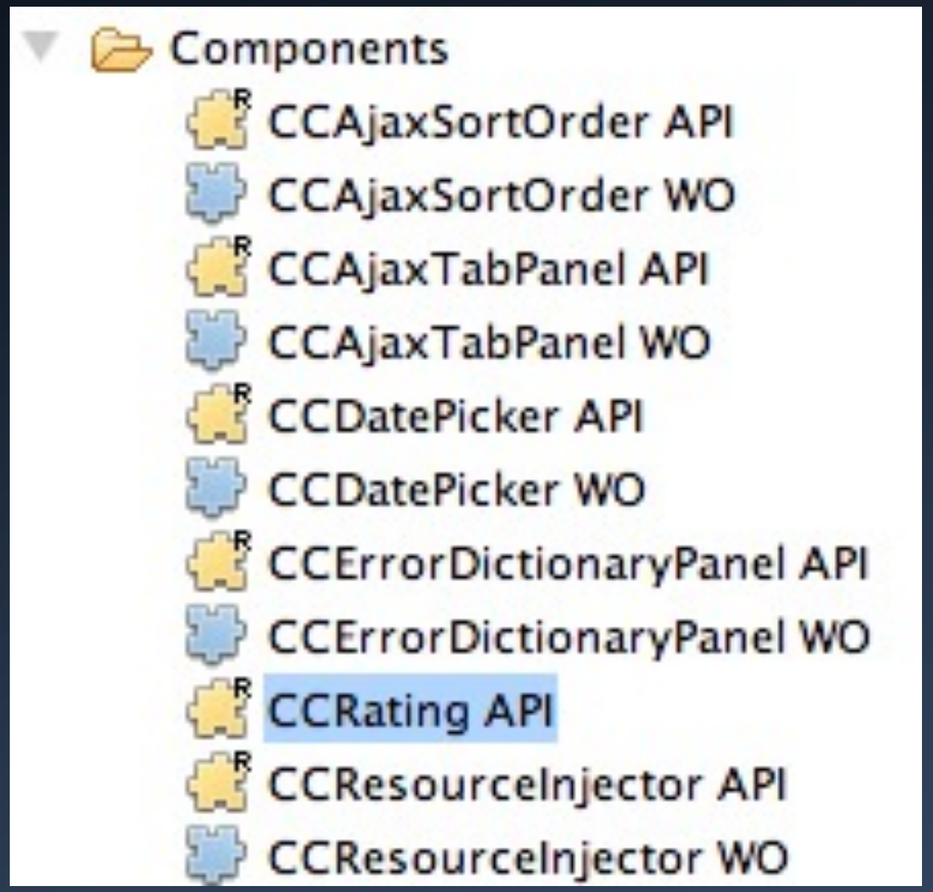
```
    * @binding value the value to show in the ratings widget and the value set
                   when the user selects a different rating

* @binding actAsInput optional, default is <code>true</code>, if false updates
                      the value binding when clicked and optionally calls action
                      method

* @binding action optional, action method to fire when rating changed.  Ignored
                  if actAsInput is <code>true</code> or unbound

* @binding min optional, the value sent to the server when the lowest rating is
               selected, indirectly controls the number of rating points displayed
```

# Step 20: Create .api File




# Q&A

Ajax.framework: Under the Hood

Chuck Hill
Global Village Consulting