Project Dynamic Faces: Using DynaFaces

- Ed Burns
- Senior Staff Engineer
- Enterprise Java Platforms

Agenda

- Design Details
  - Overview
  - The importance of the lifecycle
  - Views and Partial Views
  - JSF lifecycle review
- Application Setup
- Usage Patterns
  - AjaxZone usage
  - DynaFaces.fireAjaxTransaction usage
  - DynaFaces. installDeferredAjaxTransaction usage
  - Dispatching JSF Events from Ajax

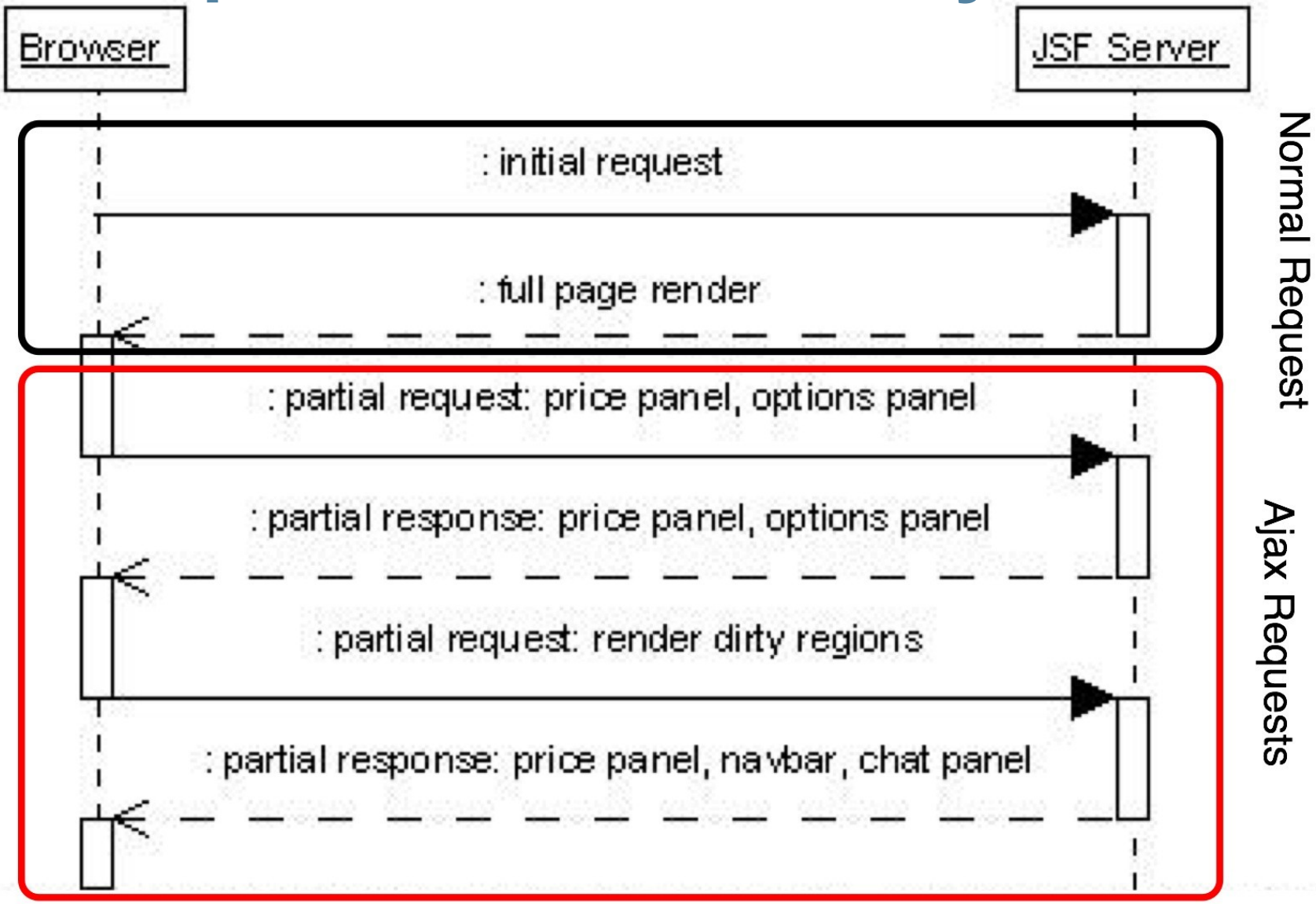
DynaFaces — Overview

- Useful for updating a JSF View in the browser without requiring a full page refresh.
- Intended for situations where you want do more on one page, without having the user go to a new page.
- Doesn't preclude tradtional “page flow” based applications, can do page transitions via AJAX.
- Expose the JSF lifecycle to partial view updates, initiated and handled via AJAX.
- Support for firing Faces Events from the browser directly.

The Importance of the Lifecycle

- Initial request to JSF Server goes through normal lifecycle.
- AJAX requests go through normal lifecycle, but only desired subviews get processed.
- Server can dynamically add and remove subviews during the lifecycle, allowing for a true “dirty region”
- Client can suggest distinct sets of subviews for “execute” and “render”.
- Server sends back XML describing subviews to be refreshed.

The Importance of the Lifecycle



Agenda

- Design Details
  - Overview
  - The importance of the lifecycle
  - Views and Partial Views
  - JSF lifecycle review
- Application Setup
- Usage Patterns
  - AjaxZone usage
  - DynaFaces.fireAjaxTransaction usage
  - DynaFaces. installDeferredAjaxTransaction usage
  - Dispatching JSF Events from Ajax

Views and Partial Views

CarStore

Sun Java(TM) ... | CarStore | CarStore | An overview ... | Using DynaFa...

Duke's Vulgar Luxury Car

Those Joneses next door have everything, don't they? Well, show them that you're the one pulling in the big bucks by driving up in this loaded chassis.

Base Price 20700
Your Price 22650

Buy

Options and Packages

Custom | Standard | performance | Deluxe

Engine: V8

Brakes: disc / drum

Suspension: performance

Speakers: 4 / 6

Audio: Standard / premium

Transmission: manual

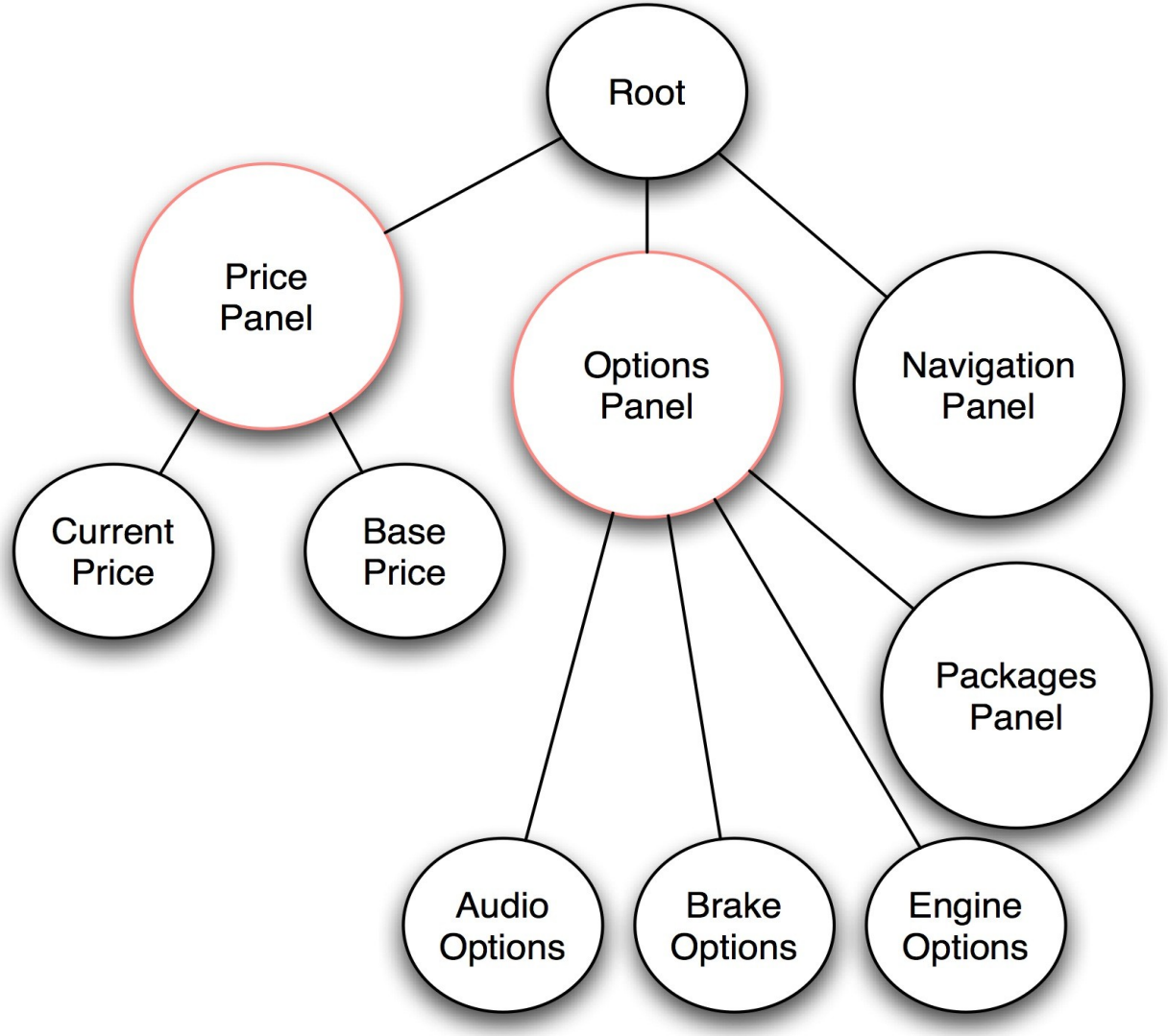
Other Options

- [x] Sunroof
- [x] Cruise Control
- [x] Keyless Entry
- [ ] Security System
- [ ] Ski Rack
- [x] Tow Package
- [ ] GPS

Recalculate | Buy

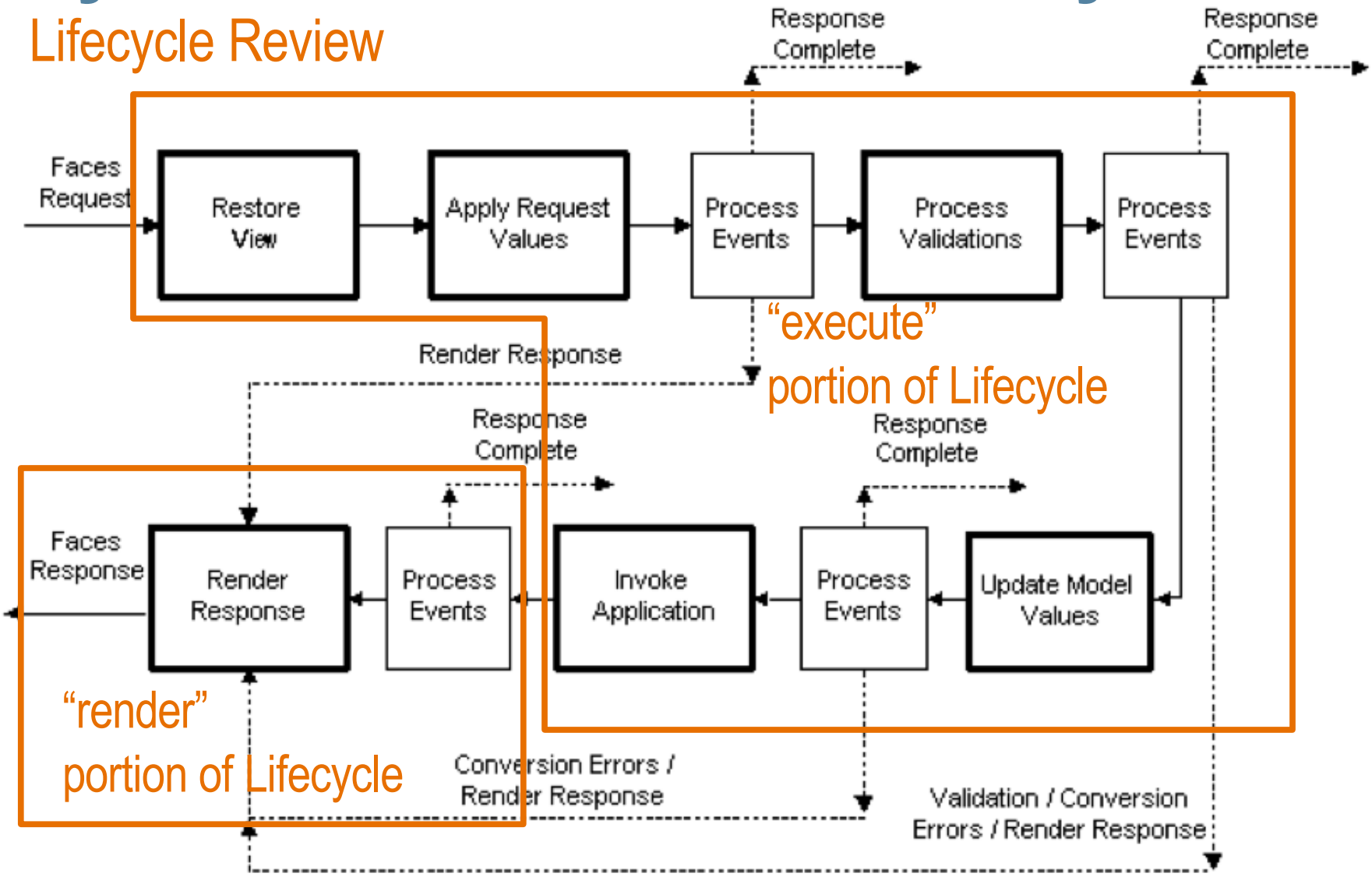
Documentation for this demo

Views and Partial Views



DynaFaces and the JSF Lifecycle

Lifecycle Review



Agenda

- Design Details
  - Overview
  - The importance of the lifecycle
  - Views and Partial Views
  - JSF lifecycle review
- Application Setup
- Usage Patterns
  - AjaxZone usage
  - DynaFaces.fireAjaxTransaction usage
  - DynaFaces. installDeferredAjaxTransaction usage
  - Dispatching JSF Events from Ajax

DynaFaces Application Setup

web.xml sugar

- Add an `<init-param>` element to your `<servlet>` element for the FacesServlet

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet
    </servlet-class>
  <init-param>
    <param-name>javax.faces.LIFECYCLE_ID</param-name>
    <param-value>com.sun.faces.lifecycle.PARTIAL
      </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

DynaFaces Application Setup

Dependency Jars

- Use the dynafaces-facelets-blank.war, or dynafaces-jsp-blank.war as a starter.

—OR—

- Put in WEB-INF/lib

  jsf-extensions-dynafaces.jar

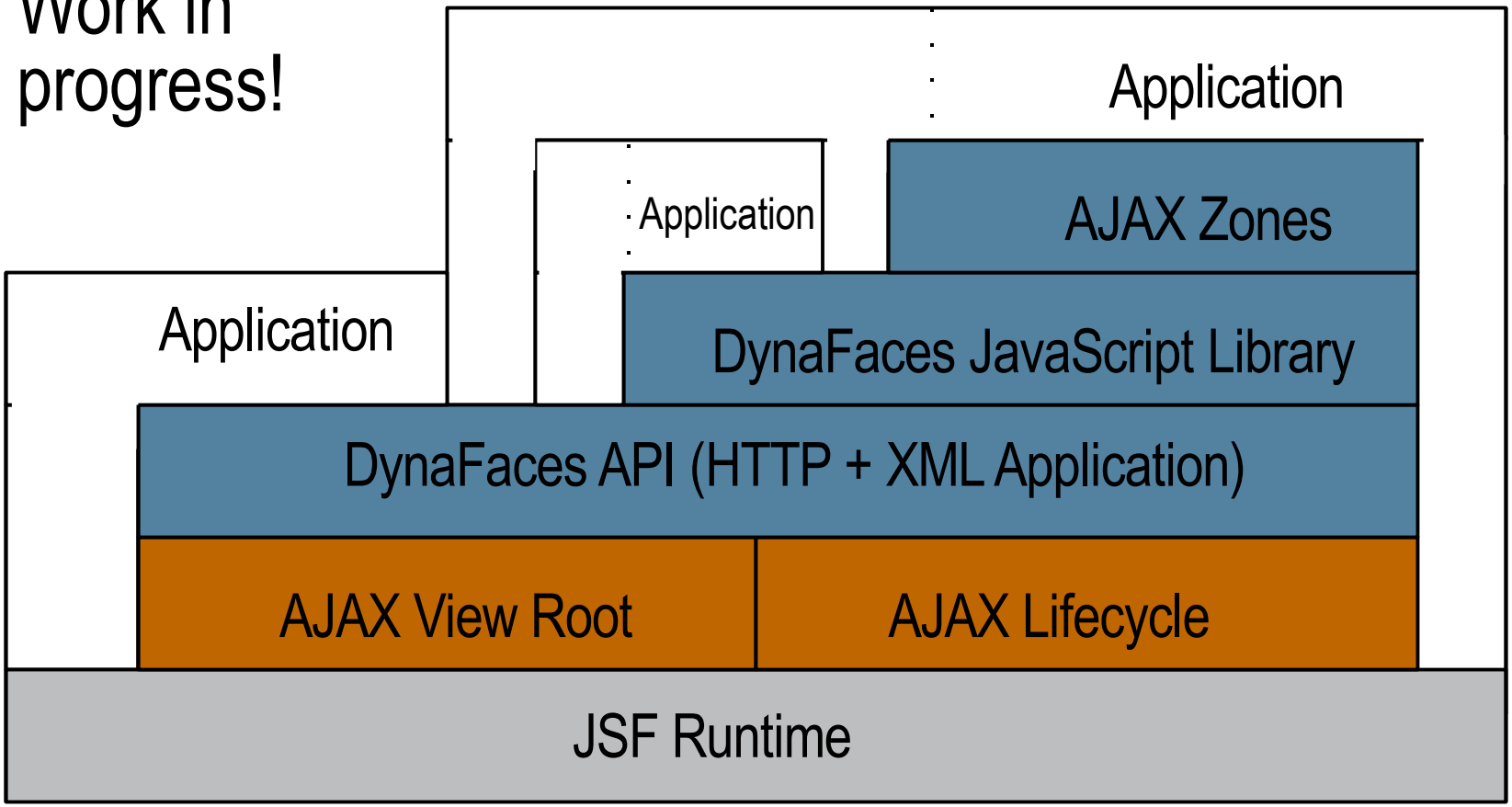
  jsf-extensions-common.jar

  shale-remoting.jar (and dependencies)

  - commons-beanutils.jar
  - commons-chain.jar
  - commons-codec.jar
  - commons-collections.jar
  - commons-digester.jar
  - commons-el.jar
  - commons-fileupload.jar
  - commons-logging.jar

DynaFaces — Usage Patterns

- Several entry points
- Work in progress!



DynaFaces — Usage Patterns

Increasing Complexity

- Page Author
  - Use AJAX enabled components
  - Use AjaxZone tag to AJAXify regions of the page
  - Use provided JavaScript library to AJAXify page elements and components
- Component Author
  - Use provided JavaScript library in custom components
  - Write your own JavaScript that talks directly to the HTTP protocol and the XML application defined by DynaFaces

Increasing Complexity

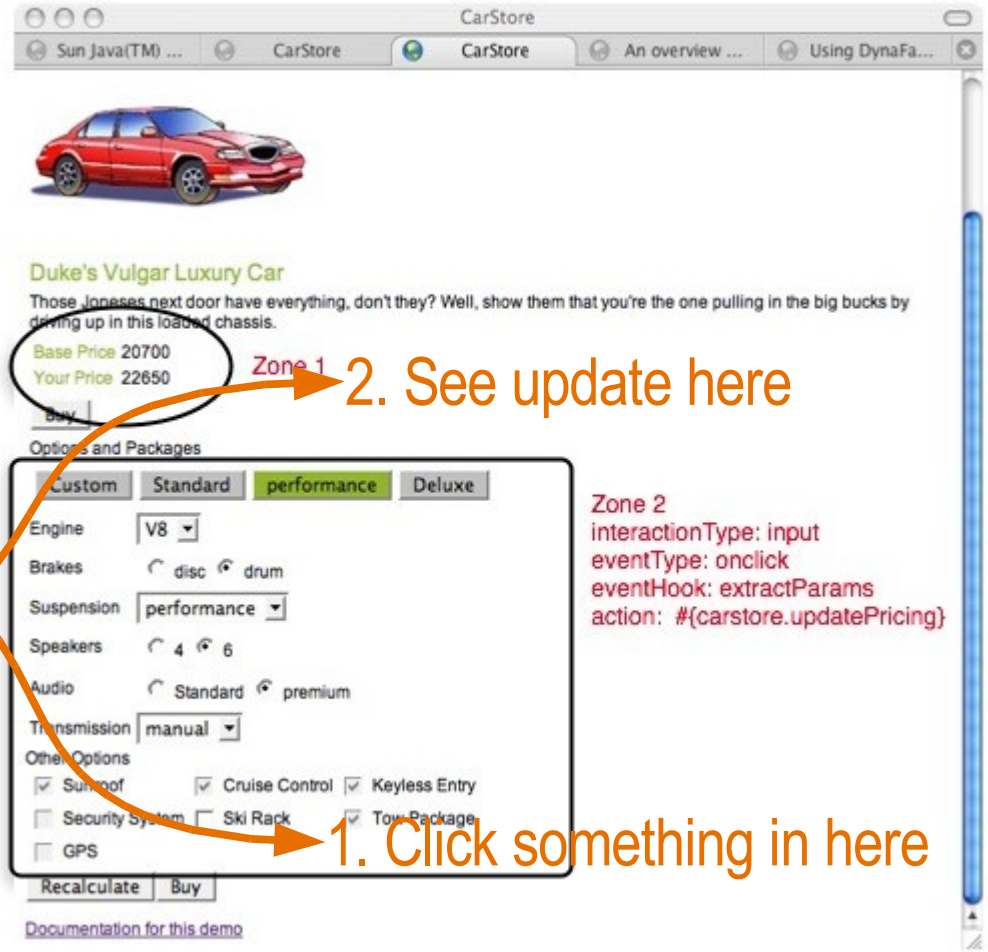
Agenda

- Design Details
  - Overview
  - The importance of the lifecycle
  - Views and Partial Views
  - JSF lifecycle review
- Application Setup
- Usage Patterns
  - AjaxZone usage
  - DynaFaces.fireAjaxTransaction usage
  - DynaFaces. installDeferredAjaxTransaction usage
  - Dispatching JSF Events from Ajax

DynaFaces Usage Patterns

Using AjaxZones

- The easiest way to AJAXify an existing application
- Demarcate one or more AJAX zones within a page
- Zones will refresh via AJAX, without full page refresh.
- Action in one zone causes reaction in another zone



Demonstration

DynaFaces Usage Patterns

Using AjaxZones

```
<jsfExt:ajaxZone id="zone1"> <h:panelGrid columns="2">

Base Price <h:outputText binding="#{currentModel.basePrice}"/>

Your Price <h:outputText value="#{currentModel.currentPrice}"/>
   </h:panelGrid> </jsfExt:ajaxZone>

<jsfExt:ajaxZone id="zone2"
   action="#{carstore.currentModel.updatePricing}">

Option Packages <h:panelGrid columns="4">

   <h:commandButton value="Custom"
   actionListener="#{carstore.choosePackage}"/>

   <h:commandButton
   value="Standard"actionListener="#{carstore.choosePackage}"/>

</h:panelGrid> <h:panelGrid columns="2">

Engine <h:selectOneMenu binding="#{currentModel.components.engine}"/>

Breaks <h:selectOneRadio binding="#{currentModel.components.brake}"/>

Suspension
   <h:selectOneMenubinding="#{currentModel.components.suspension}"/>

Speakers <h:selectOneRadio
   binding="#{currentModel.components.speaker}"/>

</jsfExt:ajaxZone>
```

DynaFaces Usage Patterns

Using AjaxZones — available attributes

- `action` (optional)
  - MethodExpression to invoke when the request processing lifecycle in which this zone is being processed reaches its invokeApplication phase.
- `immediate` (optional): Just like commandButton
- `inspectElement`: (optional)
  - User defined JavaScript function that takes an HTML element and returns `true` or `false` depending on whether or not this element should be AJAXified. The default `inspectElement` function will return `true` for every child of this zone that is an HTML `input`, `button`, or `option` tag.
- `eventType`: (optional)JavaScript event to cause the AJAX request. If not specified, defaults to “click”.

DynaFaces Usage Patterns

Using AjaxZones — available attributes

- `collectPostData` (optional)
  - User defined JavaScript function called when the `eventType` event occurs. Extracts name=value pairs to send in AJAX request. If not specified, the following name=value pairs are sent
    - The name=value pair of any input field within the zone.
    - For any radio button or menu, the name=value pair of the currently selected choice.
    - If the activated component that resulted in this callback being called is a button, the name=value pair for that button only. Any other buttons within the zone do not have their name=value pairs contributed to the AJAX request.
- `postReplace` (optional)
  - This optional attribute names a function to be called after the new content from the server for this zone has been installed into the view. Used for jMaki support.

DynaFaces Usage Patterns

Using AjaxZones — available attributes

- `replaceElement` (optional)
  - This optional attribute is or names a JavaScript function that will be called when the system needs to replace a chunk of markup in the view based on the return from the server. The default implementation is sufficient for most cases.
- `getCallbackData` (optional)
  - This optional attribute names a function to be called to provide a closure argument that will be passed to the ajax request and made available to the ajax response in the replaceElement or postReplace functions.

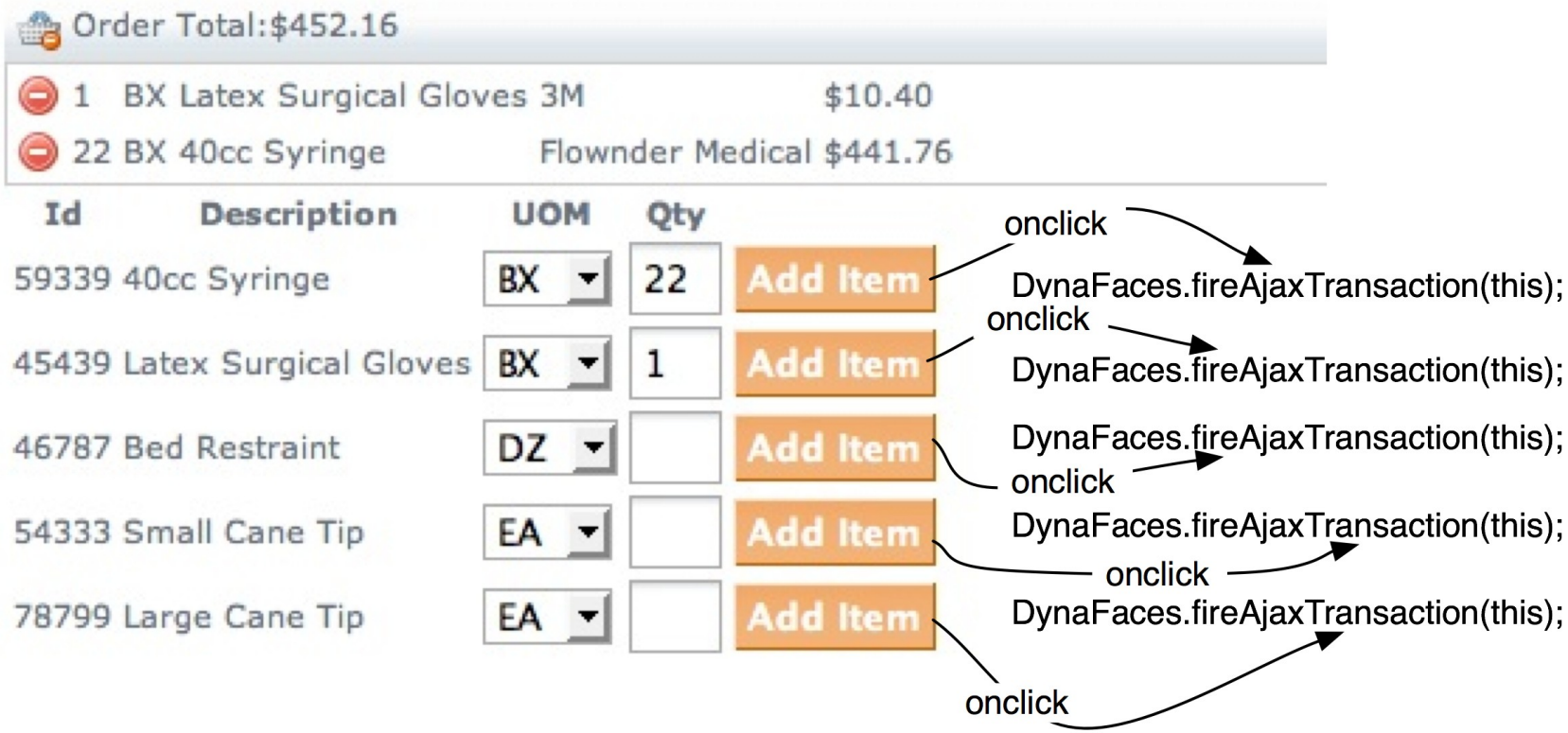
Agenda

- Design Details
  - Overview
  - The importance of the lifecycle
  - Views and Partial Views
  - JSF lifecycle review
- Application Setup
- Usage Patterns
  - AjaxZone usage
  - DynaFaces.fireAjaxTransaction usage
  - DynaFaces. installDeferredAjaxTransaction usage
  - Dispatching JSF Events from Ajax

DynaFaces Usage Patterns

Using DynaFaces.fireAjaxTransaction

- Defined in built-in JavaScript library.
- When called, causes an AJAX transaction to the Faces server.
- Many options for customizing the transaction.



Demonstration

DynaFaces Usage Patterns

Using DynaFaces.fireAjaxTransaction

```
<h:commandButton value="Add Item"
  action="#{orderEntry.addProduct}"
  onclick="new
  DynaFaces.fireAjaxTransaction(this);"/>
```

- Useful when you want the AJAX transaction to happen as soon as the script is executed.
- Default action just does a refresh of the whole view via AJAX, using JavaScript DOM methods to update the elements.
- Can add options to choose exactly which subtrees get sent, processed, and re-rendered.

DynaFaces Usage Patterns

DynaFaces.fireAjaxTransaction General Form

- Generally used in a tag attribute

```
<ANY_HTML_OR_JSF_ELEMENT
    on|EVENT|="DynaFaces.fireAjaxTransaction(this,{|
  OPTIONS|});" />
```

- Where
  - `ANY_HTML_OR_JSF_ELEMENT` is any HTML element or JSF tag
  - `on|EVENT|` is any JavaScript event type, such as `onclick`
  - `{|OPTIONS|}` is an optional argument. If present, it is a JavaScript associative array containing any options desired for this transaction.
- `DynaFaces.fireAjaxTransaction()` may be used from non event-handler JavaScript as well.

DynaFaces Usage Patterns

Using DynaFaces.fireAjaxTransaction

- `execute` (optional)
  - Comma separated list of client ids to be traversed on the “execute” portion of the JSF Request Processing Lifecycle (everything but render). If not specified, the value of the render option is used. “none” indicates that the view must not be traversed during the execute portion of the lifecycle.
- `render` (optional)
  - Comma separated list of client ids to be traversed on the “render” portion of the JSF Request Processing Lifecycle. If not specified it's up to the server to decide what to re-render. By default the whole view is re-rendered. “none” indicates that the view must not be rendered.
- `inputs` (optional)
  - Comma separated list of clientIds for which the value should be sent in the AJAX request. If not specified, all input components in the current form are submitted.

DynaFaces Usage Patterns

Using DynaFaces.fireAjaxTransaction — options

`postReplace` (optional)

User defined JavaScript function called after element replacement. Useful when integrating with jMaki. If not specified, any scripts present in the markup to be rendered are evaluated.

`replaceElement` (optional)

User defined JavaScript function called for element replacement. If not specified, default element replacement occurs.

`getCallbackData` (optional)

User defined JavaScript function, returns closure object that is passed to the `replaceElement` or `postReplace` functions.

`immediate` (optional)

boolean value that tells the server to set the immediate option, for this transaction only, on any input or command components in the traversal.

DynaFaces Usage Patterns

Using DynaFaces.fireAjaxTransaction — options

`asynchronous` (optional)

Should this transaction be asynchronous (the default) or synchronous

`xjson` (optional)

Any JSON data that should be sent along with the transaction, as the value of the `X-JSON` header

`closure` (optional)

Closure object that is passed to the `replaceElement` or `postReplace` functions.

`immediate` (optional)

boolean value that tells the server to set the immediate option, for this transaction only, on any input or command components in the traversal.

Agenda

- Design Details
  - Overview
  - The importance of the lifecycle
  - Views and Partial Views
  - JSF lifecycle review
- Application Setup
- Usage Patterns
  - AjaxZone usage
  - DynaFaces.fireAjaxTransaction usage
  - DynaFaces.installDeferredAjaxTransaction usage
  - Dispatching JSF Events from Ajax

JSF, The Blueprints Scroller, and jMaki

Account Id	Customer Name	Symbol	Total Sales
120	ffffff	symbol_120	120.0
121	777777777777	symbol_121	121.0
122	name_122	symbol_122	122.0
123	sds	symbol_123	123.0
124	name_124	symbol_124	124.0
125	name_125 ok cancel	symbol_125	125.0
126	name_126	symbol_126	126.0
127	name_127	symbol_127	127.0
128	name_128	symbol_128	128.0
129	name_129	symbol_129	129.0

clientId: form:table

Result Page: Previous 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 Next

clientId: form:scroller

For each anchor *e* in the scroller, do:

```
DynaFaces.installDeferredAjaxTransaction(e,
'mousedown', { postReplace: 'postReplace',
   render: 'form:table,form:scroller' });
```

Demonstration

DynaFaces Usage Patterns

Using DynaFaces.installDeferredEvent

- "Extends" DynaFaces.fireAjaxTransaction to provide deferred kickoff of AJAX transaction when an arbitrary JavaScript event occurs.
- Defined in built-in JavaScript library. Used by AjaxZones.
- Can be installed on any DOM element to cause an AJAX transaction to start when a given JavaScript event happens.
- Options are the same as for DynaFaces.fireAjaxTransaction

DynaFaces Usage Patterns

Using DynaFaces.installDeferredEvent

```
<script type='text/javascript'>
document.forms[0].submit = function() {};
var a = $('form:scroller').getElementsByTagName('a');
$A(a).each(function(e) {
new DynaFaces.installDeferredAjaxTransaction(e, 'mousedown',
  { postReplace: 'postReplace',
    render: 'form:table,form:scroller });
    });
</script>
```

- Globally scoped script in the page.
- Happens to use “prototype” library, but need not do so.
- For each anchor element in the scroller, call `new DynaFaces.installDeferredAjaxTransaction()`, passing the anchor element.

Agenda

- Design Details
  - Overview
  - The importance of the lifecycle
  - Views and Partial Views
  - JSF lifecycle review
- Application Setup
- Usage Patterns
  - AjaxZone usage
  - DynaFaces.fireAjaxTransaction usage
  - DynaFaces. installDeferredAjaxTransaction usage
  - Dispatching JSF Events from Ajax

DynaFaces Usage Patterns

Dispatching JSF Events via Ajax

ActionEvent Example

Submit queued actionEvents. submit via ajax ← 2. press this button once.

ActionListener Output: [ajax] [ajax] [ajax] ← 3. See these appear via AJAX.

Queue additional ActionEvents. queue action ← 1. Press this button 3 times.

- Allow queuing of arbitrary `javax.faces.event.FacesEvent` subclasses from JavaScript directly into JSF Lifecycle.
- JavaScript classes for standard `ValueChangeEvent` and `ActionEvent` classes.
- Use JavaScript “subclassing” for custom events.

Demonstration

DynaFaces Usage Patterns

Dispatching JSF Events via Ajax—`ActionEvent`

```
<script type='text/javascript'>
   function queueEvent() {
       var actionEvent = new DynaFaces.ActionEvent("ajax",
       DynaFaces.PhaseId.INVOKE_APPLICATION);
       DynaFaces.queueFacesEvent(actionEvent);
       return false;
   }

</script>

   <h:commandButton id="ajax" value="submit via ajax"
   onclick="DynaFaces.fireAjaxTransaction(this, { render: 'label'});
   return false;" actionListener="#{bean.processAction}" />


   <h:outputText id="label" value="#{requestScope.actionEvents}" />


   <input type="submit" name="queueAction" id="queueAction"
   value="queue action" onclick="queueEvent(); return false;" />
```

- Non-JSF button used to queue action events.
- JSF commandButton used to fire the AJAX transaction. Requests re-render of the `label` component.
- `actionListener` updates request scoped property `actionEvents`, which is output from the `label` component.

DynaFaces Usage Patterns

Dispatching JSF Events via Ajax-`ValueChangeEvent`

```
<script type='text/javascript'>
   function queueEvent() {
       var valueChangeEvent = new DynaFaces.ValueChangeEvent("input",
               DynaFaces.PhaseId.UPDATE_MODEL_VALUES,
               “oldValue”, “newValue”);
       DynaFaces.queueFacesEvent(valueChangeEvent);
       return false;

</script>

<h:inputText id="input" valueChangeListener="#{bean.valueChange}" />

<h:commandButton value="submit via ajax"
   onclick="DynaFaces.fireAjaxTransaction(this, { execute: 'input',
   render: 'label,input', inputs: 'input' }); return false;"/>

<h:outputText id="label" value="#{requestScope.valueChangeEvents}" />

<input type="submit" name="newValue" id="newValue" value="queue event"
   onclick="queueEvent(); return false;" />
```

- Non-JSF button used to queue value change events.
- JSF commandButton to fire the AJAX transaction. Requests execution of `input` component, and re-render of the `label` component. The `inputs` attribute is used to cause only the `input` component to be submitted.
- `valueChangeListener` updates request scoped property `valueChangeEvents`, which is output from the `label` component.

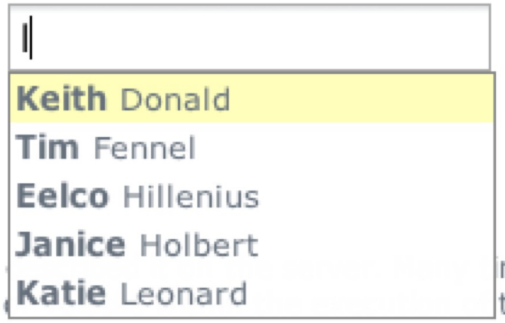
DynaFaces Usage Patterns

Dispatching JSF Events via Ajax-Custom Event Class

Auto Suggest

The token AJAX widget for frameworks. This example shows how a developer can easily customize the presentation of the widget, just as they would with any other content on the page with server-side templating.

1. Type into this field.
2. 'onkeypress' event fires.
3. Custom event is instantiated, queued, and a DynaFaces Ajax Transaction is initiated
4. 'onComptele' is called, and text updates.



Source

Contextual Auto Suggest

Ideally, we want everything to work just as we mes you can run into issues with referencing/using variables that are contextual the page. The example below shows that this isn't an issue for JavaServer Faces.

- Uses JavaScript “subclassing” for custom events.
- Combined with a custom component `<e:serverSuggest />`
- This component overrides the standard `UIComponent.broadcast(FacesEvent event)` to look for an instance of `SuggestEvent`, from which it extracts the submitted information and dispatched to the Renderer.

Demonstrations

DynaFaces Usage Patterns

Dispatching JSF Events via Ajax-Custom Event Class

- Add event definition to web.xml

```
<context-param>
   <param-name>com.sun.faces.extensions.avatar.FacesEvents
   </param-name>
   <param-value>
   SuggestEvent:com.enverio.jsf.SuggestEvent:com.enverio.jsf.UISuggest
   </param-value>

</context-param>
```

- "subclass" the base FacesEvent in JavaScript:

```
Enverio.SuggestEvent = function(clientId, phaseId) {
   this.base = DynaFaces.FacesEvent;
   this.base("SuggestEvent", clientId, phaseId);

}

Enverio.SuggestEvent.prototype = new DynaFaces.FacesEvent;
```

- Queue the event, and fire the transaction, as you would for any FacesEvent:

```
  var suggest = new Enverio.SuggestEvent(elementId,
     DynaFaces.PhaseId.RENDER_RESPONSE);
   DynaFaces.queueFacesEvent(suggest);
   DynaFaces.fireAjaxTransaction(this.element, this.options);
```

Summary

- DynaFaces is easy to use for simple “Ajaxification”.
- DynaFaces lets you add AJAX to your site without writing any JavaScript.
- If you are not against writing JavaScript, DynaFaces allows you to do very powerful things with JSF and AJAX.
- DynaFaces leverages the strengths of JSF while extending the richness of your application via AJAX.

DynaFaces

Ed Burns

ed.burns@sun.com