Project Dynamic Faces™
DynaFaces under the hood

- Ed Burns
- Senior Staff Engineer
- Enterprise Java Platforms

Agenda

- Design Details
  - > Overview
  - > JSF lifecycle review
  - > How to direct the lifecycle
  - > XML response
- Application Setup
- Usage Patterns
  - > Direct HTTP Interaction
  - > DynaFaces.fireEvent usage
  - > DynaFaces. installDeferredEvent usage
  - > AjaxZone usage

DynaFaces — Overview

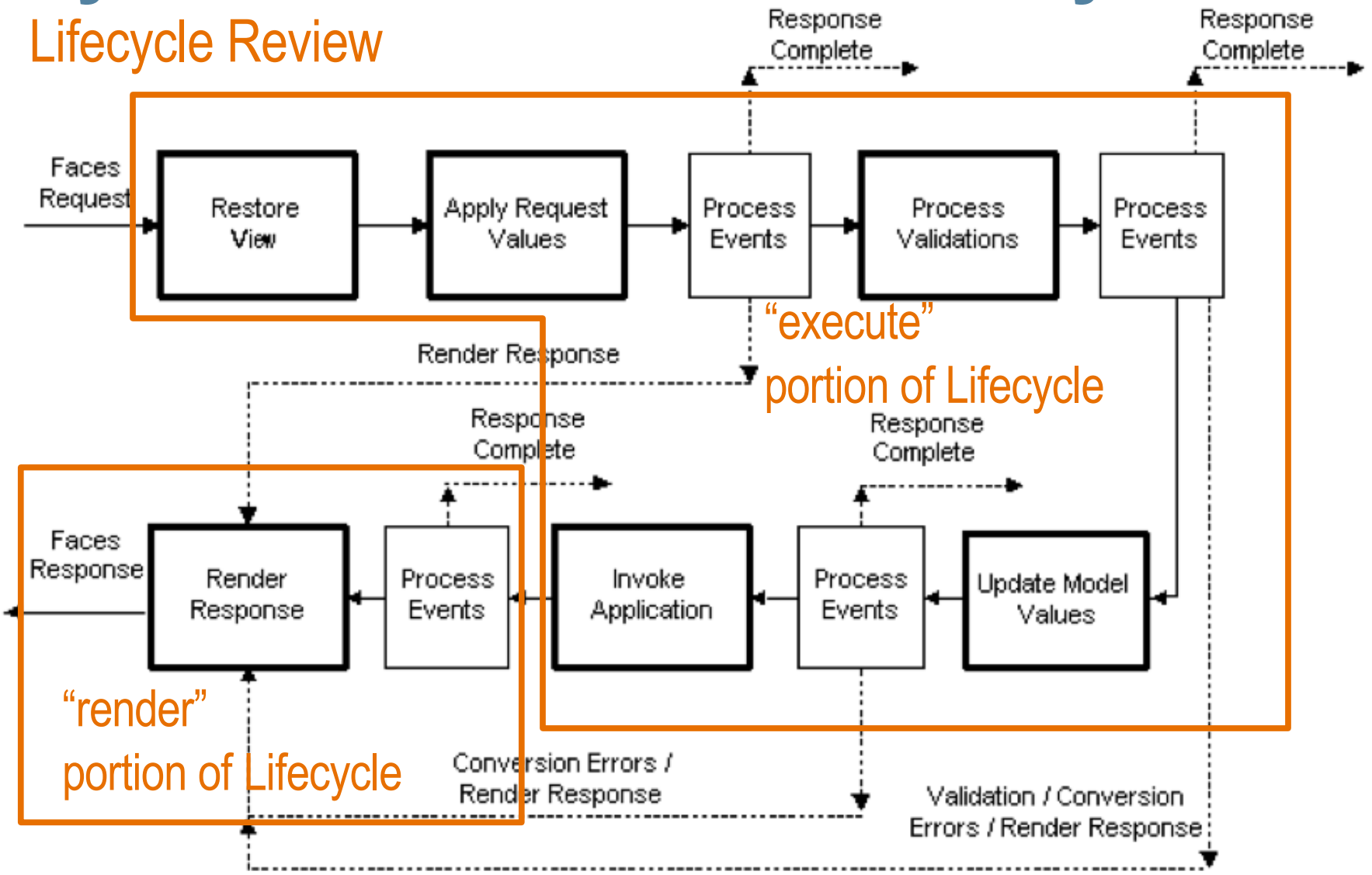
- Useful for updating a JSF View in the browser without requiring a full page refresh.
- Intended for situations where you want do more on one page, without having the user go to a new page.
- Doesn't preclude tradtional “page flow” based applications.
- Expose the JSF lifecycle to partial view updates, 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 request 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.

DynaFaces and the JSF Lifecycle

Lifecycle Review



DynaFaces XML Application

```
<partial-response>
  <components>
    <render id="form:table"/>
      <markup><![CDATA[Rendered content from component]]></markup>
      <messages>
        <message>The messages element is optional.  If present,
                 it is a list of FacesMessage.getSummary() output
        </message>
      </messages>
    </render>
    <!-- repeat for the appropriate number of components -->
    <messages><!-- global messages --></messages>
  </components>
  <state><![CDATA[state information for this view ]]></state>
</partial-response>
```

What to do with the XML response

- For each `<render>` element within the `<components>` element:
  - Get the `CDATA` element from inside the `<markup>` element inside the `<render>` element.
  - Strip out any JavaScript
  - Locate the corresponding DOM element by id.
  - Replace the DOM element with the markup from the XML response.
  - If desired, evaluate the JavaScript in the markup.
  - If the `<render>` element has a `<messages>` element, render the messages for that DOM element appropriately.

What to do with the XML response (continued)

- If there is a global `<messages>` element, render the messages within it appropriately.
- Extract the `CDATA` from the `<state>` element and replace the value of the `javax.faces.ViewState` hidden fields with this value.

Agenda

- Design Details
  - Overview
  - JSF lifecycle review
  - How to direct the lifecycle
  - XML response
- Application Setup
- Usage Patterns
  - Direct HTTP Interaction
  - DynaFaces.fireEvent usage
  - DynaFaces.installDeferredEvent usage
  - AjaxZone usage

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

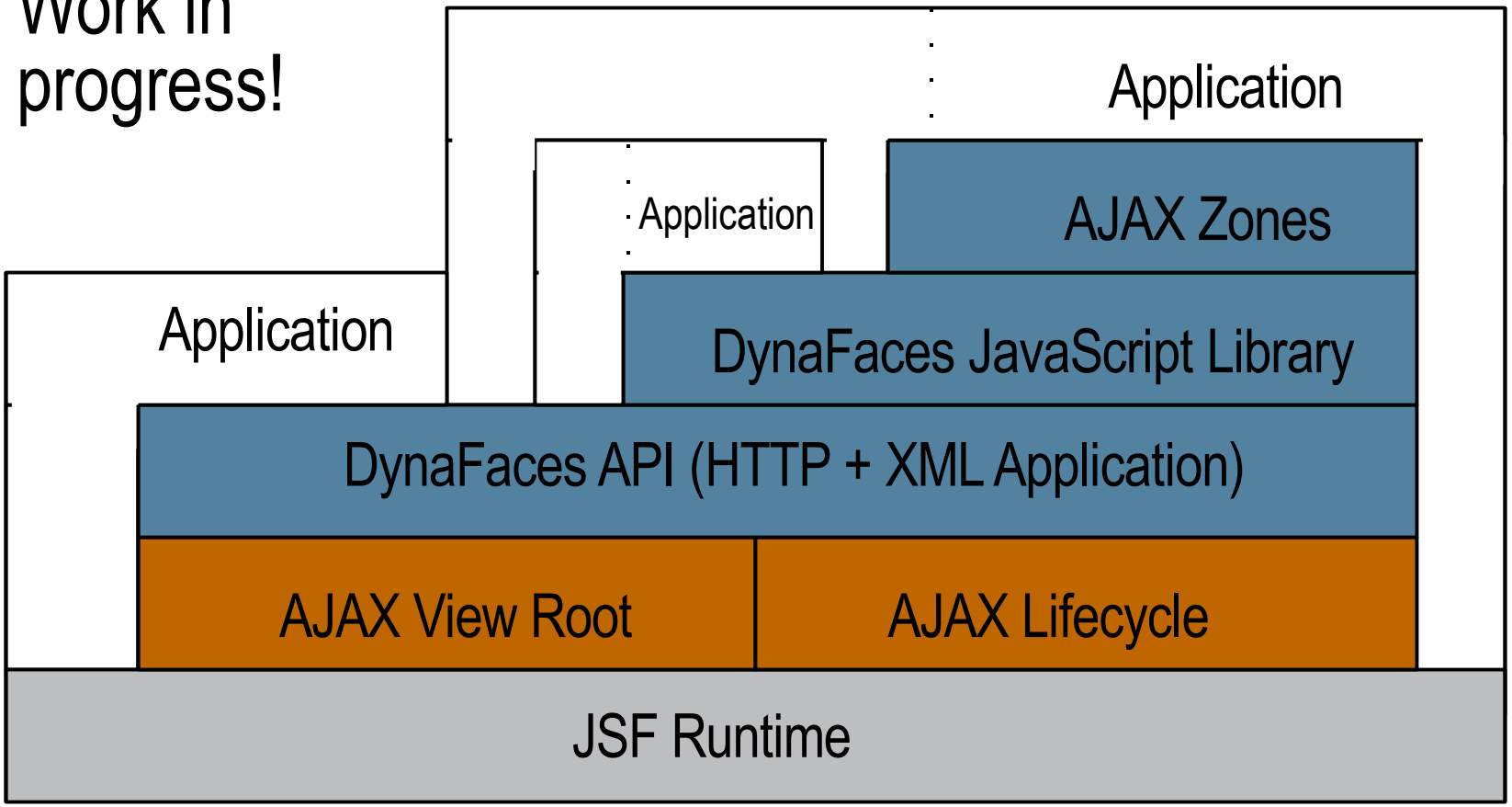
    commons-logging.jar

Agenda

- Design Details
  - Overview
  - JSF lifecycle review
  - How to direct the lifecycle
  - XML response
- Application Setup
- Usage Patterns
  - Direct HTTP Interaction
  - DynaFaces.fireEvent usage
  - DynaFaces.installDeferredEvent usage
  - AjaxZone usage

DynaFaces — Usage Patterns

- Several entry points
- Work in progress!
- Plan to incorporate IceFaces, AJAX4JSF into DynaFaces, then into JSF 2.0



DynaFaces — Usage Patterns

- 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
  - Build composite components with AjaxZones
  - 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

Increasing Complexity

Agenda

- Design Details
  - Overview
  - JSF lifecycle review
  - How to direct the lifecycle
  - XML response
- Application Setup
- Usage Patterns
  - Direct HTTP Interaction
  - DynaFaces.fireEvent usage
  - DynaFaces.installDeferredEvent usage
  - AjaxZone usage

DynaFaces Usage Patterns

Using Direct HTTP Interaction

- Write your own JavaScript to interact with the JSF server directly.
- Requires understanding of HTTP basics and JavaScript XML manipulation techniques.
- Requires understanding of JSF lifecycle (see following slides)
- Usage pattern:
  - Craft an HTTP Request with proper headers and data.
  - Send the request to the JSF server over XmlHttpRequest.
  - Server responds with XML Document in DynaFaces format.
  - Manipulate the XML Document to update the DOM of the currently displayed page.

HTTP Method and Headers

- HTTP Method is generally POST, but can be GET if data is small enough to fit in the query string.
- The following HTTP request headers are defined
  - `com.sun.faces.avatar.Partial`

    Must be present and have a value of “true” (no quotes) on all AJAX requests. JSF lifecycle uses this to tell if the request is an AJAX request or not.
  - `com.sun.faces.avatar.Execute`

    Equivalent to the “execute” option to DynaFaces.fireEvent
  - `com.sun.faces.avatar.Render`

    Equivalent to the “render” option to DynaFaces.fireEvent

Headers (continued) and POST Data

- HTTP Request Headers
  - `com.sun.faces.avatar.Partial`

    Must be present and have a value of “true” (no quotes) on all AJAX requests. JSF lifecycle uses this to tell if the request is an AJAX request or not.
  - `com.sun.faces.avatar.Event`

    Equivalent to the “event” option to DynaFaces.fireEvent
  - `X-JSON`

    JSON data sent to the server, and also returned in response
- POST (or GET) Data
  - Must include “javax.faces.ViewState”
  - Should include name=value pairs for form data.

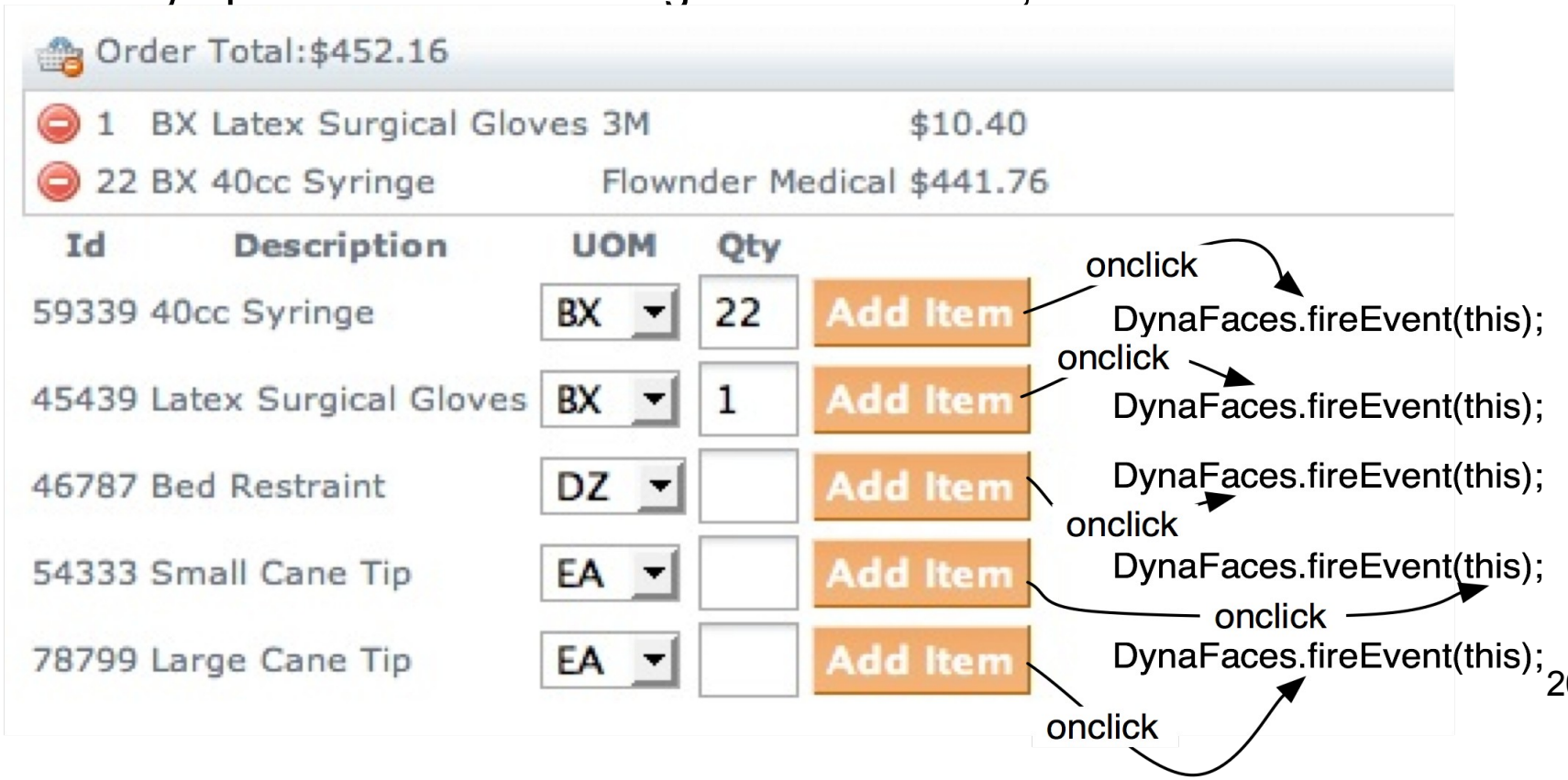
Agenda

- Design Details
  - Overview
  - JSF lifecycle review
  - How to direct the lifecycle
  - XML response
- Application Setup
- Usage Patterns
  - Direct HTTP Interaction
  - DynaFaces.fireEvent usage
  - DynaFaces.installDeferredEvent usage
  - AjaxZone usage

DynaFaces Usage Patterns

Using DynaFaces.fireEvent

- Defined in built-in JavaScript library. Used by AjaxZones.
- When instantiated, causes an AJAX transaction to the Faces server.
- Many options for customizing the transaction, more flexible than zones.



DynaFaces Usage Patterns

Using DynaFaces.fireEvent

```
<h:commandButton value="Add Item"
  action="#{orderEntry.addProduct}"
  onclick="new
  DynaFaces.fireEvent(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.

DynaFaces Usage Patterns

DynaFaces.fireEvent General Form

- Generally used in a tag attribute

```
<ANY_HTML_OR_JSF_ELEMENT
    on|EVENT|="DynaFaces.fireEvent(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 a JavaScript associative array containing any options desired for this transaction

DynaFaces Usage Patterns

Using DynaFaces.fireEvent

- `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.
- `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.
- `inputs` (optional)
  - Comma separated list of clientIds for which the value should be sent in the AJAX request. This is similar to the `collectPostData` attribute on ajaxZone.

DynaFaces Usage Patterns

Using DynaFaces.fireEvent — options (in addition to AjaxZone)

`postReplace` (optional)

User defined JavaScript function called after element replacement. Useful when integrating with jMaki. If not specified, any scripts 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`.

DynaFaces Usage Patterns

Using DynaFaces.fireEvent — options (in addition to AjaxZone)

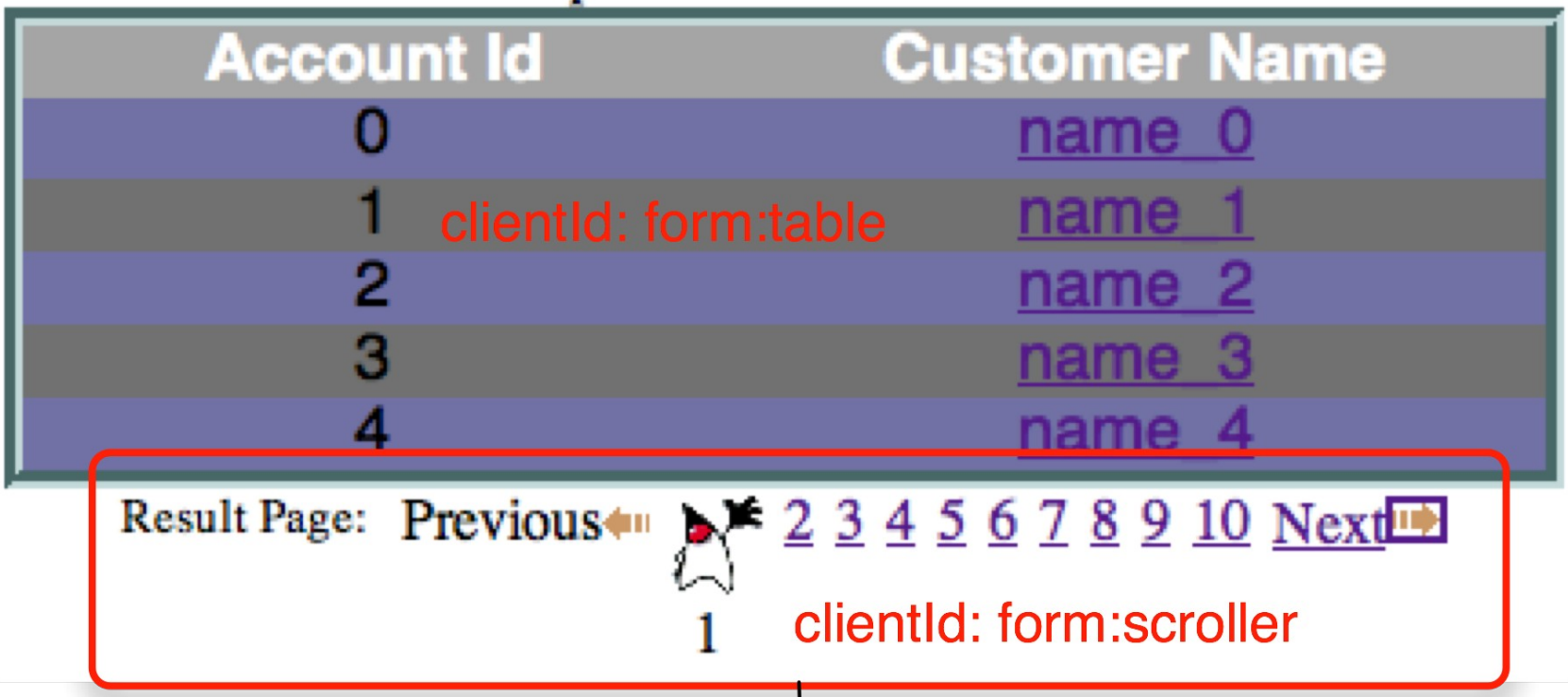
- `event` (optional)
  - If present, this must be the only option. Allows an arbitrary method of an arbitrary component in the view to be invoked, as if a FacesEvent had occurred.
- `closure` (optional)
  - The same intent as the `getClosure` function on AjaxZone, but in this case, the JavaScript object is just passed directly.

Agenda

- Design Details
  - Overview
  - JSF lifecycle review
  - How to direct the lifecycle
  - XML response
- Application Setup
- Usage Patterns
  - Direct HTTP Interaction
  - DynaFaces.fireEvent usage
  - DynaFaces.installDeferredEvent usage
  - AjaxZone usage

The Scroller Component

Rendered via Faces components:



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

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

DynaFaces Usage Patterns

Using DynaFaces.installDeferredEvent

- “Extends” DynaFaces.fireEvent to provide deferred kickoff of AJAX transaction.
- 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.fireEvent

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.fireEvent(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.fireEvent()`, passing the anchor element.

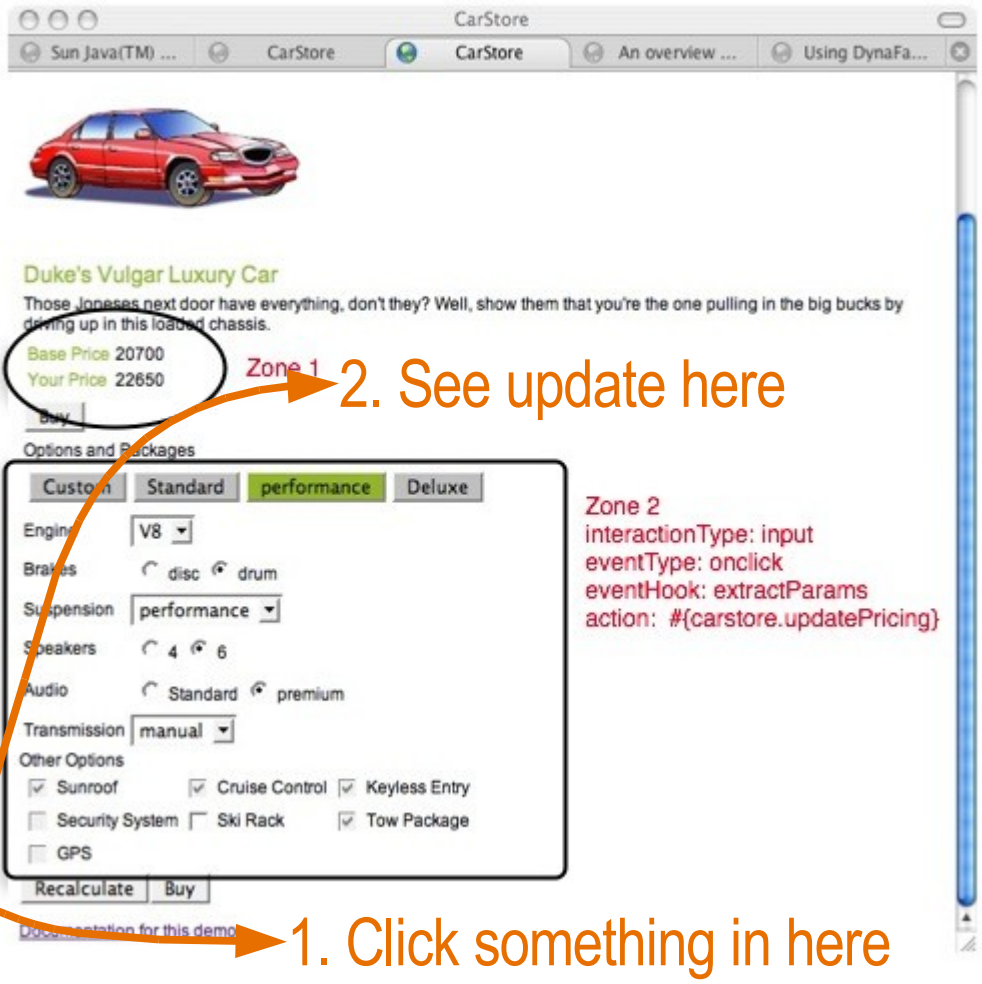
Agenda

- Design Details
  - Overview
  - JSF lifecycle review
  - How to direct the lifecycle
  - XML response
- Application Setup
- Usage Patterns
  - Direct HTTP Interaction
  - DynaFaces.fireEvent usage
  - DynaFaces.installDeferredEvent usage
  - AjaxZone usage

DynaFaces Usage Patterns

Using AjaxZones

- The easiest way to AJAXify an existing application
- Demarcate one or more AJAX zones within a page
- For each zone, provide some helper attributes to inform DynaFaces how to AJAXify the components within that zone.
- Zones will refresh via AJAX, without full page refresh.
- Action in one zone causes reaction in another zone



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" interactionType="input"
   inspectElement="inspectElement"
   eventType="click" collectPostData="extractParams"
   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
- `interactionType` (optional): “input” or “output” depending on what kind of components are in this zone
- `eventType`: JavaScript event to cause the AJAX request, ie “click”

DynaFaces Usage Patterns

Using AjaxZones — available attributes

- `collectPostData` (optional)
  - User defined JavaScript function called when the `eventType` event occurs. Extracts values to send in AJAX request. If not specified, all components in zone are sent.

Demonstrations

DynaFaces

Ed Burns

ed.burns@sun.com