AJAX in Apache MyFaces

A New Approach

To Web Applications

Gerald Müllan

Matthias Weßendorf

ApacheCon Europe 06

Table of Content

The New Web - Web 2.0

- desktop- vs. web applications
- Web 2.0
  - general definition
  - fully-fledged computing platforms
  - serve web applications to end users
  - personal content, social networks
  - purely web based
  - RSS, Blogs, Wikis

The New Web - Web 2.0

- complex and evolving technology infrastructure
  - server software
  - client applications
  - content syndication
  - messaging protocols
- A Web 2.0 site is built of many techniques
  - one of these is **AJAX**

What is AJAX?

- a new approach to web applications
- a terminology affected by “Jesse James Garrett” from Adaptive Path in february 2005
- short name for “**A**synchronous **J**avaScript **A**nd **X**ML”
- becomes a hype in 2005
- popularity raised with the help of Google
    - Gmail, Google Maps, Google Calendar

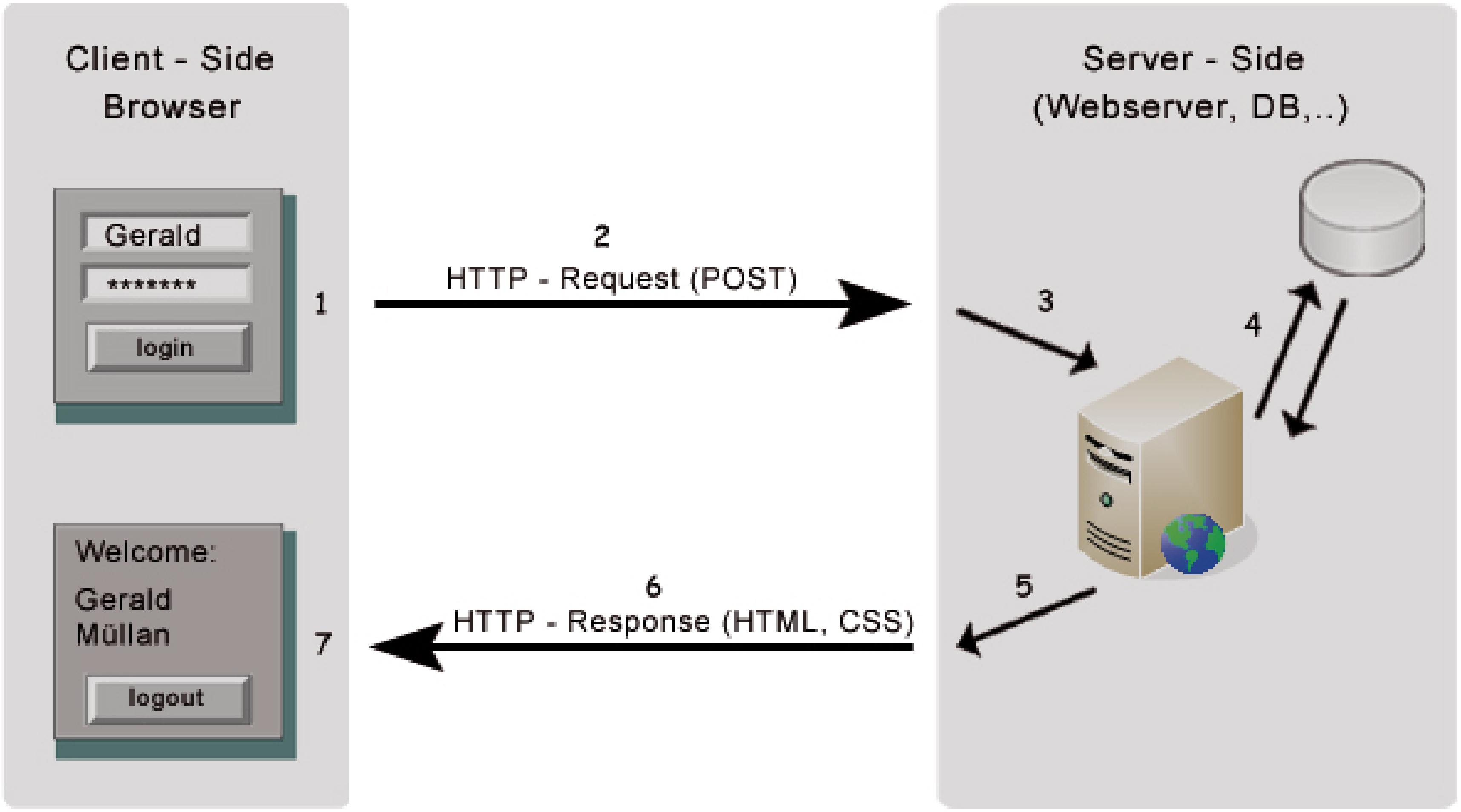
What is AJAX?

- a bundle of common used techniques
  - HTML (or XHTML) and CSS
  - Document Object Model (DOM)
  - XML
  - JavaScript, XMLHttpRequest Object
- hence not a new technology
- former called “server interaction via XMLHttpRequest”

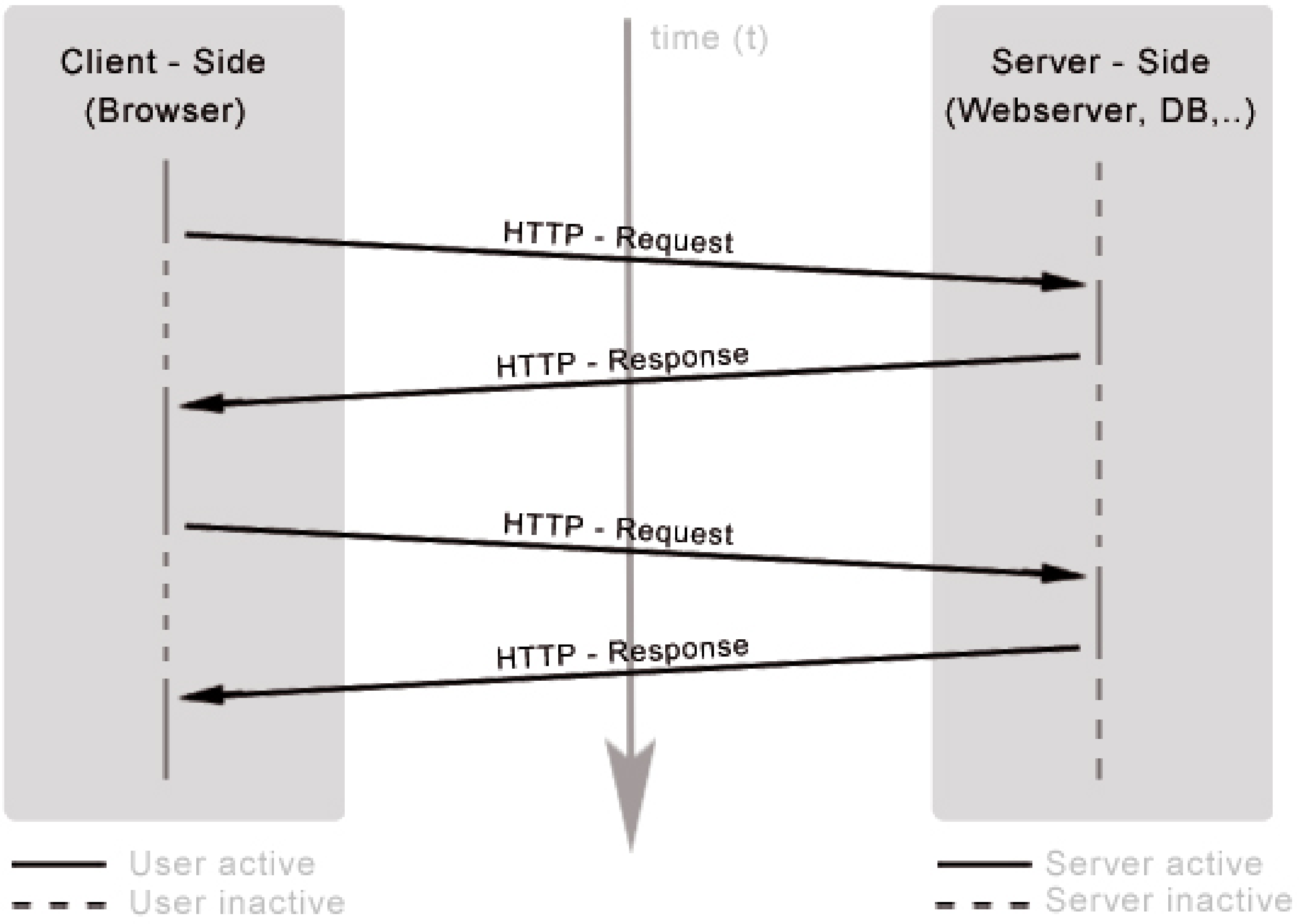
AJAX Interaction

- an AJAX application looks as if it resided on the user's machine
- data is asynchronously fetched
- JavaScript call to AJAX engine
  - HTTP - Request back to server
- browser is updated with gathered information
  - not entirely refreshed

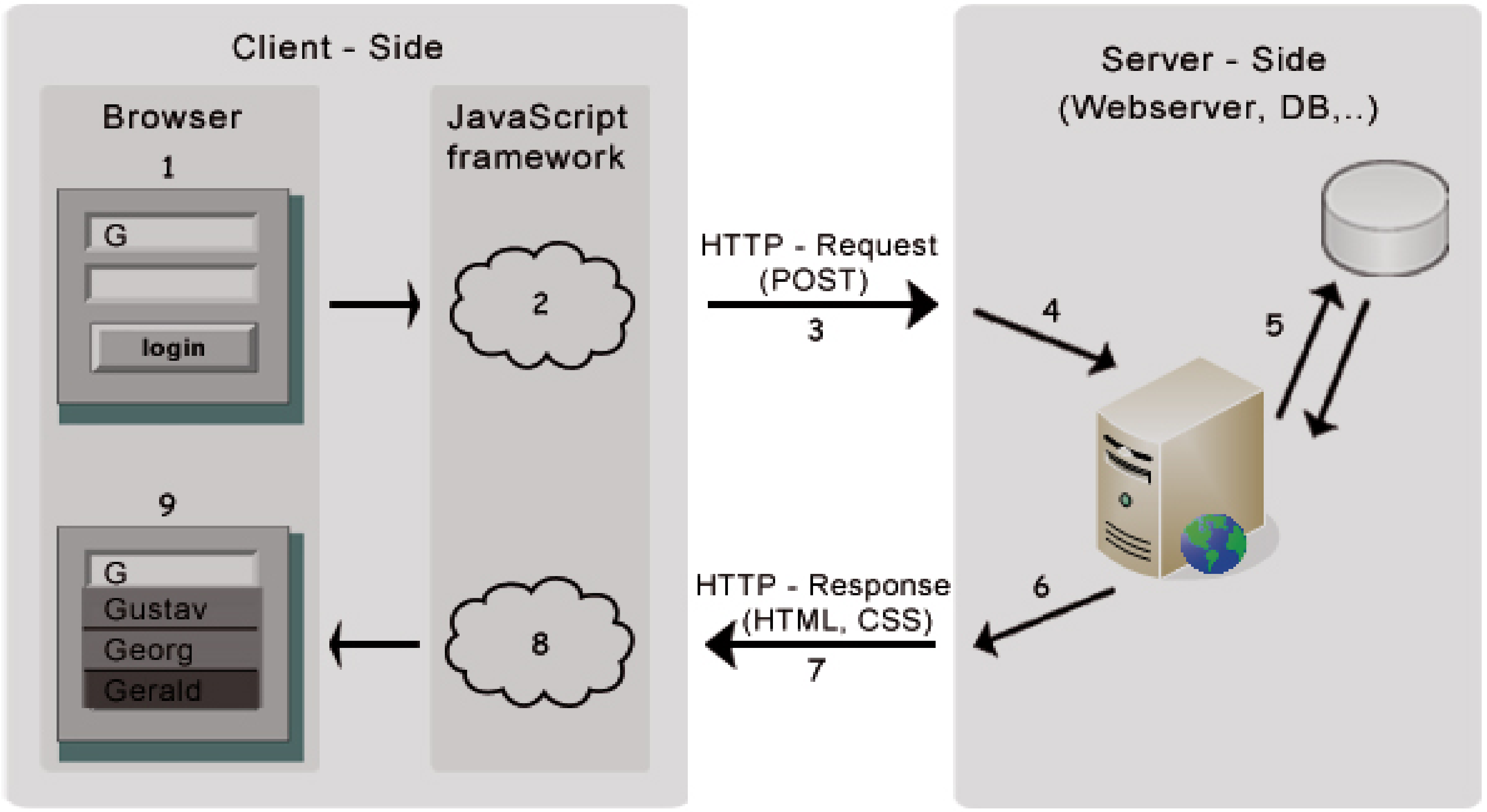
HTTP Request - Response



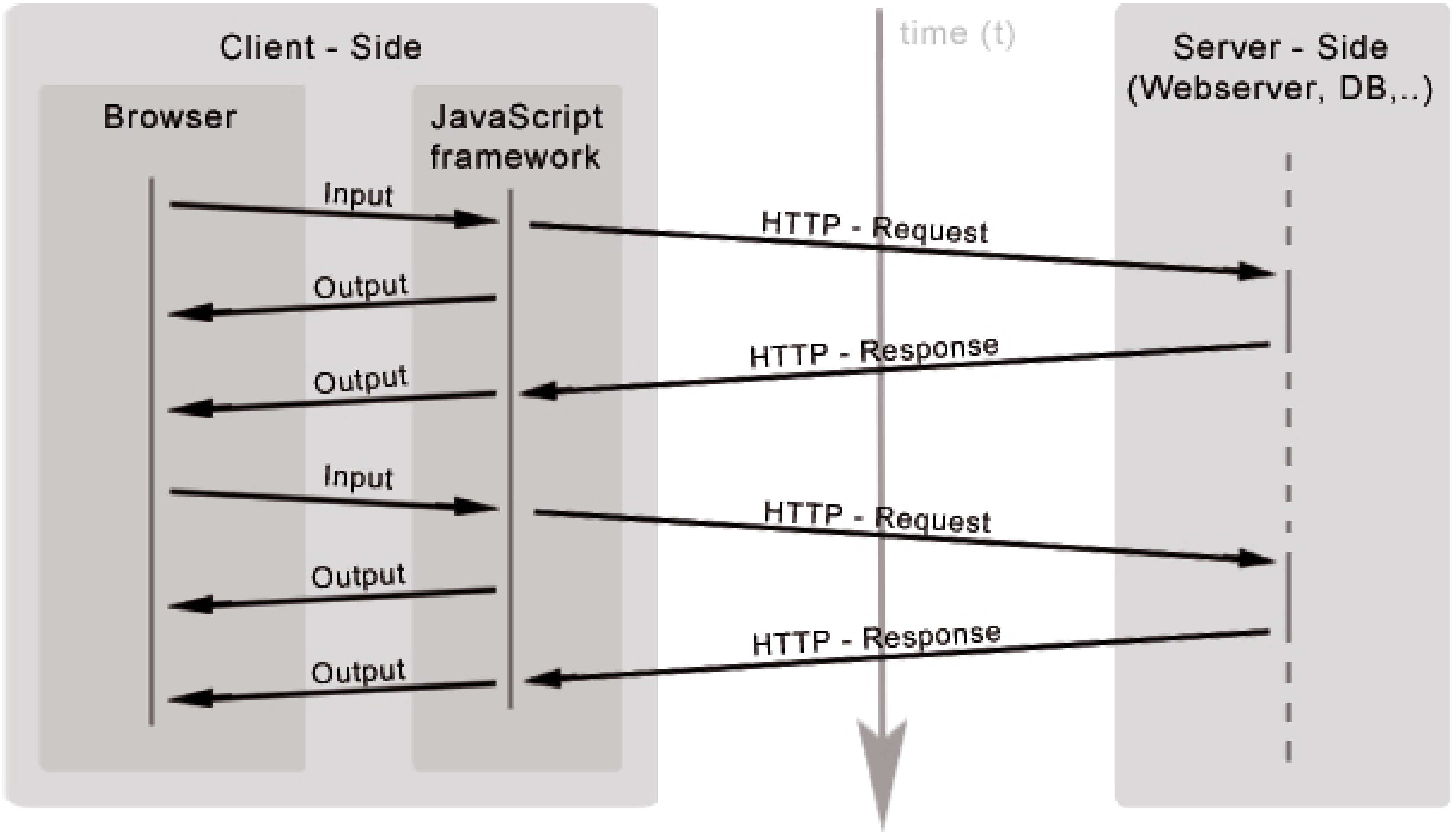
HTTP Process Flow



AJAX Request - Response



AJAX Process Flow



XMLHttp Object

- Http request to server using a JavaScript object
- Microsoft:
  - XMLHttp since IE 5.0 (ActiveX Object)
  - Instantiation:

```
if (window.ActiveXObject) // IE
{
     http_request = new ActiveXObject
                                ("Microsoft.XMLHTTP");
}
```

XMLHttpRequest Object

- Mozilla, Safari..
  - XMLHttpRequest
  - supports same methods and properties

```
if (window.XMLHttpRequest) // Mozilla, Safari
{
      http_request = new XMLHttpRequest();
}
```

- function call after server response: (for both)

```
http_request.onreadystatechange = function(){
    // do useful stuff
};
```

ApacheCon Europe 06

XMLHttpRequest Object

- do the request:

```
http_request.open('GET', URL, true);
```

GET, POST, HEAD… **asynchronous**/synchronous

AJAX!!

```
http_request.send();
```

- response ok, continue processing:

```
if (http_request.readyState == 4)
```

- check the status code:

```
if (http_request.status == 200)
```

JavaScript Libraries

- abstraction layer
  - no handling of XMLHttpRequest object
- libraries provide..
  - ..plain AJAX support
  - ..visual effects
  - ..widgets (dojo toolkit)
  - ..ease of JavaScript handling

JavaScript Libraries

- prototype
  - open-source JavaScript framework
  - former used in MyFaces AJAX components
  - also used by script.aculo.us
- open rico
  - drag and drop management
- dojo toolkit framework
  - widgets
  - namespace structure

Pros of AJAX

- richness and interactivity
  - web application -> desktop application
  - AJAX effects look great
- only parts of the page are changed
  - less redundant data
  - faster update of the page
- no stalling of user’s interaction with the application
- no plug-in needed
  - like flash, shockwave..

Pros of AJAX

- platform independent
  - works across all browsers if JavaScript is enabled
  - works across all operating systems

Cons of AJAX

- stressing the server
  - more requests, data
- web application can feel inactive to the user
  - visualization
- state handling
  - Browsers „back“ button
  - bookmarking
- need of JavaScript support
- differences of browsers -> more testing
  - not a big issue with JavaScript framework

ApacheCon Europe 06

Table of Content

First Strategy (simplest one)

- uses only standard components
  - `<h:form/>,<h:inputText/>,<h:outputText/>`
- for the result
  - `<div>…</div>`
- binding of ressources
  - JavaScript libraries
  - binding of CSS files
- separate Servlet (may be a JSP file as well)
  - reads from `<h:inputText>` through id `form:input`
  - returns `<ul>..<li> ..</li>…</ul>`

First Strategy: The .jsp

```
  <f:view>
     <h:form id="form">
      <h:panelGrid columns="2">
         <h:outputText value="Input"/>
         <h:inputText id="input"
                      value="#{ajaxbacking.input}"/>
         <h:commandButton value="send"/>
      </h:panelGrid>
     </h:form>
     <h:outputTextvalue="Value:#{ajaxbacking.input}"/>
  </f:view>
     <div id="results" class="ajax"></div>

<script>
     new Ajax.Autocompleter('form:input',

     'results','ajax/suggest');
</script>
```

First Strategy: Problems

- change of id -> not generic
- no `autocomplete` attribute of `<h:inputText/>`
- dependence of JavaScript libraries
- mixing of roles
  - page author
  - component writer

Second Strategy

- reuse of standard component `<h:inputText/>`
- -> AJAX component `<ajax:inputText/>`
  - takes care of ressource binding
  - renders the “ajax call” and its markup (`<div>`)
  - important attributes: *`value`*, *`resourceURL, id`*
- component writer:
  - renderer class
  - tag class
- more compact

Second Strategy: The .jsp

```
<f:view>
   <h:form id="form">
     <h:panelGrid columns="2">
       <h:outputText value="Input:"/>
       <ajax:inputText id="input"
                       value="#{ajaxbacking.input}"
                       resourceURL="ajax/suggest"/>
       <h:commandButton value="send"/>
       <h:outputText
              value="Value:#{ajaxbacking.input}"
              rendered="#{!empty ajaxbacking.input}"/>
     </h:panelGrid>
   </h:form>
</f:view>
```

Strategy 2: The Renderer

```
public void encodeBegin(FacesContext context,
                  UIComponent component) throws
  IOException {

this.encodeResources(context, component);
ResponseWriter out = context.getResponseWriter();
String clientId = component.getClientId(context);

out.startElement("input", component);
…
out.writeAttribute("id", clientId, null);
out.writeAttribute("autocomplete", "off", null);
…
"new Ajax.Autocompleter
                  ('“+clientId+"','"+DIV_ID+"',
'"+component.getAttributes().get("resourceURL")+"');
…
```

Second Strategy: Problems

- still relying on servlet (or jsp)
  - parameter form:input must match
- -> solution without external resources like servlet filter or even jsp files

- a plain jsf solution is required!

Third Strategy

- adding a PhaseListener
  - handles the request
  - serves the resources
  - writes `<ul>..<li> ..</li>…</ul>` in the response
- tag class
  - no `resourceURL` attribute
- renderer
  - no servlet mapping
  - handling through JSF engine (actionURL)

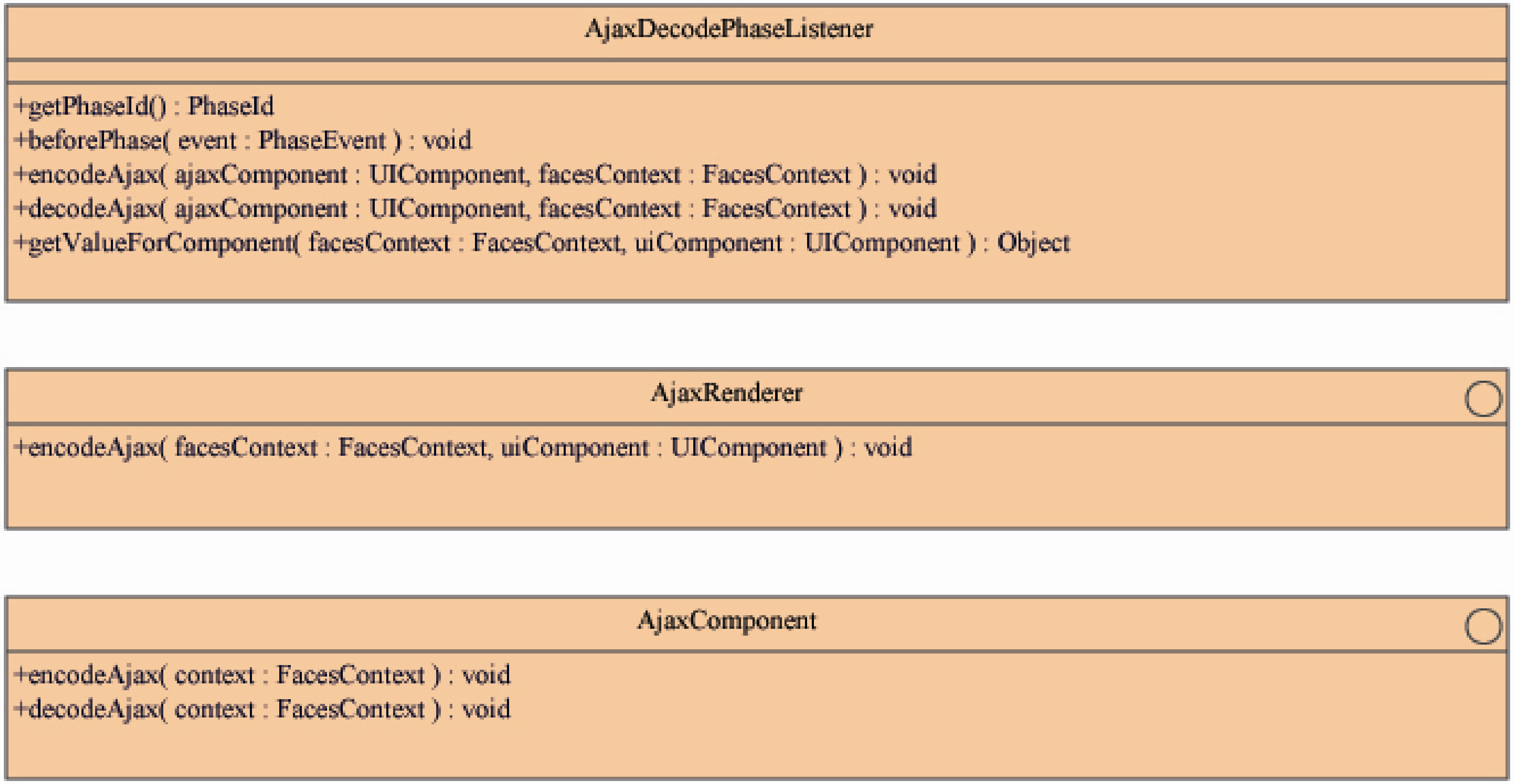
Third Strategy: Impacts

- no external resources
  - Servlet
  - ServletFilter
  - JSP
- seams to be the cleanest strategy...
- but: lookup of list items in PhaseListener
- better solution:
  - component attribute with reference to backing bean
  - -> MyFaces AJAX components

AJAX handling in MyFaces

- AjaxDecodePhaseListener
  - listening for an incoming AJAX request
    - tagged through “affectedAjaxComponent” (URL param)
    - value = id of AJAX component
  - seeks AJAX component in JSF tree
  - further processing delegated to component/renderer
  - quits JSF life cycle

MyFaces AJAX API


AjaxDecodePhaseListener

```
+getPhaseId() : PhaseId
+beforePhase( event : PhaseEvent ) : void
+encodeAjax( ajaxComponent : UIComponent, facesContext : FacesContext ) : void
+decodeAjax( ajaxComponent : UIComponent, facesContext : FacesContext ) : void
+getValueForComponent( facesContext : FacesContext, uiComponent : UIComponent ) : Object
```

AjaxRenderer

```
+encodeAjax( facesContext : FacesContext, uiComponent : UIComponent ) : void
```

AjaxComponent

```
+encodeAjax( context : FacesContext ) : void
+decodeAjax( context : FacesContext ) : void
```

AJAX Processing in MyFaces

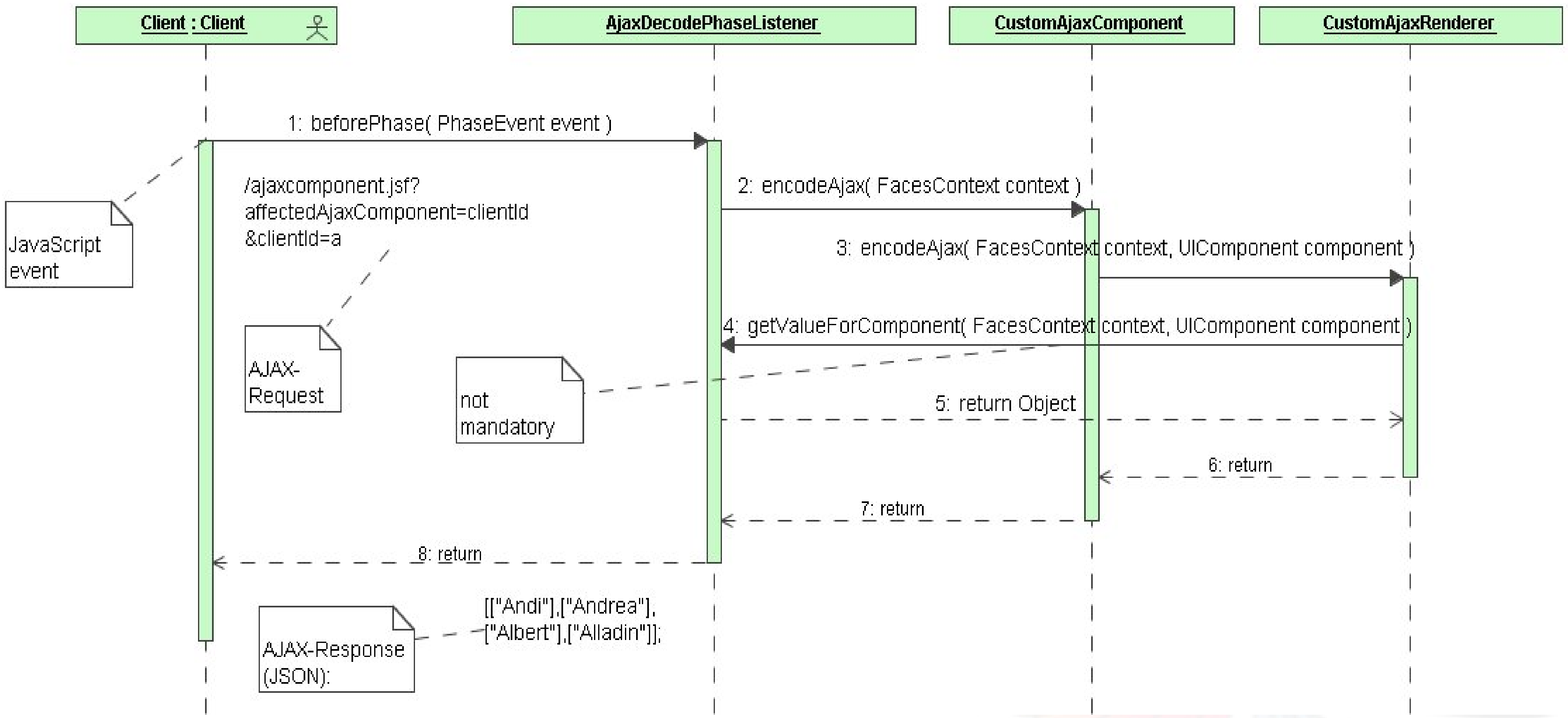


Table of Content

The Dojo Toolkit

- MyFaces has used prototype -> now Dojo!
- advantages
  - huge library
    - abstracts common js handling
    - AJAX api
    - animation
    - event handling
    - widgets!
  - clean separation through namespacing
    - avoids naming conflicts
  - compression of js code

Dojo`s Widgets

- client side js components
- encapsulated js objects
- represent an abstraction layer
  - no need for html/dom handling
  - well defined interface: only one tag!
- hence easy to use
  - eg. <input dojoType="comboBox" dataUrl="..">
- not well-formed XHTML
  - can be also done programatically

Dojo`s Widgets

- fast widget authoring
- client side widgets -> server side JSF comps
- ease of rendering
  - former: <div/>, <input>, <table/>, js-code...
  - now: only one widget tag!
- under the hood
  - one widget = many html tags at runtime

Dojo`s Widgets

- examples
  - ComboBox (InputSuggestAjax)
  - Menues (FishEyeList)
  - Accordion
  - Tree
  - Editor
  - TabbedPane
  - Wizard

InputSuggestAjax

- sandbox component
- autosuggest control with AJAX
  - completely based upon dojo`s comboBox widget
- suggestedItemsMethod
  - method of backing bean
  - delivers preview data
  - realized with MethodBinding

InputSuggestAjax

- call order
  - 1.) Ajax request
  - 2.) AjaxDecodePhaseListener calls InputSuggestAjax
  - 3.) delegates encoding to renderer
  - 4.) renderer calls suggestionMethod in backing bean
  - 5.) computing of result list
  - 6.) result sent back to client; dojo control shows suggestion drop down

InputSuggestAjax: .jsp

```
<h:form>
  <h:panelGrid columns="3“>
        <x:outputLabel value="#{label.title_product}"/>
        <s:inputSuggestAjax
            suggestedItemsMethod=“
                  #{product.getSuggestedProducts}"

  value=“#{product.productNameToSearch}"/>
        <x:commandButton value="#{label.productButton}"

      action="#{product.searchForProducts}"/>
  </h:panelGrid>
</h:form>
```

InputSuggestAjax : suggestedItemsMethod

```
public List getSuggestedProducts(String prefix)
{
   List<String> suggestedNames = new ArrayList<String>();

   Session session = getSession();
   Criteria crit = session.createCriteria(Product.class)
                                  .add(Restrictions

   .like("name","%"+prefix+"%"));

   List<Product> tempProds = crit.list();

    for(Product prod : tempProds)
                suggestedNames.add(prod.getName());

    return suggestedNames;
}
```

sample

AutoUpdateDataTable

- sandbox component
- frequency controlled updated DataTable
- uses prototype.js
  - Ajax.PeriodicalUpdater(...)

AutoUpdateDataTable: .jsp

```
<h:form>
   <s:autoUpdateDataTable
               var="bids“
               value="#{dataTableData.bids}"
               preserveDataModel="true"
               frequency="3">
         <h:column>
               <f:facet name="header">
               <h:outputText escape="false"
   value=“Bid"/>
        </f:facet>
        <h:outputText value="#{bids.bidData}" />
         </h:column>
   </s:autoUpdateDataTable>
</h:form>
```

sample

ApacheCon Europe 06

Links and books

- MyFaces AJAX examples
  - http://www.irian.at/open_source.jsf (sandbox components)
- AJAX web resources
  - http://www.script.aculo.us
  - http://www.adaptivepath.com
  - http://www.dojotoolkit.org
  - http://www.ajaxpatterns.org/Ajax_Frameworks
  - http://www.ajaxdeveloper.org

Questions ?

- Answers!