BootsFaces and AngularFaces

Deep Dive

Riccardo Massera (BootsFaces team lead)
Stephan Rauh (Leiter CC „modern Clients“)

What's it all about

1. History
2. BootsFaces in a Nutshell
3. BootsFaces Deep Dive
4. AngularFaces in a Nutshell
5. AngularFaces Deep Dive

Breaking News

BootsFaces 1.0 has been released!

Image source: 留米市民(Kurume-Shimin) https://commons.wikimedia.org/wiki/File:ColorfulFireworks.png,

History

- Company frameworks:
  Medusa, Perseus and „The Framework“
- PrimeFaces and Liferay
- AngularPrime and AngularDartPrime
- AngularFaces
- BootsFaces

History: Medusa (1997-2000)

- Stephan's first company framework
- Full-stack framework: persistence and UI
  - Two-layer desktop application architecture
- Based on SQLWindows / Centura
- Challenge:
  - Buggy programming language, buggy libraries (in 1997)
- Solution:
  - Circumvent the bugs by providing your own framework
- Lessons learnt:
  - Writing a framework pays!

History: Perseus (2000 – 2005)

- Successor of Medusa
- Full-stack 3-layer web framework
- Frequent refactoring
- Technologies examined:
  - C++ with Roguewave vs. Java
  - Rational Rose vs. TogetherJ
  - Distributed computing with CORBA
  - Applets vs. HTML clients
- Later abandoned in favor of standard open source frameworks

History: „The Framework“ (2002 – today)

- Full-stack three-layer web framework
- Optimized for Stephan‘s former company
- 10+ applications featuring a million lines of code
- Lessons learnt:
  - Implementing our own framework shielded developers from the fast-paced Java community
  - Technologies avoided: Struts, EJB 2.x, …
  - It‘s possible to evolve a framework over 10+ years without breaking compatibility!
  - Maintaining a company framework costs roughly 5-10% of the total budget
  - Writing your own framework still pays if you get the opportunity to optimize it for your company‘s needs
  - Frameworks don‘t need a name to be successful ☺

History: Liferay and PrimeFaces (2010-2015)

- Goal: migrate „the framework“ to LifeRay
- Turned out to be impossible
  - Portlets enforced a different project setup
- Quick adoption of PrimeFaces, CDI and Hibernate
- Later abandoned Liferay in favor of Tomcat
  - Portlets proved to be too fine-grained modules for most real-world applications
- Lessons learnt:
  - Always choose frameworks matching your requirements and your company culture
  - Standard Java frameworks have grown up (roughly around 2005-2010)
  - Even today, developer productivity is still worse than using the former, highly optimized company framework!

History: AngularPrimeDart (2012-2013)

- Goal: migrate Rudy de Busscher's AngularPrime to the Dart language
- Result:
  - Nice and useful set of UI components
  - Which were largely ignored by the broad market
- Lessons learnt:
  - The time is ripe to move logic to the client
  - Doing so reduces both network traffic and server load significantly
  - Interesting programming languages don't always convince the market

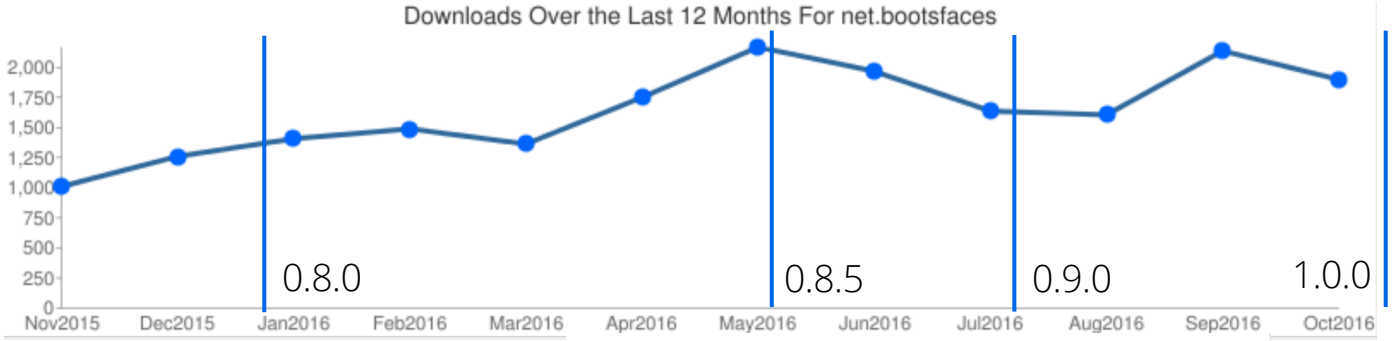
(End of all) History: AngularFaces and BootsFaces

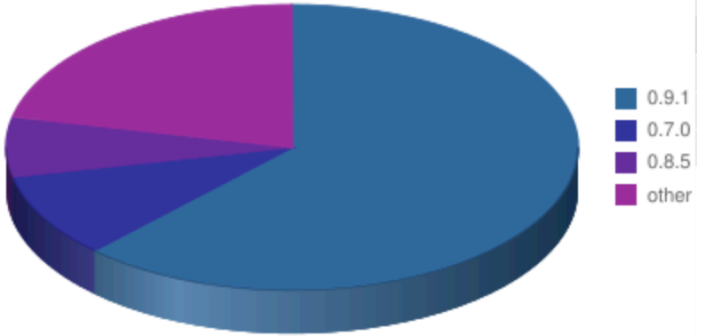
- AngularFaces:
  - Replace AJAX by AngularJS
  - Add rapid prototyping
  - Simplify i18n
- BootsFaces:
  - Add Bootstrap to JSF
  - Make JSF applications beautiful...
  - ... without effort!
  - Simplified approach to AJAX
  - Powerful search expressions
- Common goals:
  - Make JSF simple
  - Boost developer productivity
  - Improve the user experience

BootsFaces in a Nutshell

BootsFaces in a Nutshell: Numbers

- JSF component framework
  - 4 developers
  - 20 releases
  - 62 pull requests
  - 70+ components
  - 400 solved issues
  - 1200 commits
  - 2000 downloads per month
  - 65000 lines of HTML in our showcase
  - 68000 lines of Java code

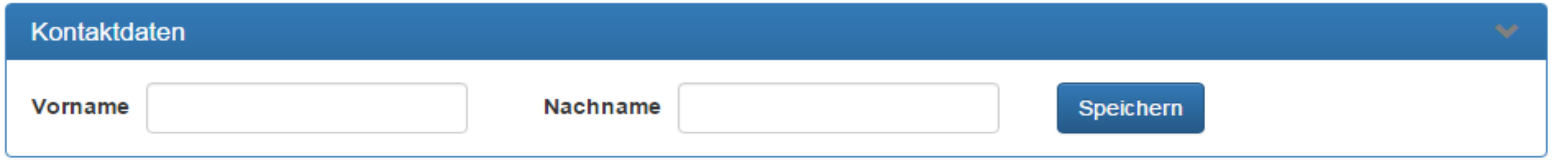




BootsFaces in a Nutshell: Goals

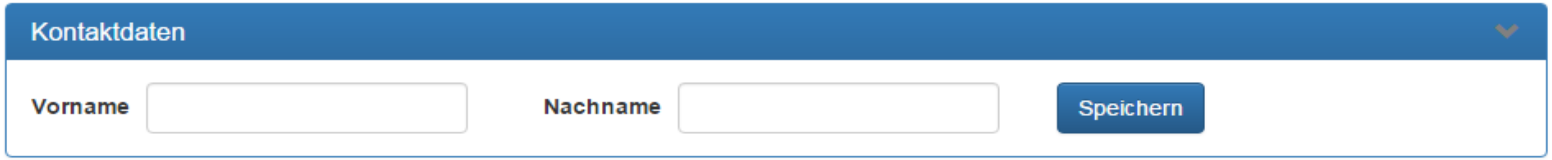
- Add Bootstrap to JSF
  - Used to be a hard task in JSF until 2.1!
  - Pass-through attributes made the integration of Bootstrap easier
- Make JSF compact and simple
  - Code that‘s not there is code you don‘t have to maintain!
- Make AJAX simple and more intuitive
  - Evolving the standard JSF API
- Create professional looking Enterprise Web Applications
- Currently Bootstrap 3.x, Bootstrap 4 still in Alpha (ETA 2017)
- BootsFaces 2.0 will support Bootstrap 4

BootsFaces in a Nutshell: Example




```
<b:panel title="Kontaktdaten" look="primary">
  <b:row>
    <b:column small-screen="one-third">
      <b:inputText label="Vorname" render-label="true" inline="true" />
    </b:column>
    <b:column small-screen="one-third">
      <b:inputText label="Nachname" render-label="true" inline="true" />
    </b:column>
    <b:column small-screen="one-third">
      <b:commandButton value="Speichern" look="primary" />
    </b:column>
  </b:row>
</b:panel>
```

BootsFaces in a Nutshell: Generated Code




```
<b:panel title="Kontaktdaten" look="primary">
  <b:row>
    <b:column small-screen="one-third">
      <b:inputText label="Vorname" render-label="true" inline="true" />
    </b:column>
    <b:column small-screen="one-third">
      <b:inputText label="Nachname" render-label="true" inline="true" />
    </b:column>
    <b:column small-screen="one-third">
      <b:commandButton value="Speichern" look="primary" />
    </b:column>
  </b:row>
</b:panel>
```

```
<div class="panel-group">
  <div class="panel panel-primary">
    <div class="panel-heading">
      <h4 class="panel-title">
        <a data-toggle="collapse" data-target="#content">Kontaktdaten</a>
      </h4>
    </div>
    <div id="content" class="panel-body  panel-collapse collapse in">
      <div class="row">
        <div class="col-sm-4">
          <div class="form-inline">
            <label for="firstname">Vorname</label>
            <input id="firstname" class="form-control" />
          </div>
        </div>
        <div class="col-sm-4">
          <div class="form-inline">
            <label for="lastname">Nachname</label>
            <input id="lastname" class="form-control" />
          </div>
        </div>
        <div class="col-sm-4">
          <div class="form-inline">
            <button type="submit" class="btn btn-primary">Speichern</button>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

BootsFaces in a Nutshell: Compatibility

- PrimeFaces (within certain limits)
  - Different approaches to CSS cause incompatibilities
- OmniFaces
- HighFaces
- GISFaces (no example in our showcase yet)
- ButterFaces (work in progress)
- JSF 2.0 – JSF 2.3
- Java 7 + 8
- Java 6 (within certain limits)

BootsFaces Deep Dive

- Separate renderers?
- Large-scale refactoring
- Xtext
- AddResourcesHandler
- Gradle ~~vs.~~ and Maven
- PatternFly
- Relaxed, HTML-like coding style
- Anything else?

BootsFaces Deep Dive: Separating Renderers

- Original approach: Components render themselves
  - Less code
  - Simple programming model

- New approach: separate renderers and components
  - More flexibility
  - Developers can modify and improve our implementations easily
  - Code generators reduce the amount of extra work

BootsFaces Deep Dive: Large-Scale Refactoring

- We ran several large-scale refactorings and restructurings:
  - Separating renderers from components as an afterthought
  - Moving every component into a package of their own
  - Adding AJAX to every component
  - Adding layout attributes to every component
  - Adding and maintaining the copyright header in every class
  - Getting rid of annotations after implementing 40+ components
- Key points to success:
  - Tool support
  - Generate as much code as possible
  - Writing utility programs manipulating the source code

BootsFaces Deep Dive: XText

```
widget accordion
       implemented_by net.bootsfaces.component.accordion.Accordion
       extends UIComponentBase
       has_children
       has_tooltip
       is_responsive
{
    id                      inherited
    rendered                Boolean inherited
    expanded-panels
    +tooltip
    +responsive
    +style
}
```

- DSL + code generator
- Simplified component definition
- Generate everything:
  - Component class
  - Renderer class (stub)
  - Taglib
  - Documentation (stub)
- Disadvantages:
  - Only available for Eclipse
  - Pull-request contributors ignore the plugin

PrimeFaces Approach to Code Generation

- Maven plugin for code generation
  - Source: taglib files + partial Java classes adding non-standard code
  - Component classes are (mostly) generated from the partial taglib files
- Advantages:
  - Works with every IDE
  - Code generator is every bit as powerful as its BootsFaces counterpart
  - Writing a Maven plugin is surprisingly easy
- Disadvantages:
  - Source code project is incomplete until the code generator runs
  - It takes some time and effort to build PrimeFaces from source

Alternative Approaches to Code Generation

Ideas suggested by Thomas Andraschko:

- Sourcecode: abstract classes + Java annotations
  - → you've got a syntactically correct project
  - → The sourcecode is Java, not XML
- Code generator options:
  - Custom Maven plugin
  - Java Agents
- Java Agents:
  - Standardized approach to code generation that comes out-of-the box with Java
  - Requires an additional JVM parameter
  - Modifies the byte code at load time (within certain limits)

BootsFaces Deep Dive: AddResourcesHandler

- Challenge:
  - Sometimes, developers need to add their own version of jQuery etc.
  - JSF always adds a fixed set of resource files (if you‘re using annotations)
- Solution:
  - Implement an AddResourcesHandler to avoid adding resources provided by the user
- Challenge:
  - Need to modify the folder name of resources to support theming
- Solution:
  - AddResourcesHandler modifies resource folder names on the fly
- This is the most crucial class of BootsFaces!

BootsFaces Deep Dive: Getting Rid of Annotations

- JSF simplifies component development by providing annotations
- Problem: annotations are immutable
  - Resource files are always added
  - … even if there‘s already a different version provided by the programmer
  - E.g. jQuery, jQueryUI
- Resources are added in the wrong order

→ BootsFaces moved every @ResourceDependency to Java code

BootsFaces Deep Dive: Getting Rid of Annotations

Traditional declarative approach:

```
@ResourceDependencies({
    @ResourceDependency(library="primefaces", name="components.css"),
    @ResourceDependency(library="primefaces", name="jquery/jquery.js"),
    @ResourceDependency(library="primefaces", name="jquery/jquery-plugins.js"),
    @ResourceDependency(library="primefaces", name="core.js"),
    @ResourceDependency(library="primefaces", name="components.js")
})
public class Growl extends UIMessages implements org.primefaces.component.api.Widget
```

BootsFaces Deep Dive: Getting Rid of Annotations

BootsFaces programmatic approach:

```
@FacesComponent("net.bootsfaces.component.growl.Growl")
public class Growl extends UIMessages {

    public Growl() {
        super();
        setRendererType("net.bootsfaces.component.GrowlRenderer");
        AddResourcesListener.addResourceToHeadButAfterJQuery(
                                C.BSF_LIBRARY,
                                "js/bootstrap-notify.min.js");
        AddResourcesListener.addThemedCSSResource("core.css");
        AddResourcesListener.addExtCSSResource("animate.css");
    }
```

BootsFaces Deep Dive: Two Build Systems

- Gradle: powerful build system generating 13 themes in a minute
  - Installation usually takes some time
- Maven: simple build system
  - Easy start: usually up and running in a minute
  - Most pull-request contributors prefer Maven
  - Consumes the JS and CSS files generated by Gradle
  - Simple upload to Maven Central
- Challenge:
  - Keep the Maven build in sync with the Gradle build!

BootsFaces Deep Dive: PatternFly

- Initial support is introduced with BootsFaces 1.0
- PatternFly is a HTML5 UI framework for Enterprise Web Applications
- Based on Bootstrap 3.x it's an open source Project developed by Red Hat
- BootsFaces and PatternFly share almost the same set of widgets
- The first step has been to add it to BootsFaces as a theme
- Adds a collection of UI design patterns that solve common design problems considering the needs of complex enterprise IT software
- The goal is to give BootsFaces users the chance to develop professional looking Enterprise Web Applications

BootsFaces Deep Dive: BootsFacesDecorator

- Relaxed, HTML-like coding style:

```
<well>
  <h:form id="loginForm">
    <row>
        <inputText id="usernameDiv" small-screen="half"
                   value=""
                   label="Username" render-label="true"/>
        <message for="@previous" small-screen="half"/>
    </row>
    <row>
        <inputSecret id="passwordDiv" small-screen="half"
                     value=""
                     label="Password"  render-label="true"/>
        <message for="passwordDiv" small-screen="half"/>
     </row>
    <row>
       <select small-screen="half" label="Login as" render-label="true">
            <option itemLabel="(Please select)" itemValue="" />
            <option value="1" label="admin"/>
            <option value="2" label="customer" />
            <option value="3" label="user" />
       </select>
    </row>
    <row>
        <column small-screen="full-width">
        <button update="@form:loginForm" value="Login" onclick="ajax:loginBean.login2()" look="primary"/>
        </column>
    </row>

  </h:form>
</well>
```

- First port of an AngularFaces idea to BootsFaces

```
<well>
  <h:form id="loginForm">
    <row>
        <inputText id="usernameDiv" small-screen="half"
                   value=""
                   label="Username" render-label="true"/>
        <message for="@previous" small-screen="half"/>
    </row>
    <row>
        <inputSecret id="passwordDiv" small-screen="half"
                     value=""
                     label="Password"  render-label="true"/>
        <message for="passwordDiv" small-screen="half"/>
     </row>
    <row>
       <select small-screen="half" label="Login as" render-label="true">
            <option itemLabel="(Please select)" itemValue="" />
            <option value="1" label="admin"/>
            <option value="2" label="customer" />
            <option value="3" label="user" />
       </select>
    </row>
    <row>
        <column small-screen="full-width">
        <button update="@form:loginForm" value="Login" onclick="ajax:loginBean.login2()" look="primary"/>
        </column>
    </row>

  </h:form>
</well>
```

AngularFaces in a Nutshell

- It's a Plugin!
- AngularJS
- Rapid Prototyping
- Reception and Market Share
- What about the Future?

AngularFaces in a Nutshell: It‘s a Plugin!

Unorthodox JSF framework:

- No components
- Plugin hooking itself in JSF 2.x and PrimeFaces
- Enhance existing JSF components
- Theoretically compatible with every JSF component framework

AngularFaces in a Nutshell: Integrating AngularJS

- Central idea:
  - Add AngularJS to existing JSF applications
  - Replace AJAX by AngularJS
  - Bridge technology, preparing the road to AngularJS
- Challenge:
  - Two complex and incompatible life cycles
- Solution:
  - JSF first
  - Integrate AngularJS into the JSF life cycle
  - Every JSF page is a SPA
  - AngularFaces apps consist of several, independent SPAs

AngularFaces in a Nutshell: Dealing with tags

- Most AngularJS tutorials rely heavily on tags
- is no first class citizen of JSF
- Solution:
  - Convert and into first-class citizens
  - … by creating a special component: <pui-div>
  - … and implementing a TagDecorator
- → AngularFaces requires JSF 2.2 or higher

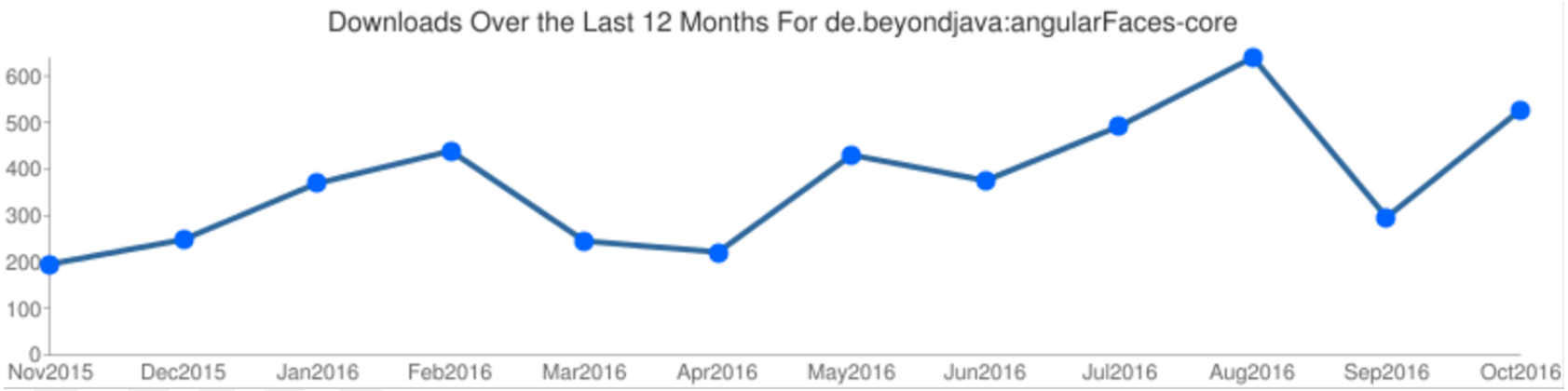
AngularFaces in a Nutshell: Rapid Prototying

- Simplyfy common tasks:
  - Add labels and FacesMessages automatically
  - Add i18n support automatically

- Optional relaxed, HTML-like coding style
  - Omit most h: prefixes of standard JSF
  - If BootsFaces is found, also omit most b: prefixes

AngularFaces in a Nutshell: Reception and Market Share

- Warm reception among JSF framework developers
  - Kito Mann, Reza Rahman, Çağatay Çivici
- Scepticism among JSF users
- Up to 600 downloads per month

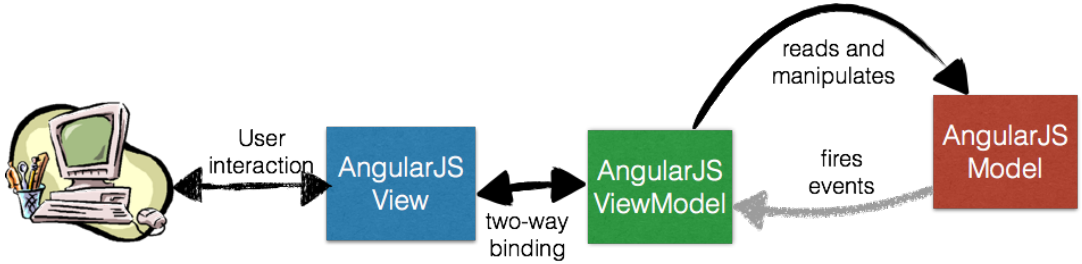


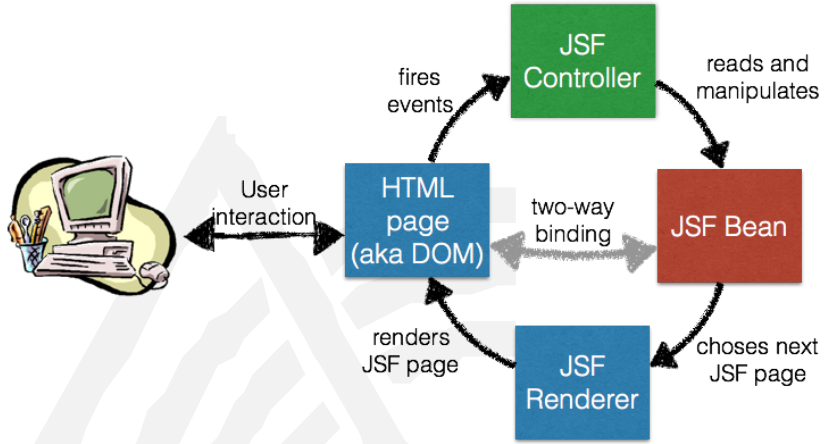
AngularFaces in a Nutshell: What about the Future?

- Current status:
  - Finished, only need some polishing once in a while
  - One active contributor: Oscar Mendel
- Angular2:
  - No plans for Angular2 support for technical reasons
  - Angular2 applications can‘t be remotely controlled by JSF
- Alternative:
  - Angular2 controls a JSF application
  - You can do that today, even without AngularFaces!

AngularFaces Deep Dive

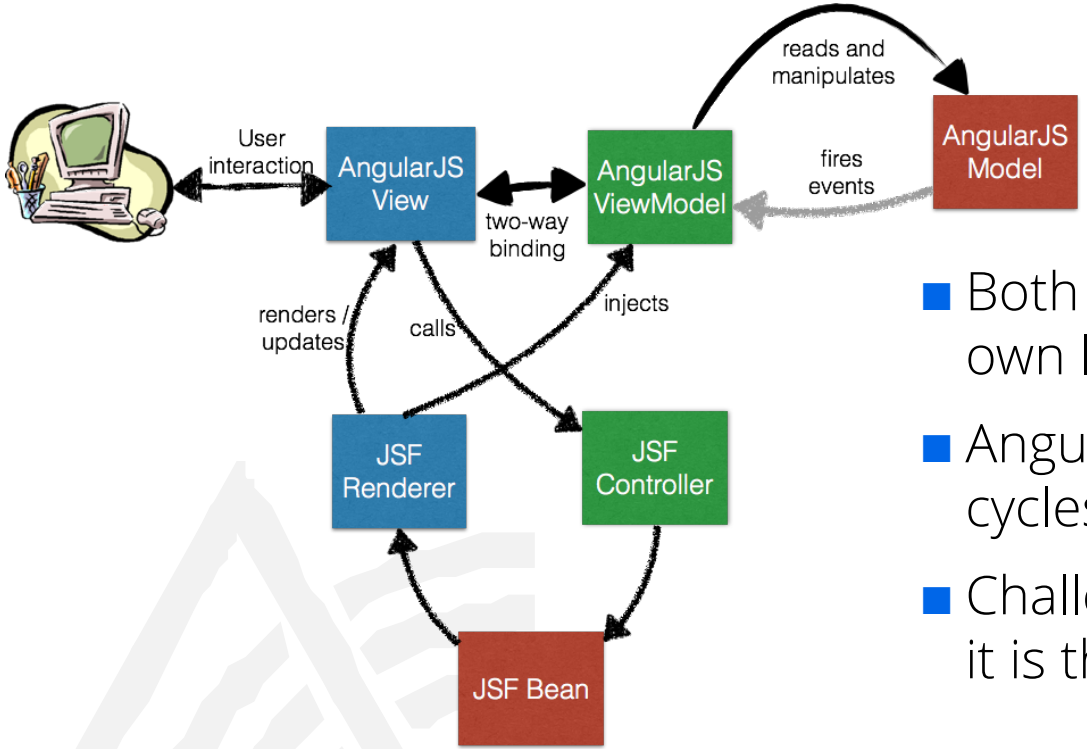
AngularFaces Deep Dive: Incompatible Life Cycles





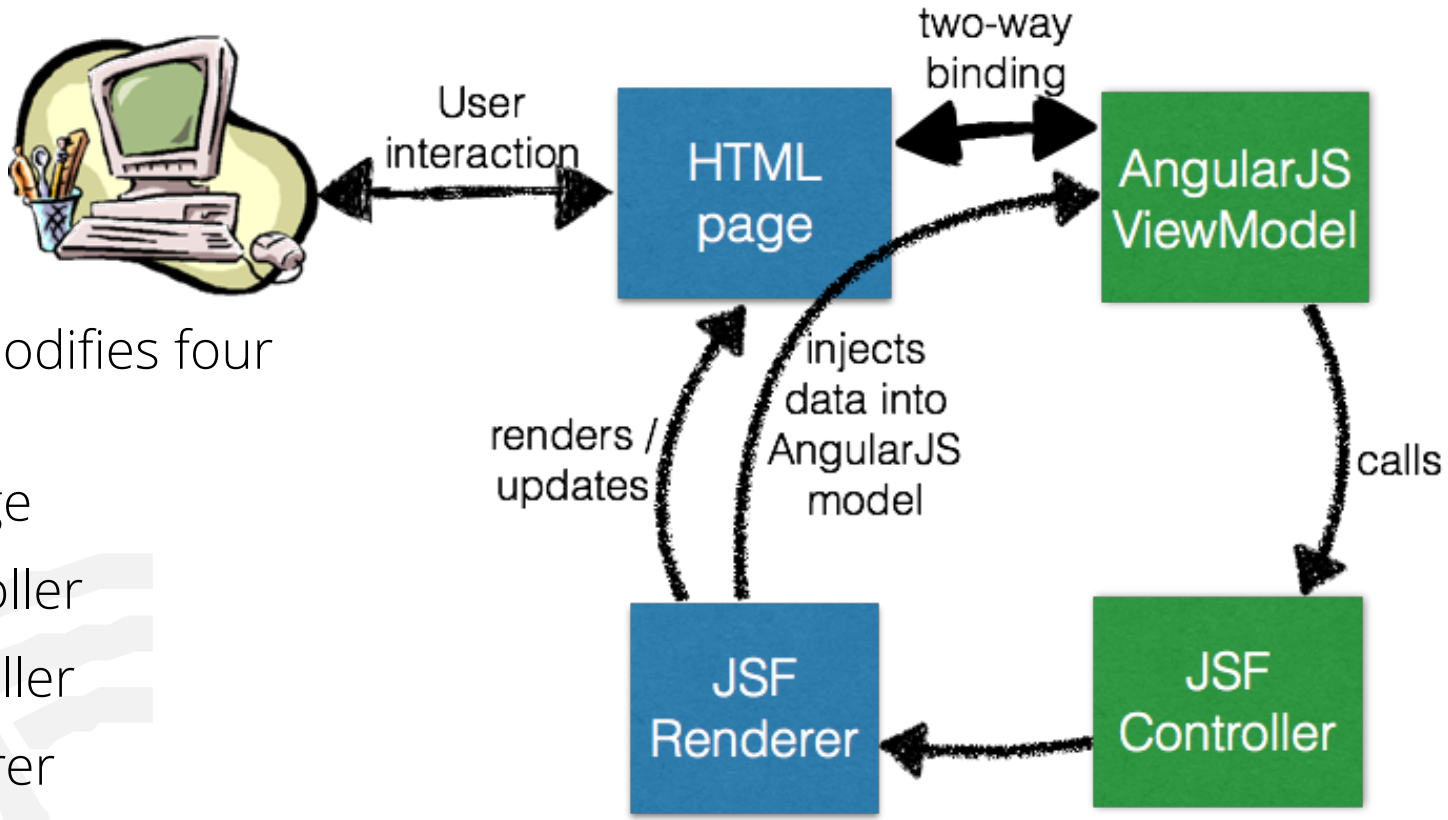
- Both JSF and AngularJS have their own life cycle
- AngularFaces merges these two life cycles
- Challenge: each framework believes it is the owner of the DOM

AngularFaces Deep Dive: Incompatible Life Cycles



- Both JSF and AngularJS have their own life cycle
- AngularFaces merges these two life cycles
- Challenge: each framework believes it is the owner of the DOM

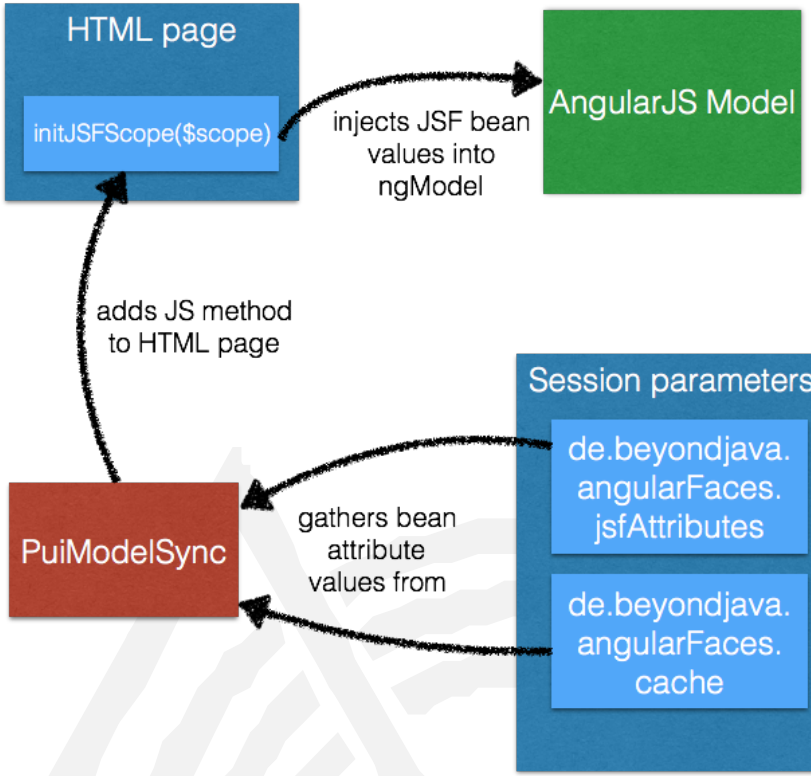
AngularFaces Deep Dive: Simplified View

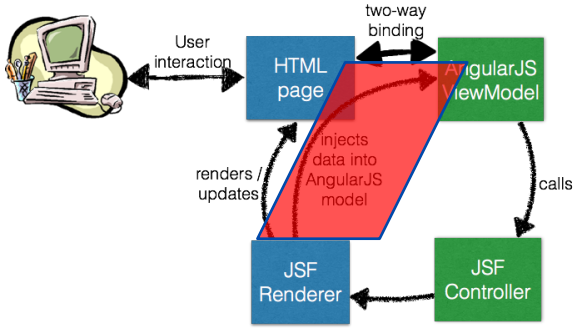


AngularFaces modifies four components:

- the HTML page
- the NG controller
- the JSF controller
- the JSF renderer

Injecting the JSF model into the ngModel

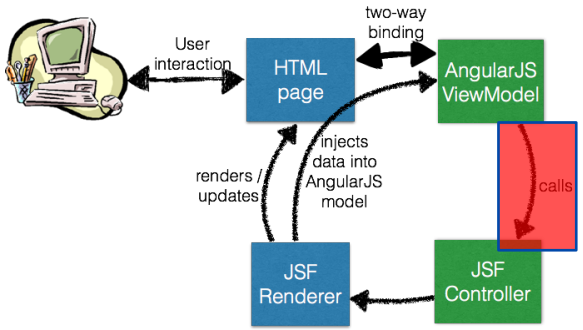




- Most difficult challenge of AngularFaces
- AngularFaces goes into some lengths to generate as much code as possible for you (so you have to write less)
- Still requires some help by the developer, who has to call initJSFScope()

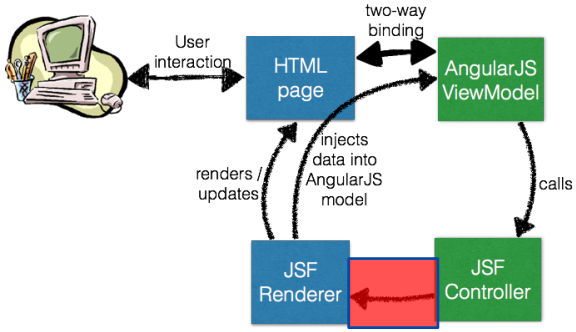
AngularFaces Deep Dive: Updating the JSF Bean

- Basic idea: AngularFaces adds a virtual tag, PuiSync, to each AngularJS controller
- PuiSync generates a hidden input field updated by AngularJS
- The values transmitted by PuiSync are evaluated by AngularViewContextWrapper as part of the regular decoding process



Gathering Data for initJSFScope($scope)

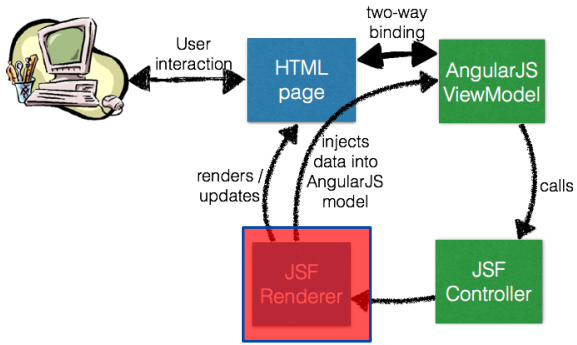
- The tag decorators play a crucial role
- The AngularTagDecorator detects mustache expressions, convert them to JSF EL expressions and stores them in the session
- Later, PuiModelSync.encode() generates initJSFScope() using the attribute list stored in the session



FindNGControllerCallback

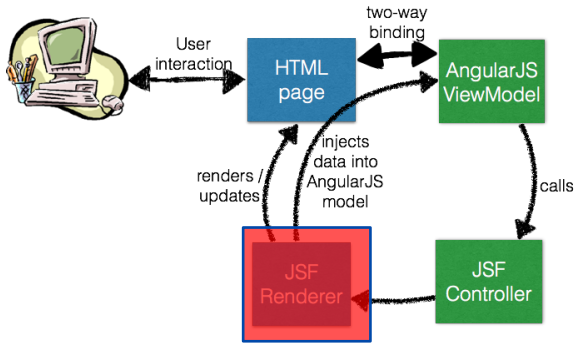
Two responsibilities:

- Adds the PuiModelSync tag to each ngController
- „Automagically“ adds labels and messages to input fields



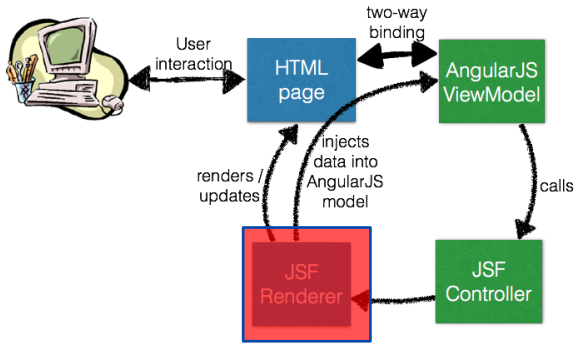
Simplified Internationalization

- Mostly implemented in the TranslationCallback class
- The most common text attributes are translated to foreign languages by calling the i18n.translate() method as part of the PreRenderViewEvent
- ac:include is a language-aware ui:include (adding the language as a suffix to the file name)



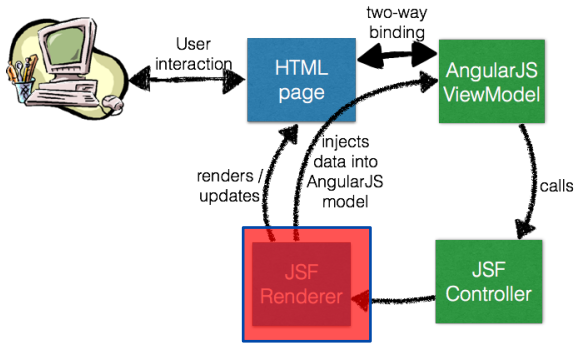
Adding Labels and Messages „automagically“

- Already described at the FindNGControllerCallback section
- Can be deactivated



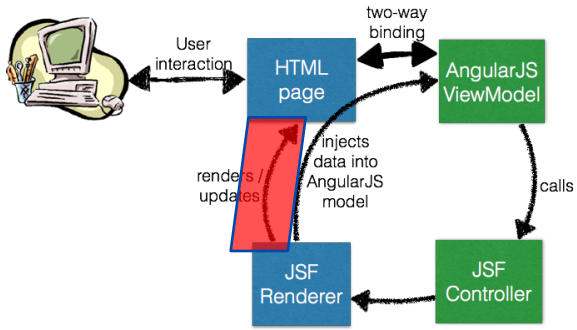
Bringing the Bean Validation API to the Client

- Implemented in the AddTypeInformationCallback class
- The type information is added to the JSF tags as part of the PreRenderViewEvent



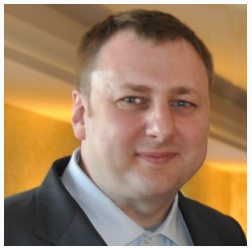
JavaScript Library Dependency Management

- The JavaScript dependencies of AngularFaces are added on-the-fly as part of the PreRenderViewEvent
- Implemented in PuiAngularTransformer
- Predecessor of the BootsFaces AddResourcesHandler



Any questions?

- It‘s your turn!



Riccardo Massera

BootsFaces teamlead

ric.massera@gmail.com

WWW.OPITZ-CONSULTING.COM

@OC_WIRE

OPITZCONSULTING

opitzconsulting

opitz-consulting-bcb8-1009116

Let‘s make the web a place to be!

Stephan Rauh

Leiter CC „moderne Clients und agile Architekturen“

0172-205 59 66

Stephan.Rauh@opitz-consulting.com

@beyondjava

http://www.beyondjava.net

WWW.OPITZ-CONSULTING.COM

@OC_WIRE

OPITZCONSULTING

opitzconsulting

opitz-consulting-bcb8-1009116