irian

THE SOFTWARE EXPERTS

JSF 2 - Kompositkomponenten

Kompositkomponenten im Einsatz

Gerald Müllan | IRIAN

irian | THE JAVA EXPERTS

Agenda

- Motivation
- JSF 2-Ressourcenverwaltung
- Einführung Kompositkomponenten
- Erweiterte Konzepte
- Beispiele

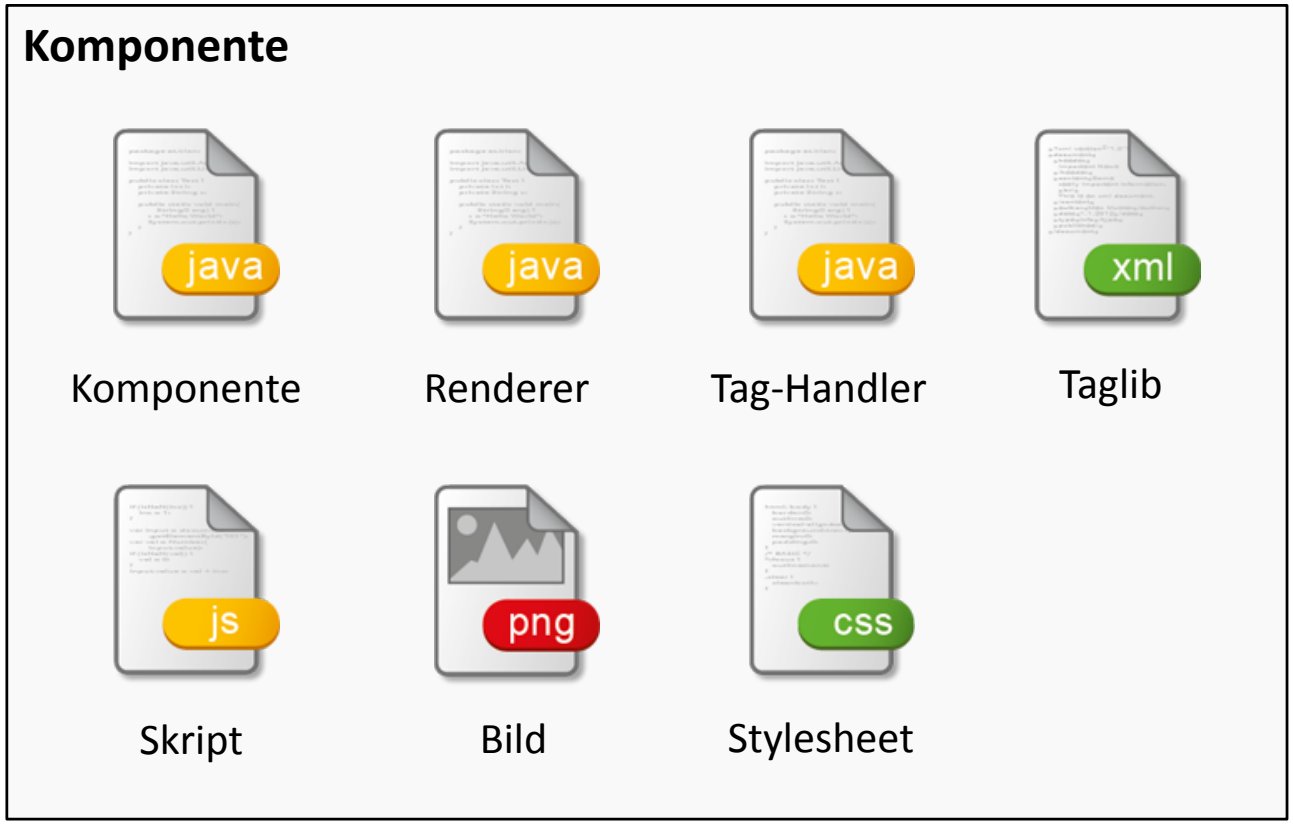
MOTIVATION

Motivation

- Komponente zentraler Bestandteil von JSF

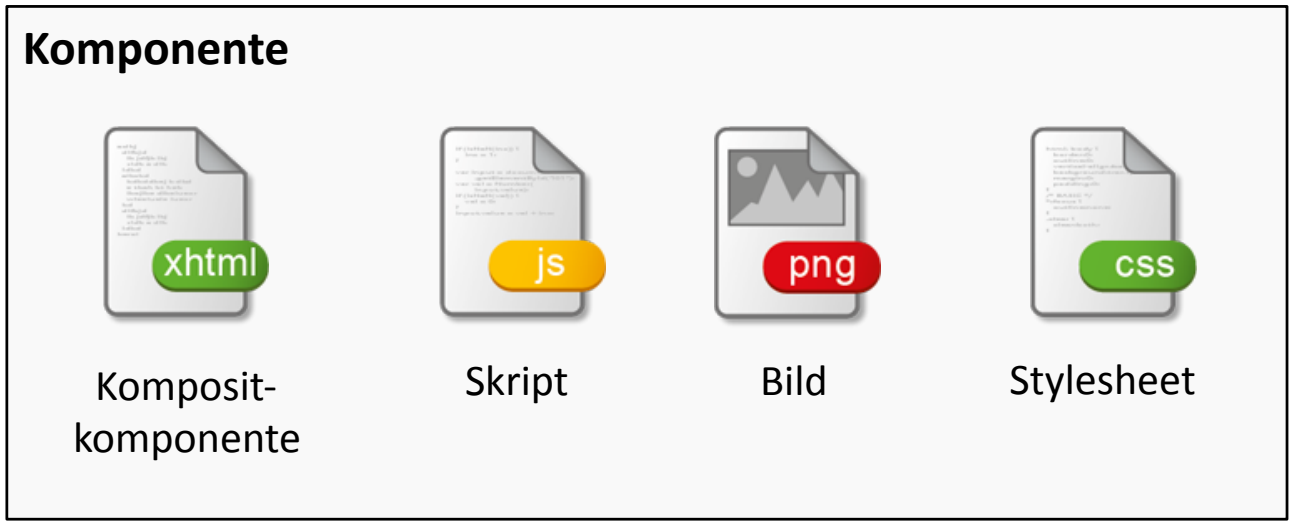
- Eigene Komponente nötig/sinnvoll
- Klassische Komponenten
  - Mächtig aber komplex (JSF-Meisterprüfung)
- Neuer Ansatz: Kompositkomponenten
  - Komponenten ohne Java und XML
  - Wiederverwendung von Seitenfragmenten

Klassische Komponenten



Kompositkomponenten

- Komponenten ohne Java und XML



JSF 2-RESSOURCEN

Einsatz von Ressourcen

- Ressourcen (Bilder, Skripte oder Stylesheets)
  - In JSF-Seiten über Tags
  - Als Abhängigkeiten in Komponenten
  - Aber auch Kompositkomponenten sind Ressourcen!
- Ressource hat einen Namen...

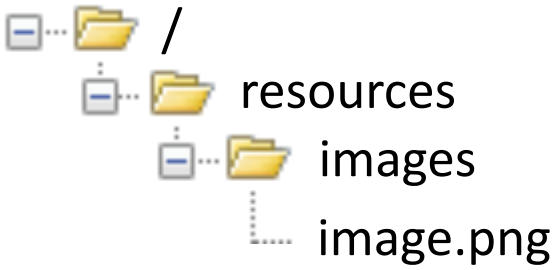
```
<h:graphicImage name="image.png"/>
```

- ... und liegt optional in einer Bibliothek

```
<h:graphicImage library="images" name="image.png"/>
<h:outputScript library="scripts" name="script.js"/>
<h:outputStylesheet library="css" name="style.css"/>
```

Auflösung von Ressourcen

- Realer Pfad wird abstrahiert
- Auflösung von Ressourcen in JSF (Default!):
  1. In **/resources** in der Webapplikation
  2. In **/META-INF/resources** im Classpath
- Bibliothek als Verzeichnis interpretiert



- Positionierung in der Seite
- **h:head** und **h:body** nicht vergessen!

JSF 2.2: Konfigurierbares Verzeichnis

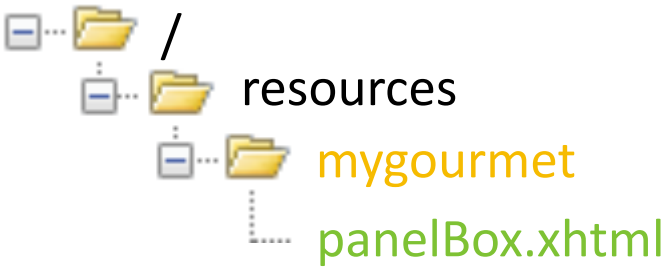
- /resources von außen sichtbar
- Unsauber besonders bei Kompositkomponenten
  - Interne Strukturen werden preisgegeben
  - Seiten, die nicht direkt gerendert werden sollen/können, sind aufrufbar
- Neuer Context-Parameter:

```
<context-param>
  <param-name>
    javax.faces.WEBAPP_RESOURCES_DIRECTORY
  </param-name>
  <param-value>/WEB-INF/resources</param-value>
</context-param>
```

EINFÜHRUNG KOMPOSITKOMPONENTEN

Einsatz einer Kompositkomponente

- Kompositkomponente **panelBox.xhtml**



- Verwendung in einer Seite (Neue Namespaces seit JSF 2.2!)

```
<html xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:mc="http://xmlns.jcp.org/jsf/composite/mygourmet">
  <mc:panelBox title="Panel-Header">
    <h:outputText value="Erster Text"/>
    <h:outputText value="Zweiter Text"/>
  </mc:panelBox>
</html>
```

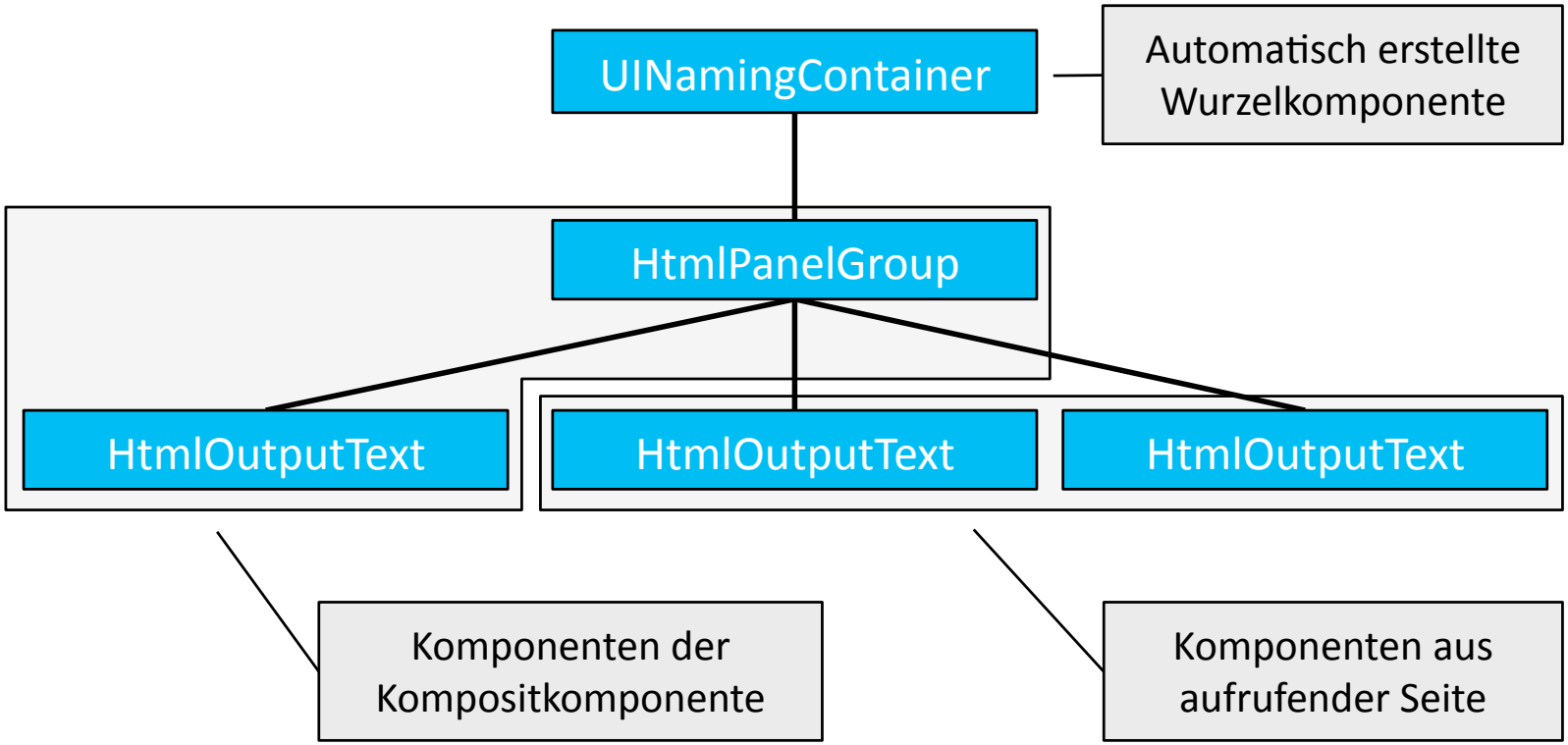
Beispiel einer Kompositkomponente

- Kompositkomponente **panelBox.xhtml**

```
<html xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:cc="http://xmlns.jcp.org/jsf/composite">
<head><title>Panel box component</title></head><body>

    <cc:interface>
        <cc:attribute name="title"/>
    </cc:interface>

    <cc:implementation>
        <h:panelGroup layout="block">
            <h:outputText value="#{cc.attrs.title}"/>
            <cc:insertChildren/>
        </h:panelGroup>
    </cc:implementation>

</body></html>
```

Resultierender Komponentenbaum

- Teilbaum der Komponente `panelBox.xhtml`



cc:interface

- Schnittstelle der Komponente nach außen
- Definition von Attributen und Facets
  - **cc:attribute**
  - **cc:facet**
- Verschachtelte Attribute

```
<cc:attribute name=”bean" required="true”>
    <cc:attribute name=”collapsed" required=“true”/>
</cc:attribute>
```

- Verhalten interner Komponenten
  - **cc:valueHolder**
  - **cc:editableValueHolder**
  - **cc:actionSource**

Attribute

- Attribute

```
<cc:attribute name="label" required="true"/>
<cc:attribute name="text" required=false/>
<cc:attribute name="header" default="Header"/>
```

- Attribute mit Typ

```
<cc:attribute name=”max" type=”java.lang.Integer"/>
<cc:attribute name=”pageBean" type=”at.jsflive.SortablePage"/>
```

Beispiel für cc:interface 1/2

- Kompositkomponente `simpleInput.xhtml`

```
<cc:interface>
  <cc:attribute name="label" default="Input"/>
  <cc:attribute name="action" targets="submit"/>
  <cc:attribute name="value" required="true"/>
  <cc:editableValueHolder name="input"/>
  <cc:actionSource name="submit"/>
</cc:interface>

<cc:implementation>
  <h:outputLabel for="input" value="#{cc.attrs.label}"/>
  <h:inputText id="input" value="#{cc.attrs.value}"/>
  <h:commandButton id="submit" value="Submit"/>
</cc:implementation>
```

Beispiel für cc:interface 2/2

- `simpleInput.xhtml` im Einsatz

```
<mc:simpleInput action="#{myBean.save}"
    value="#{myBean.longValue}">
  <f:actionListener for="submit"
      binding="#{myBean.saveListener}"/>
  <f:validateLength for="input" minimum="10"/>
  <f:valueChangeListener for="input"
      binding="#{myBean.valueChanged}"/>
</mc:simpleInput>
```

Methodenübergabe 1/2

- `targets`
  - `action`, `actionListener`, `validator`, `valueChangeListener`
  - Jeweils nur ein Attribut möglich
- `method-signature`

```
<cc:interface>
    <cc:attribute name="action" targets="submit"/>
    <cc:attribute name="cancelAction"
            method-signature="java.lang.String action()"/>
</cc:interface>

<cc:implementation>
    <h:commandButton id="submit"/>
    <h:commandButton action="#{cc.attrs.cancelAction}” />
</cc:implementation>
```

Methodenübergabe 2/2

- Ab JSF 2.1: Attribut `targetAttributeName`
  - mehrere interne Verlinkungen
  - direkte Übergabe eines Strings

```
<cc:interface>
   <cc:attribute name="submitAction" targets=”submit"
                 targetAttributeName="action"/>
   <cc:attribute name="cancelAction" targets="cancel"
                 targetAttributeName="action"/>
</cc:interface>

<cc:implementation>
   <h:commandButton id="submit"/>
   <h:commandButton id="cancel"/>
</cc:implementation>
```

cc:implementation

- Implizites Objekt **cc**
  - Aktuelle Kompositkomponente
  - **#{cc.attrs.myAttribute}**
  - **#{cc.facets.myFacet}**
- Kindkomponenten einfügen:
  - **<cc:insertChildren/>**
- Facets aus aufrufender Seite verwenden:
  - **<cc:insertFacet name="header"/>**
  - **<cc:renderFacet name="header"/>**

Beispiel für cc:implementation 1/2

- Kompositkomponente `panelBox.xhtml`

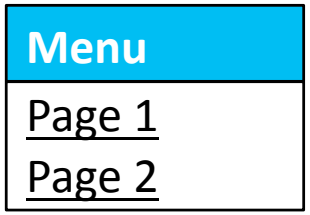
```
<cc:interface>
  <cc:facet name="header" required="false"/>
</cc:interface>

<cc:implementation>
  <h:panelGroup layout="block">
    <c:if test="#{!empty cc.facets.header}">
      <p><cc:renderFacet name="header"/></p>
    </c:if>
    <cc:insertChildren/>
  </h:panelGroup>
</cc:implementation>
```

Beispiel für cc:implementation 2/2

- Komponente `panelBox.xhtml` mit Facets

```
<mc:panelBox id="menu">
  <f:facet name="header">
    <h:outputText value="Menu"/>
  </f:facet>
  <h:panelGrid columns="1">
    <h:commandLink action="page1" value="Page 1"/>
    <h:commandLink action="page2" value="Page 2"/>
  </h:panelGrid>
</mc:panelBox>
```

- Mögliche Ausgabe:



Demonstration: collapsible01

Kompositkomponente `collapsiblePanel`

- Komponente erwartet Toggle-Model und Action Methode
- Collapsible State wird in einem Controller gehalten
- Scope ist damit nicht per se ViewScope
- Action muss selber implementiert werden

Benutzerdefinierte Wurzelkomponente

- Spezielles Verhalten in Java implementieren
- Wurzelkomponente über Komponententyp

```
<cc:interface componentType="at.jsflive.CollapsiblePanel"/>
```

```
@FacesComponent("at.jsflive.CollapsiblePanel")
public class CollapsiblePanel extends UINamingContainer{}
```

- Direkter Zugriff auf Eigenschaften

```
<h:panelGroup rendered="#{cc.visible}">
  <cc:insertChildren/>
</h:panelGroup>
```

Demonstration: collapsible02

Benutzerdefinierte Wurzelkomponente

- Kompositkomponente hält eigene Komponentenklasse
- State wird intern gehalten, liegt damit im ViewScope
- Action wird von Kompositkomponente implementiert
- Collapsible Wert kann optional in einem Controller liegen

Properties in Kompositkomponenten

- resourceBundleMap in Basisklasse UIComponent
- ResourceBundle per Package
  - Klasse der Kompositkomponente: at.jsflive.component.CollapsiblePanel
  - ResourceBundle: at.jsflive.component.CollapsiblePanel.properties
- ResourceBundle in /resources/libName
  - Kompositkomponente collapsiblePanel.xhtml
  - collapsiblePanel.properties

Die eigene Komponentenbibliothek

- JAR mit Artefakten erstellen
  1. Kompositkomponenten in **/META-INF/resources**
  2. Konfiguration (.taglib.xml) in **/META-INF**

```
<facelet-taglib version="2.0" ...>
  <namespace>http://at.irian/mygourmet</namespace>
  <composite-library-name>
       mygourmet
  </composite-library-name>
</facelet-taglib>
```

  3. Komponenten, Konverter, Validatoren...
  4. **faces-config.xml** in **/META-INF**

Demonstration: collapsible03

Kompositkomponente in eigenem .jar Artefakt

- Separates Maven Modul
- Eigene Taglibrary
- ResourceBundle in Package der Komponentenklasse
- Umschalten der Locale

Kompositkomponenten in Taglibraries

- < JSF 2.2
  - Eine Taglibrary/Namespace per Bibliothek
  - Ressourcenname = Tagname
- >= JSF 2.2
  - Unterschiedliche Bibliotheken, ein gemeinsamer Namespace möglich
  - Anderer Tagname als Ressourcenname möglich

```
<facelet-taglib version="2.2" ...>
    <namespace>http://jsflive.at/taglib</namespace>
  <tag>
    <tag-name>collapsible</tag-name>
    <component>
      <resource-id>
        jsflive/collapsiblePanel.xhtml
      </resource-id>
    </component>
  </tag>
</facelet-taglib>
```

Demonstration: ccs-in-taglibs

Kompositkomponenten in Taglibraries

- Unterschiedliche Bibliotheken, ein Namespace
- Filename != Tagname
- ResourceBundle in /resources/libName

Demonstration: html5

HTML 5 Passthrough Attribute

- HTML 5 Input Date
- HTML 5 Input Color

Neugierig?

- Michael Kurz, Martin Marinschek: **JavaServer Faces 2.2**, dpunkt.Verlag
- IRIAN JSF@Work Online-Tutorial **http://jsfatwork.irian.at**
- JSFlive Weblog **http://jsflive.wordpress.com**

JSF live