

Einführung in JavaServer Faces

Michael Müller

Grundlagen und Beispiele

Übersicht

- User Interface Framework für Webanwendungen
- Besonders geeignet für datengetriebene Anwendungen
Sind das heute nicht die meisten?
- Frontend für Enterpriseanwendungen

Übersicht

- Auszug aus der Spezifikation:
 - Makes it easy to construct a UI from a set of reusable UI components
 - Simplifies migration of application data to and from the UI
 - Helps manage UI state across server requests
 - Provides a simple model for wiring client-generated events to server-side application code
 - Allows custom UI components to be easily built and re-used

Übersicht

- Handhabung Komponentenstatus über mehrere Requests hinweg
- Formularverarbeitung, auch über mehrere Seiten
- Streng typisiertes serverseitiges Evenmodel für die vom GUI erzeugten Ereignisse
- Seitennavigation als Antwort auf bestimmte Ereignisse

Nutzung diverser Technologien

- Servlet
- Managed Beans
- CDI
- Bean Validation
- JPA
- Expression Language
- ...

JavaServer Faces selbst ist als Servlet implementiert und benötigt daher einen entsprechenden Container (Application Server)

Entwicklung, zeitlich

- 2004 JSR 127 JavaServer Faces
- 2006 JSR 252 JavaServer Faces 1.2
- 2009 JSR 314 JavaServer Faces 2.0
Zahlreiche Neuerungen: AJAX, Facelets, ...
- 2010ff JavaServer Faces 2.1
Maintenance Release
- 2013 JSR 344 JavaServer Faces 2.2
HTML5 friendly markup, WindowID, Flows, ... → nachfolgender Vortrag

Entwicklung, technologisch

- **Servlet**

Verarbeitung eines Requests, Rendern der HTML-Seite

- **JSP**

HTML-Seite mit eingestreuten Tags, wird zu Servlet kompiliert

- **JSF**

Komponentenframework, Lebenszyklus, Navigation über mehrere Seiten → Applikation

Entwicklung, technologisch

- **VDL**

View Definition Language, austauschbar

- **JSP**

JavaServer Pages, als VDL der ersten Stunde genutzt. Kann zahlreiche Neuerungen ab JSF 2.0 nicht nutzen. Kompatibel zu JSF 1.x Anwendungen

- **Facelets**

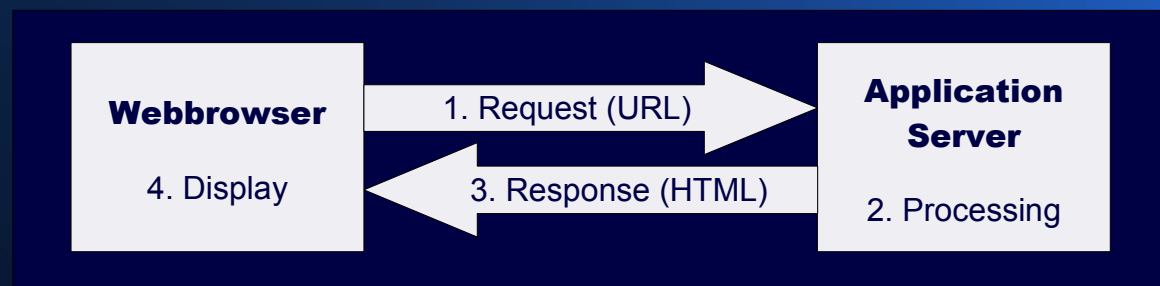
Als alternative VDL von Jacob Hookom für den JSF Lebenszyklus entwickelt. Seit JSF 2.0 Bestandteil der Spezifikation und bevorzugte VDL.

Implementierungen, Komponentenbibliotheken

- Mojarra (RI)
- MyFaces
- RichFaces
- PrimeFaces
- IceFaces
- OpenFaces
- ...

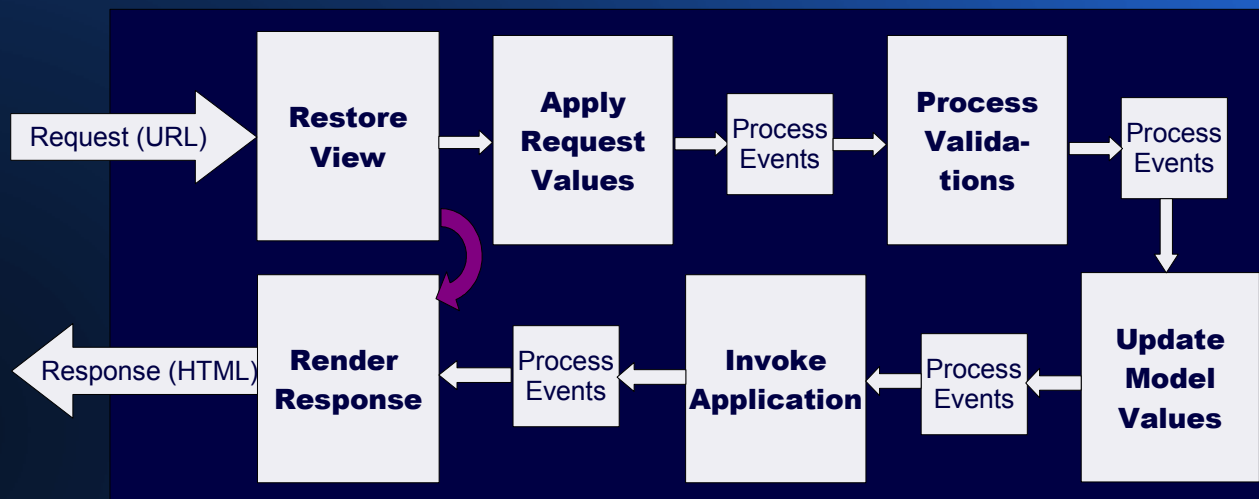
Requestzyklus

- Zustandslos
- Server lauscht auf Anfrage, versendet Antwort und kappt die Verbindung



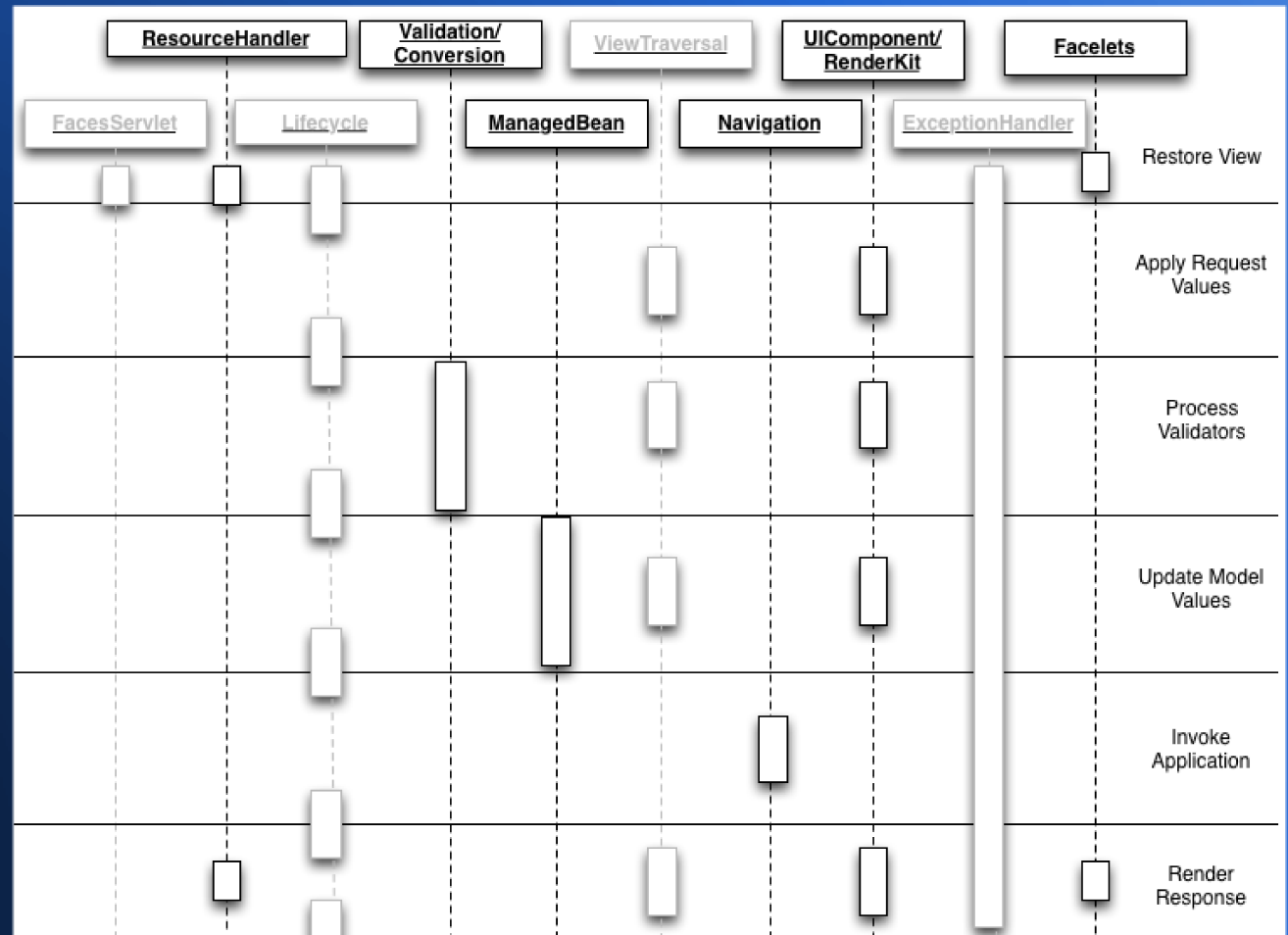
JSF Lebenszyklus

- Initialer Request: Restore View, Render Resp.
- Folge request: Fluss durch alle Phasen
- Abweichungen in Process Events möglich
→ Response Complete (End Lifecycle), → Render Response



JSF Lebenszyklus

- Quelle:
Ed Burns



Sessionverwaltung

- Problem HTTP: Zustandslos
- JavaServer Faces übernimmt eine Sessionverwaltung
- Zuordnung der Anwendersitzung zu einem komplexen serverseitigen Ablauf
- Eine Session muss nicht notwendigerweise mit der gesamten Nutzungsdauer im Browser übereinstimmen

z.B. Generierung einer neuen Session aus Sicherheitsgründen bei An- oder Abmeldung

HTML Seite und Komponentenbaum

- HTML Seite im Browser und stark vereinfachter Komponentenbaum

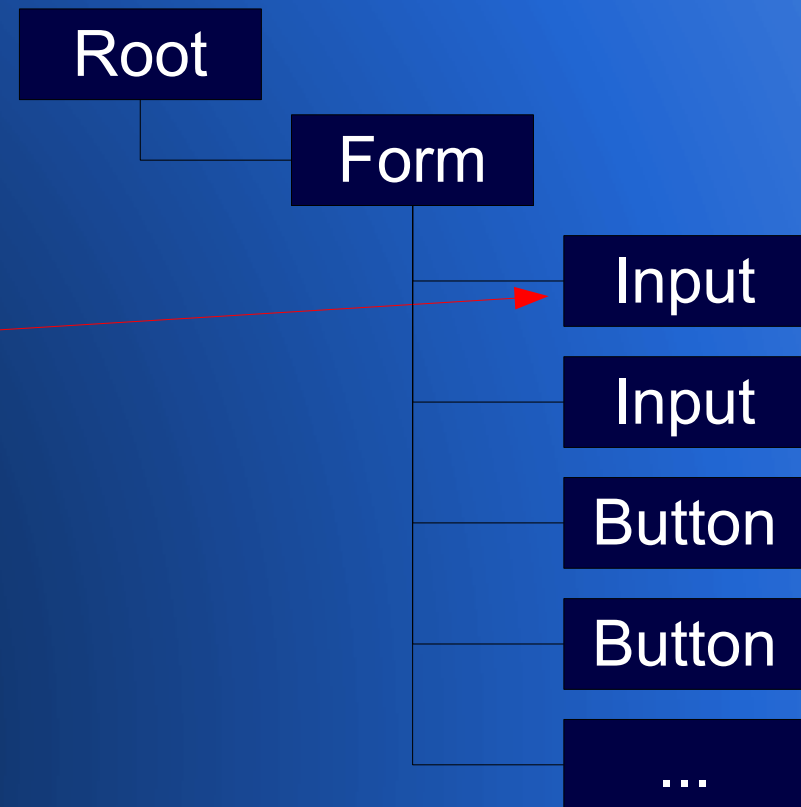
Tiny calculator

Param 1: 0

Param 2: 0

add subtract multiply divide

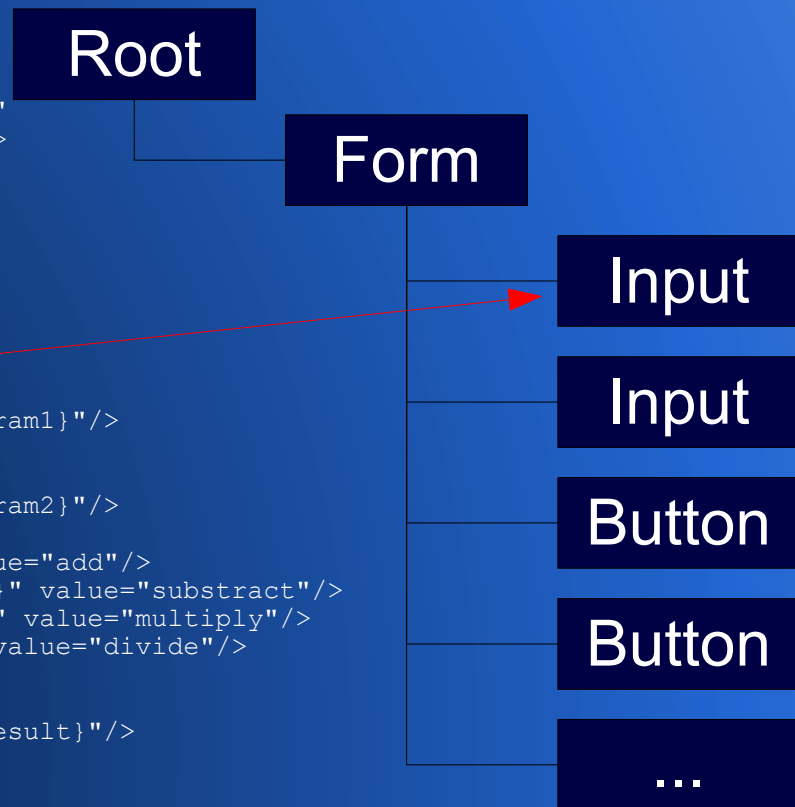
Result:



HTML Seite und Komponentenbaum

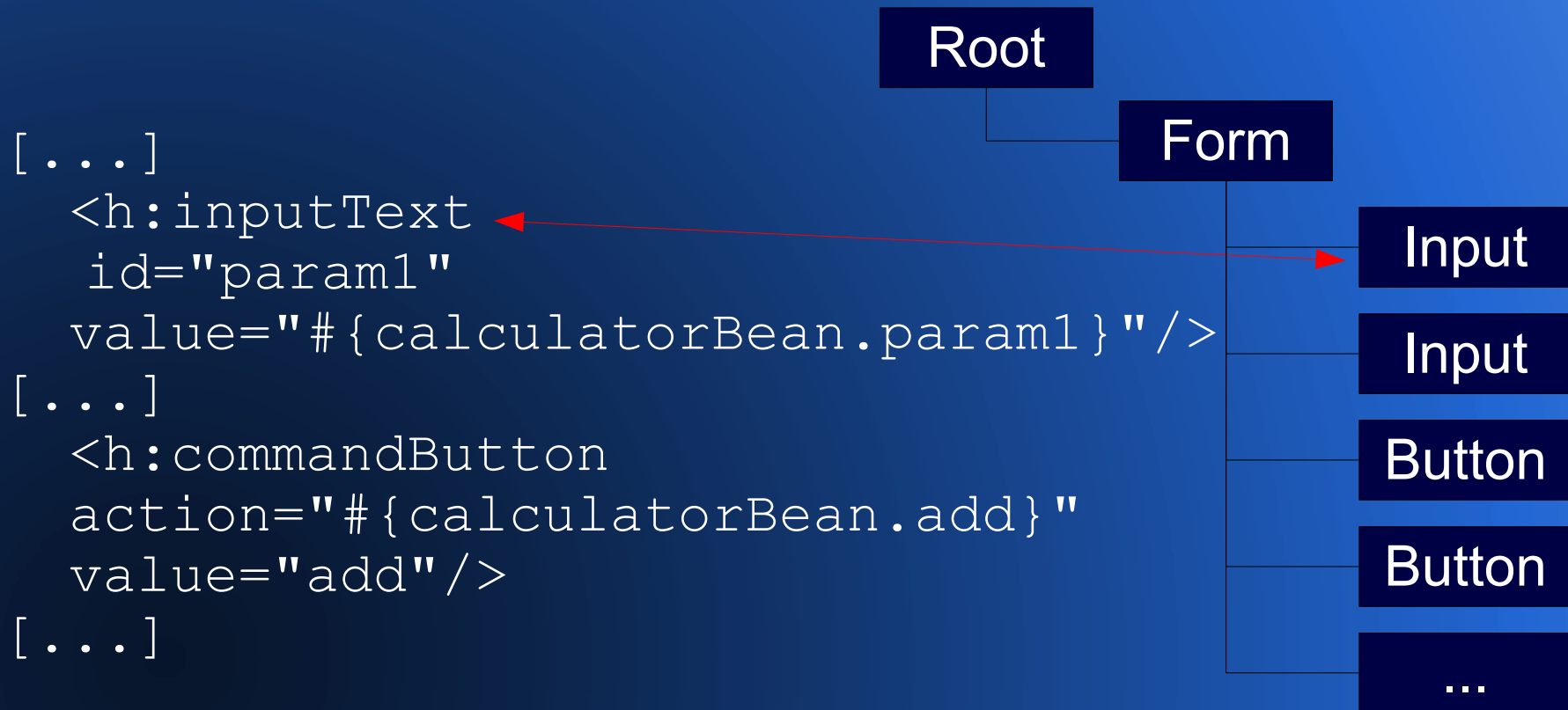
- HTML Seite und stark vereinfachter Komponentenbaum

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:head >
    <title>Tiny calculator</title>
  </h:head>
  <h:body>
    <h1>Tiny calculator</h1>
    <h:form id="calculator">
      <h:outputLabel for="param1" value="Param 1:"/>
      <h:inputText id="param1" value="#{calculatorBean.param1}"/>
      <br/>
      <h:outputLabel for="param2" value="Param 2:"/>
      <h:inputText id="param2" value="#{calculatorBean.param2}"/>
      <br/>
      <h:commandButton action="#{calculatorBean.add}" value="add"/>
      <h:commandButton action="#{calculatorBean.subtract}" value="subtract"/>
      <h:commandButton action="#{calculatorBean.multiply}" value="multiply"/>
      <h:commandButton action="#{calculatorBean.divide}" value="divide"/>
      <br/>
      <h:outputLabel for="result" value="Result:"/>
      <h:outputText id="result" value="#{calculatorBean.result}"/>
    </h:form>
  </h:body>
</html>
```



HTML Seite und Komponentenbaum

- Nichts zu erkennen? Hier der Zoom:

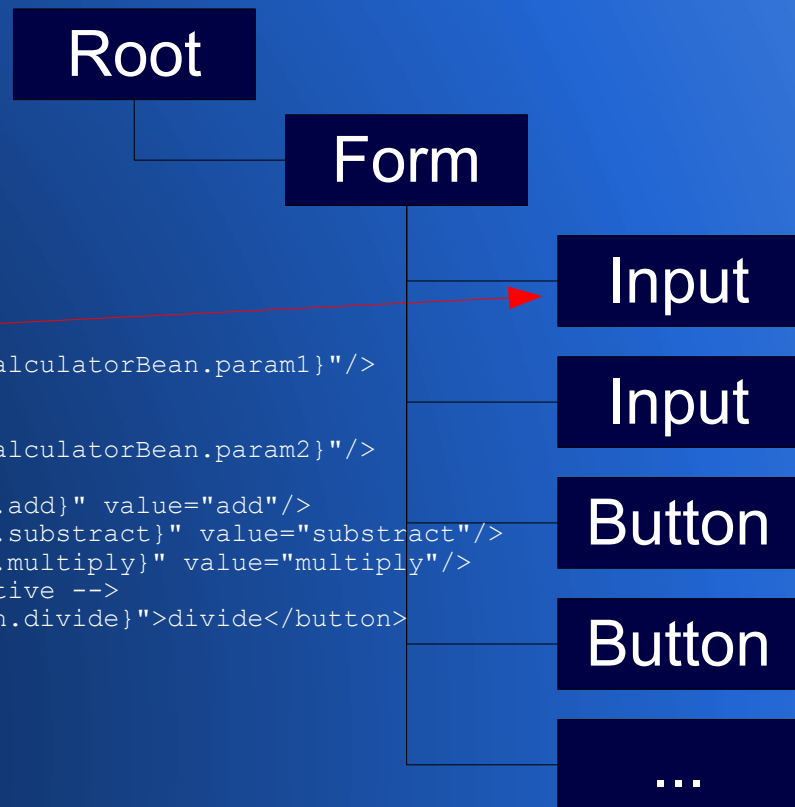


HTML Seite und Komponentenbaum

- Und als HTML5 Friendly Markup (JSF 2.2)

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsf="http://java.sun.com/jsf" >
  <head jsf:id="head">
    <title>Tiny calculator</title>
  </head>
  <body jsf:id="body">
    <h1>Tiny calculator</h1>
    <form jsf:id="calculator">
      <label jsf:for="param1">Param 1: </label>
      <input jsf:id="param1" type="text" jsf:value="#{calculatorBean.param1}"/>
      <br/>
      <label jsf:for="param2">Param 2: </label>
      <input jsf:id="param2" type="text" jsf:value="#{calculatorBean.param2}"/>
      <br/>
      <input type="submit" jsf:action="#{calculatorBean.add}" value="add"/>
      <input type="submit" jsf:action="#{calculatorBean.subtract}" value="subtract"/>
      <input type="submit" jsf:action="#{calculatorBean.multiply}" value="multiply"/>
      <!-- following example shows button as an alternative -->
      <button type="submit" jsf:action="#{calculatorBean.divide}">divide</button>

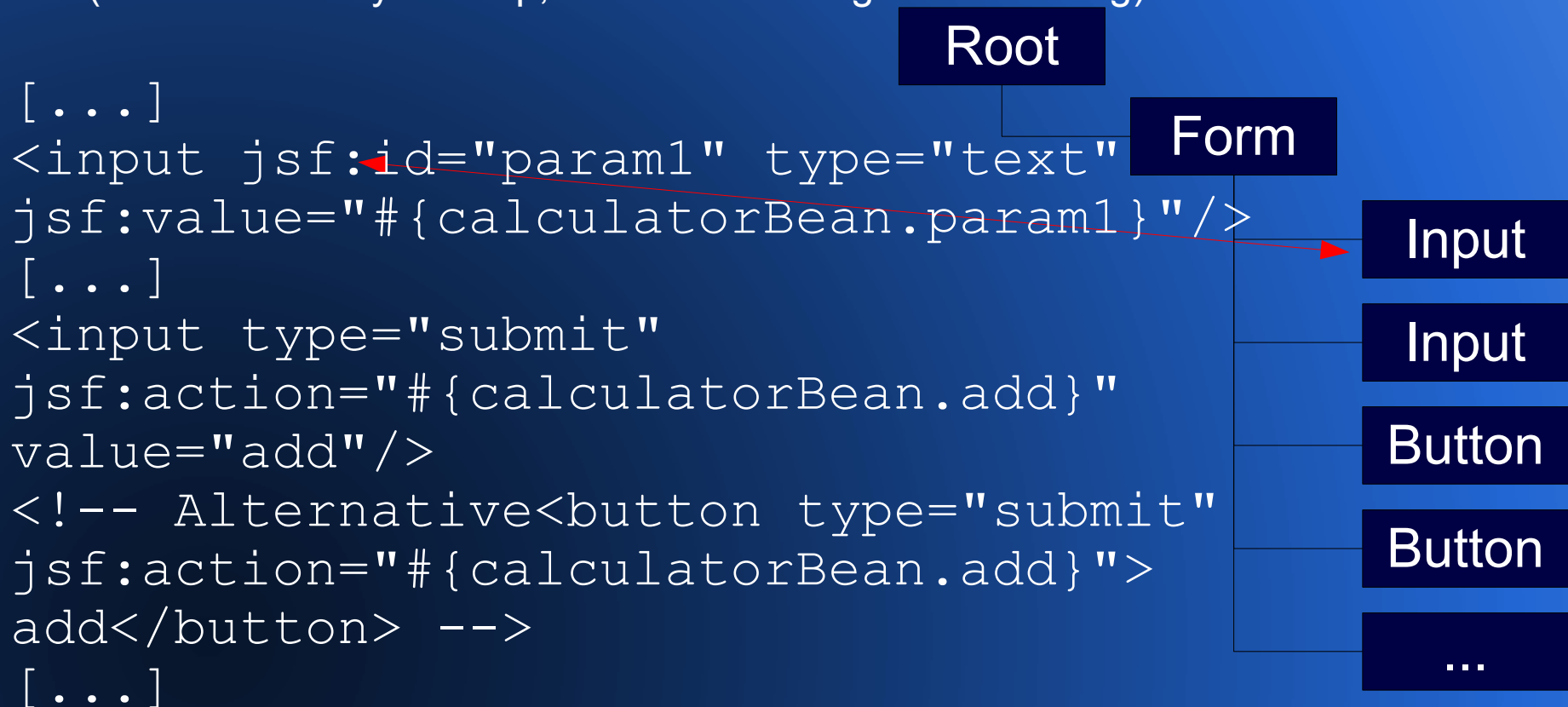
      <br/>
      <label>Result: #{calculatorBean.result}</label>
    </form>
  </body>
</html>
```



HTML Seite und Komponentenbaum

- Zoom in Pass Through Elements

(HTML5 Friendly Markup, mehr im nachfolgenden Vortrag)

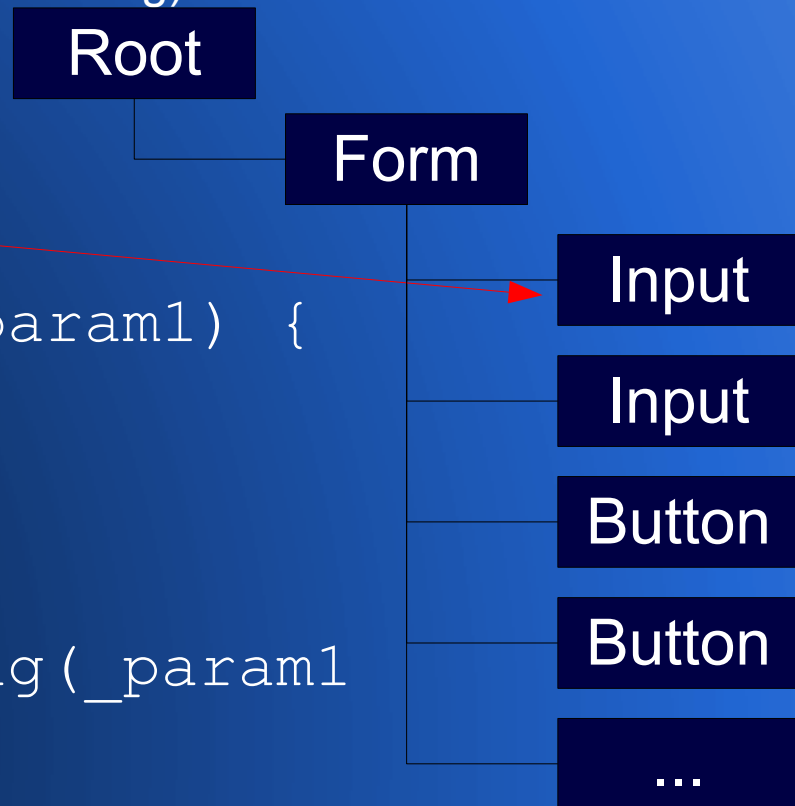


Komponentenbaum und Model

- Zoom in Pass Through Elements

(HTML5 Friendly Markup, mehr im nächsten Vortrag)

```
public int getParam1() {  
    return _param1;  
}  
public void setParam1(int param1) {  
    _param1 = param1;  
}  
[...]  
public String add() {  
    _result = Integer.toString(_param1  
                               + _param2);  
    return "";  
}
```




HTML-Seite und Model

- Nutzung der Expression Language

```
<h:inputText  
  id="param1"  
  value="#{calculatorBean.param1}"/>
```

```
public int getParam1() {  
    return _param1;  
}  
public void setParam1(int param1) {  
    _param1 = param1;  
}
```



Navigation

- Einfach als Rückgabewert einer Aktion
- Null oder Leerstring verbleibt auf der Seite
- String gibt Ziel an

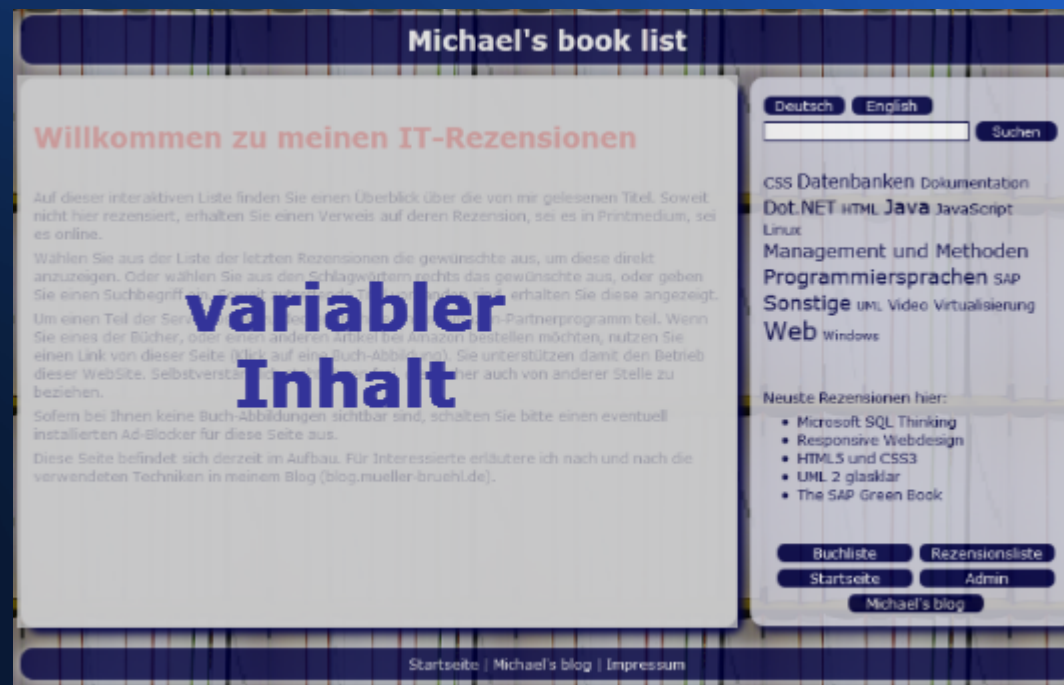
```
public String add() {  
    [...]  
    return "";  
}
```

```
public String add() {  
    [...]  
    return "result.xhtml";  
}
```

- Zielseite dynamisch bestimmbar
- Navigation via Konfiguration (XML)

Templates

- Templates sind Schablonen
- „Fester“ Inhalt mit Fenster zu variablen Teilen



Templates

- Auszug aus einem Template

```
[...]
<div id="top">
    <h1>Michael's book list</h1>
</div>
<div id="main">
    <ui:insert name="content">Content</ui:insert>
</div>
<div id="sidebar">
    <h:form id="sidebarform">
[...]
```

Templates

- Nutzung des Templates

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE ...>
<ui:composition
    xmlns:ui="http://java.sun.com/jsf/facelets"
    template="booksTemplate.xhtml" ...>
    <ui:define name="content">
        [...seitenspezifischer Inhalt...]
    </ui:define>
</ui:composition>
```


Sonstiges

- Messages
- Ressourcen
- Validation
- JPA
- ...

→ Demonstration an Codebeispielen

X

- X

X

- X