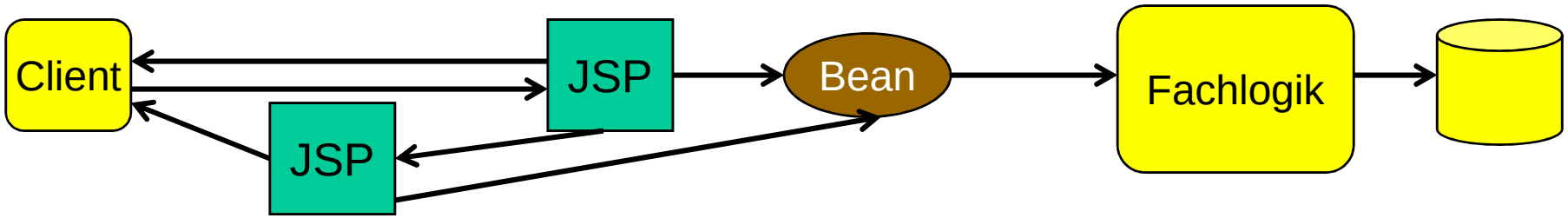
Modell 2 und JSF

- JSP und Modell 1
- MVC und Modell 2
- Java Web Frameworks und Struts
- Java Server Faces

JSP – ein mächtiges Werkzeug!



- Eine JSP empfängt Benutzer-Anfragen, "Requests"
- erstellt und benutzt Beans, die die Fachlogik aufrufen
- erzeugt und sendet eine Response
- oder verzweigt für die Response zu einer anderen JSP

- Im Prinzip ist damit alles machbar!

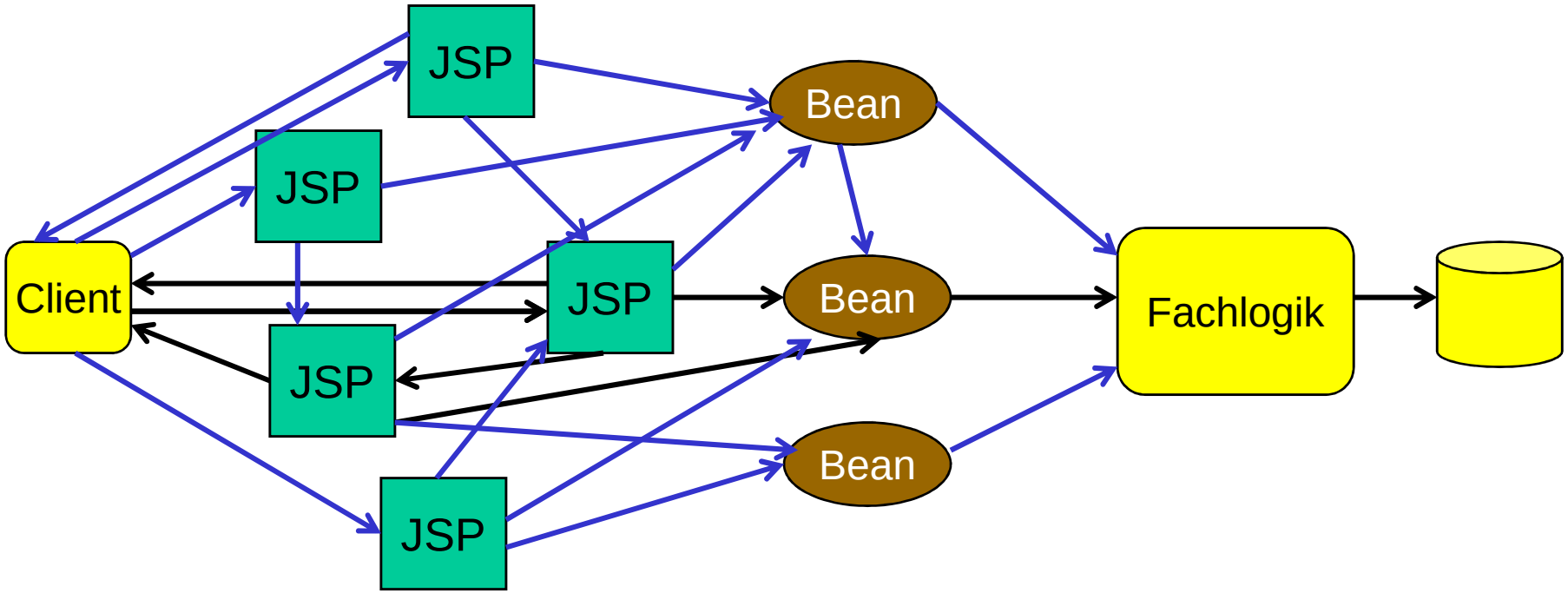
JSP Details (Review)

- Instanziierung und Benutzung von Beans
  - <jsp:usebean>-Direktive
  - das angegebene Bean wird gesucht oder instanziiert
  - Problem: verteilte Erzeugung und Verwaltung von Beans

- Kontexte
  - Kontexte bestimmen die Lebensdauer der Beans
  - Kontexte gemeinsam von allen JSP-s nutzbar
    - application
    - session
    - request
    - page

- Navigation
  - einfache HTML-Links
  - <jsp:forward>
  - <jsp:redirect>

Modell 1



- Jede JSP reagiert selbständig auf Benutzeranfragen
- erstellt und benutzt Beans, die die Fachlogik aufrufen
- verzweigt ggf. für die Response zu einer anderen JSP

Probleme mit Modell 1

- Jede JSP-Seite reagiert selbständig auf Anfragen
  - Verteilte Kontrolle
  - Verteilte Navigation
  - Harte "Verdrahtung" der Anbindung an das "Backend"

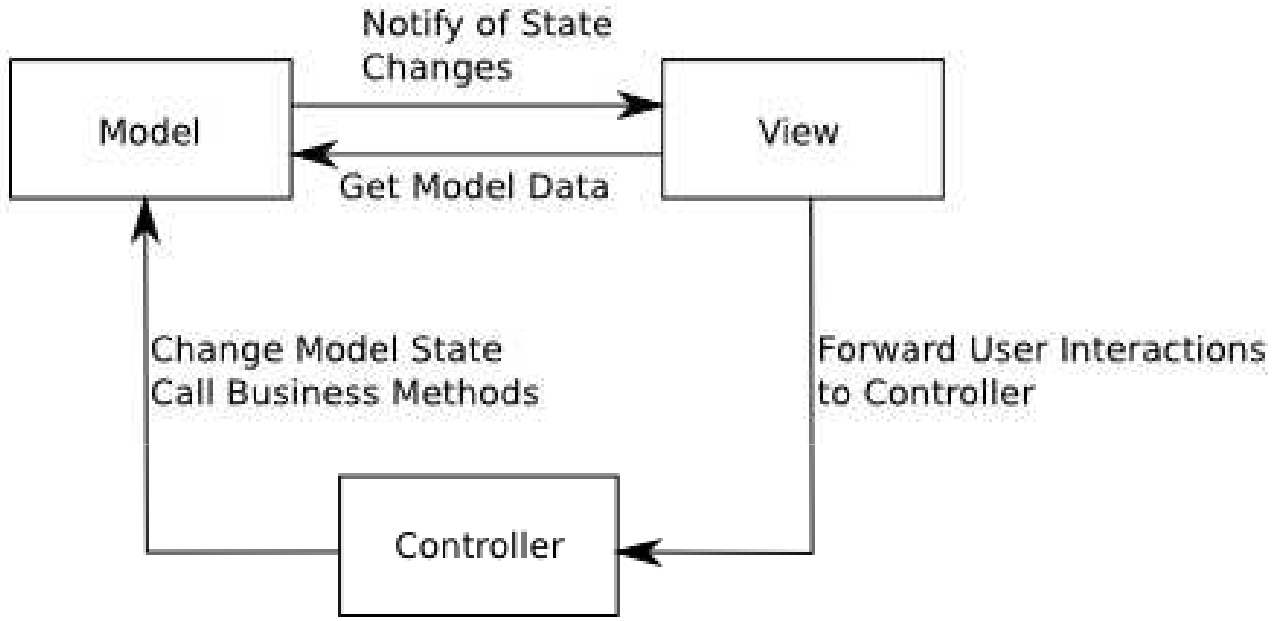
  → unübersichtlich

  → uneinheitlich

  → schlecht zu ändern

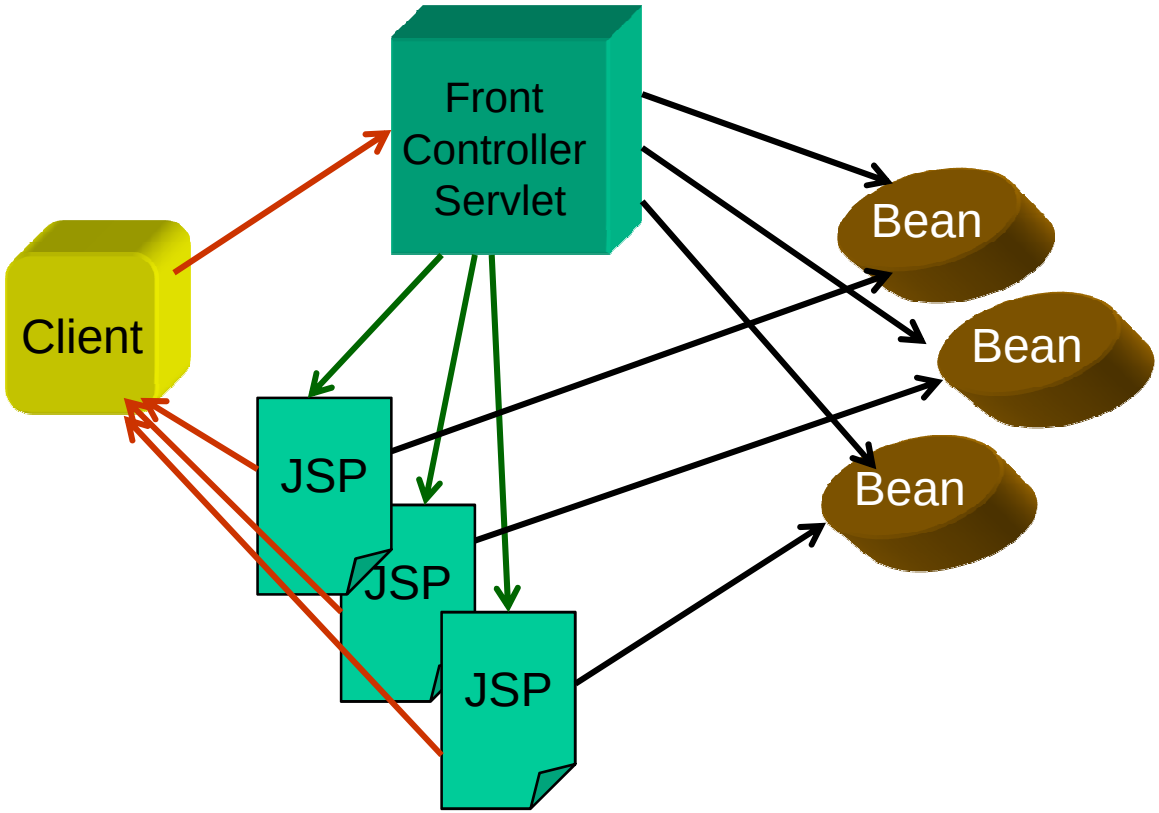
- Abhilfe
  - Rückbesinnung auf MVC

MVC - klassisch



- ***Model*** *(auch Domain Model oder Fachlogisches Modell) enthält und verwaltet die persistenten Daten und die Fachoperationen darauf*
- ***View*** *liest die erforderlichen Daten aus dem Model, bereitet sie auf und präsentiert sie.*
- ***View*** *registriert sich beim Model, um Veränderungen zu erfahren*
- ***Controller*** *nimmt die Nutzeranfragen entgegen und leitet sie an die zuständigen Stellen weiter.*

MVC 2 – oder Modell 2



- Front Controller
  - empfängt alle Requests
  - instanziiert und füllt Beans
  - ruft die Aktionen
  - leitet an die passende Antwort-JSP weiter.
- JSP
  - liest Beans
  - erzeugt und sendet Response

Front Controller Servlet

- Das Servlet übernimmt die zentrale Steuerung.
- Alle Parameter, Beans und Seession-Attribute den sind über den Servlet-Context erreichbar
- Navigation ist mithilfe der Klasse RequestDispatcher möglich
- Allerdings: Etwas mühsame Programmierung ☹

Skalierbarkeit von Modell 2

- Modell 2 macht Web-Anwendungen beliebiger Größe prinzipiell beherrschbar:
  - klare Trennung zwischen Darstellung und Kontrolle
  - alle Komponenten separat entwickelbar
  - beliebig komplexes Modell anschließbar
  - zentralisierte Ablaufsteuerung
  - Aufgabenteilung zwischen Designern und Programmierern

...dennoch harte Arbeit

- Front-Controller-Servlet sieht immer ähnlich aus, muss aber jedesmal neu geschrieben werden.
- Alle Navigationen sind "hart verdrahtet", d.h. die URL steht im Code.
- Die Zuordnung von Bean-Attributen zu GUI-Komponenten muss einzeln ausprogrammiert werden.
- Die Zuordnung von Aktionen zu GUI-Ereignissen muss über die Request-Parameter ausprogrammiert werden.
- Validierung, Textvarianten und Internationalisierung müssen von Hand gelöst werden.

Programmieren ist weniger schlimm als Ändern

Die Antwort heißt

Web Application Frameworks

- Definieren Objekttypen und ihre Zusammenarbeit:
  immer MVC
- Bieten (erweiter- oder konfigurierbare) Standardobjekte
- Ersetzen harte Verdrahtung der Seitennavigation durch Konfiguration
- Unterstützen die Zuordnung von Beans
- Unterstützen den Aufbau von GUIs

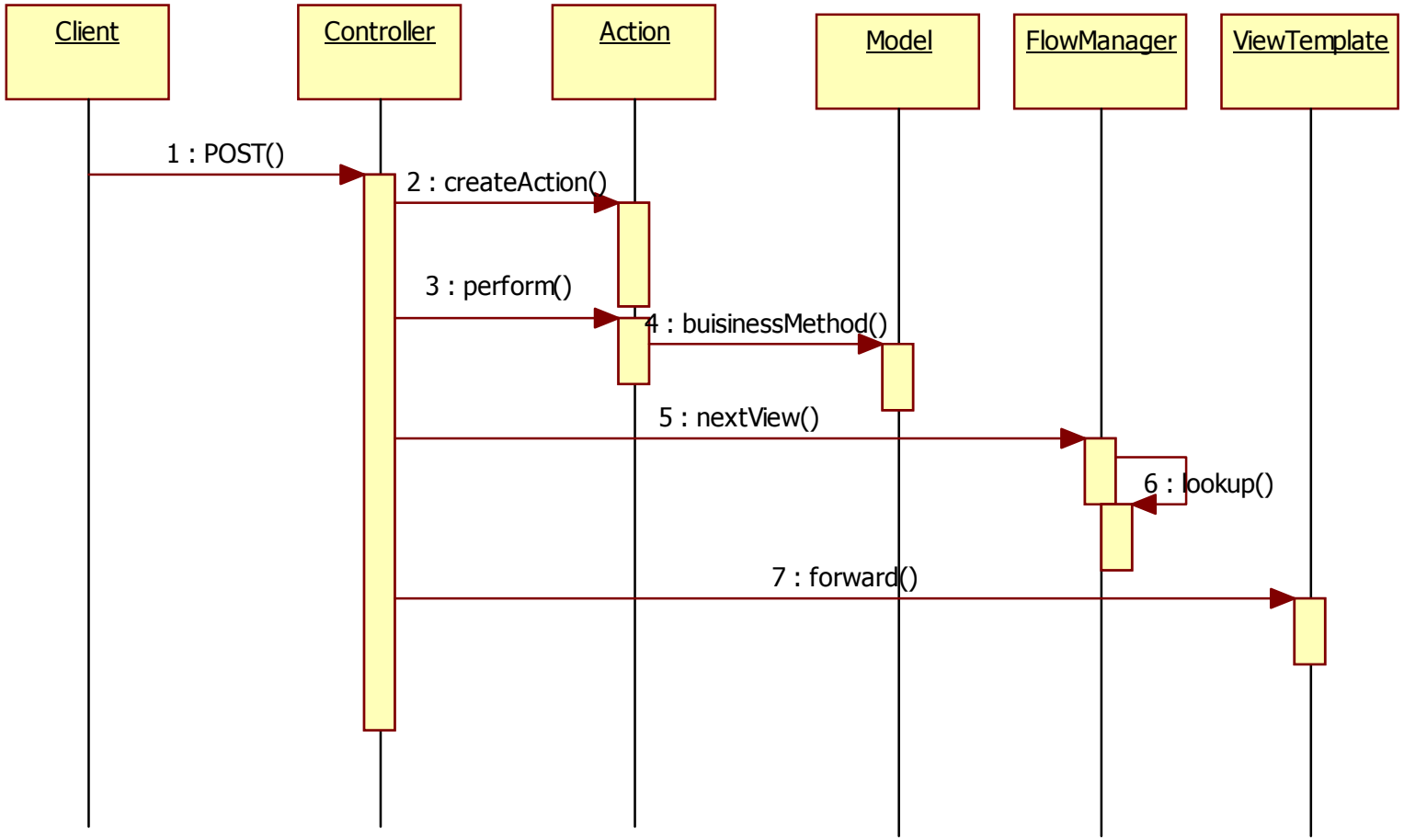
→ deutlich verbesserte Wiederverwendbarkeit der Komponenten.

Java Web-Frameworks:

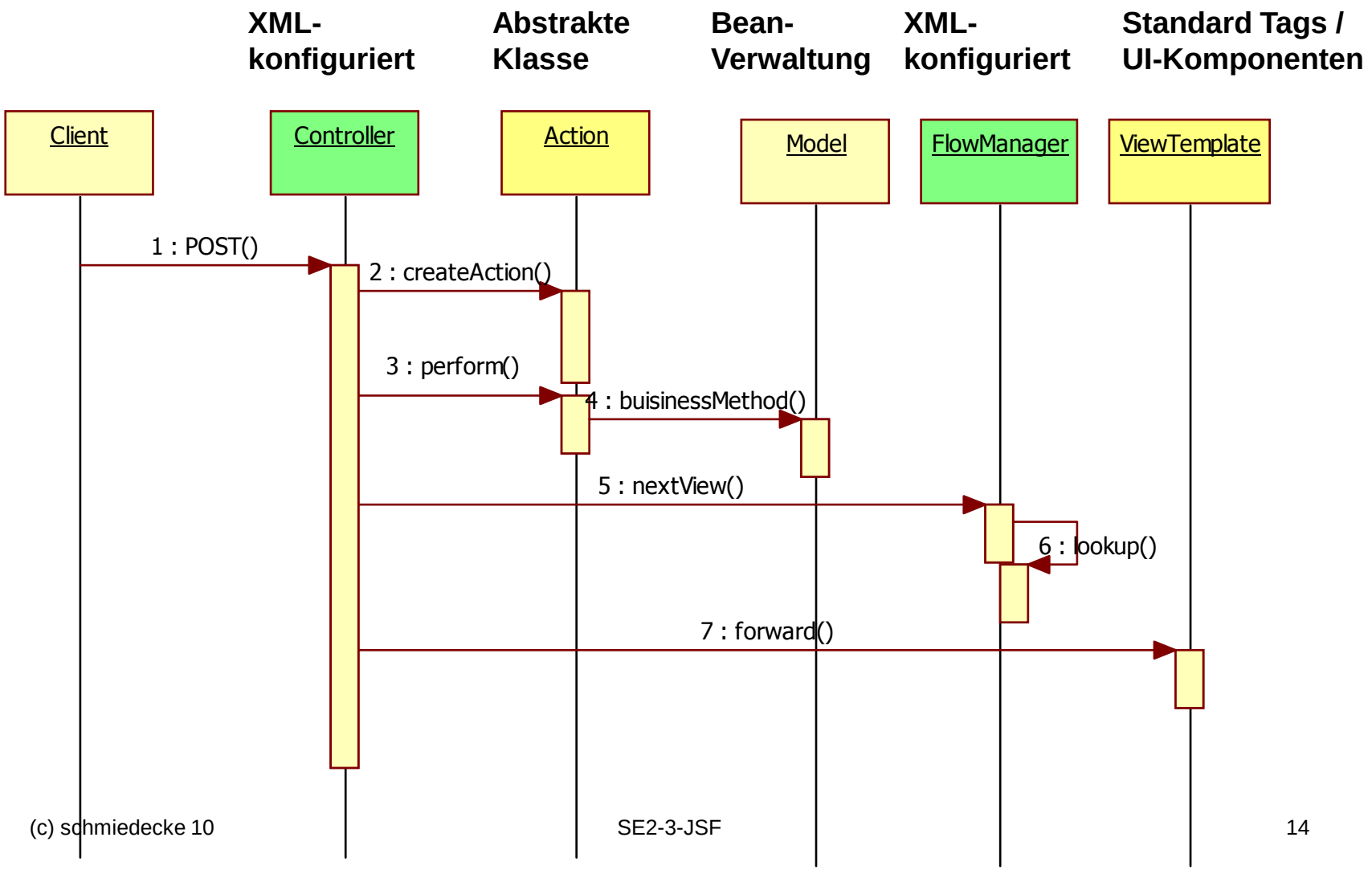
… erleichtern die Arbeit :

- Apache Struts
- Java Server Faces
- Echo
- Tapestry
- GWT
- ...

Was leisten Web-Frameworks?

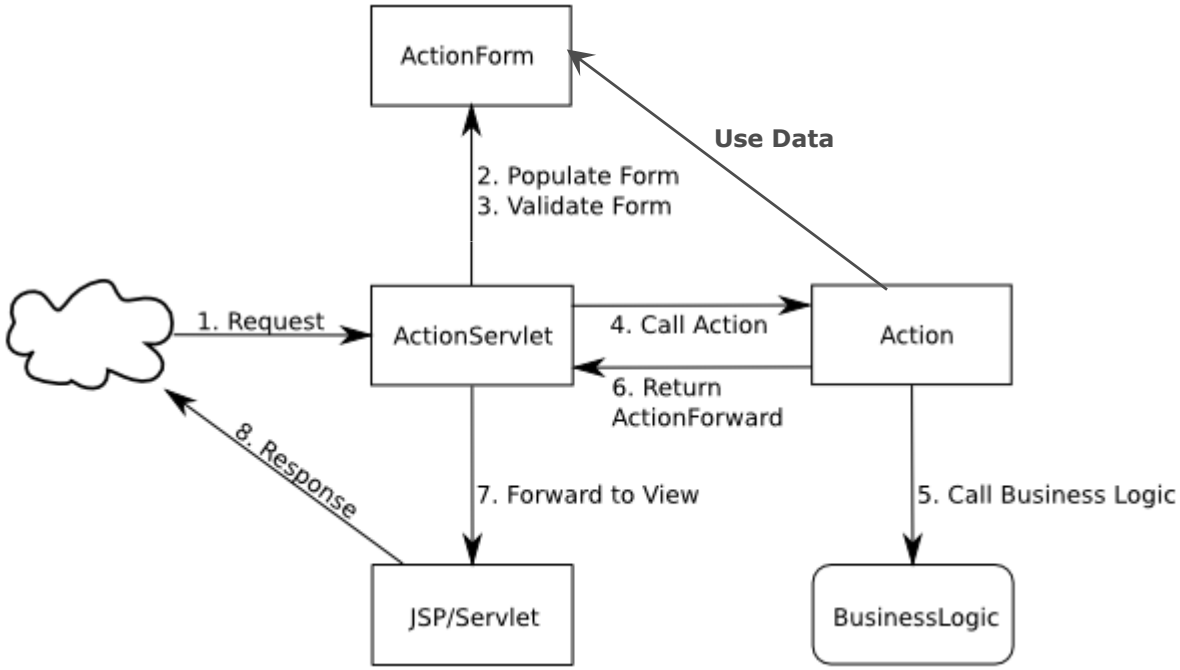


Was leisten Web-Frameworks?



Kurzer Blick auf Apache Struts

Struts - "Mutter aller Web Frameworks"



Quelle: jsptutorial.org

ActionServlet
- findet oder erzeugt passendes ActionForm-Objekt und füllt es mit Request-Daten
- ruft passendes Action-Objekt

Action-Objekt
- ruft Fachlogik
- nutzt und ändert Daten in ActionForm-Objekt
- erzeugt ActionForward-String

ActionServlet
- selektiert JSP aufgrund des ActionForward-Strings
- verzweigt dorthin

JSP/Servlet
- nutzt ActionForm-Daten
- erzeugt und sendet Response

Struts - Konzepte

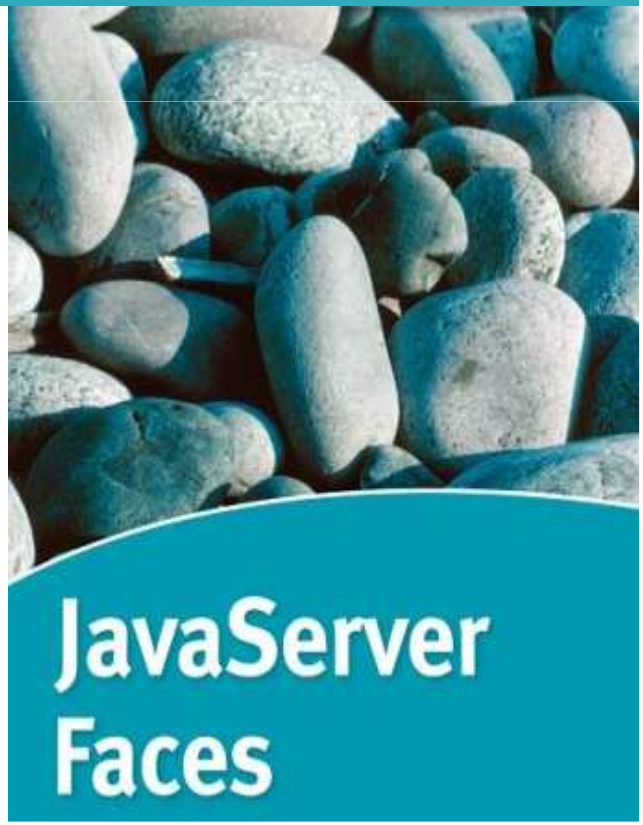
- ActionForm und Action sind abstrakte Klassen, die der Benutzer erweitert.
- Das ActionServlet ist fertig
  - es wird durch eine xml-Datei konfiguriert: stuts-config.xml
  - enthält insbesondere auch die Navigation in Abhängigkeit von ActionForward-Strings
- Für die Gestaltung der Oberfläche und die Interaktion gibt es eine komfortable "Tag-Library"
- *Details dazu vielleicht in einem Fachvortrag!*

Java Server Faces - JSF

Java Server Faces gilt als Nachfolge-Technologie zu Struts
"Vater" ist der Struts-Designer J.McCallahan

JSF-Ziele

- Web-Frontendgestaltung mit Fertigkomponenten
  - erweiterbare Komponentenbibliotheken
  - client- und serverseitige Validierung
- Visuelle Gestaltung
  - "Zusammenklicken" der Oberflächen
  - visuelle Navigations-Konfiguration
- Swing-ähnliche Programmierung
  - Ereignisbehandlung & Ajax-Funktionalität
  - "Eingebaute" Accessability-Unterstüzung
- Spezifikation
  - es gibt verschiedene Implementierungen



JSF - Features

- ✓ Klare Model2-Architektur
  - ✓ Front Controller heißt FacesServlet
- ✓ JSF-UI-Komponenten als Bausteine für Seiten
  - ✓ verschiedene Renderer möglich (z.B. für Handys)
  - ✓ ein- und ausblendbar
  - ✓ View-Definition-Languages: Facelets und JSP
  - ✓ Ereignisbehandlung
- ✓ Datentransfer
  - ✓ Konfigurierbare Managed Beans
  - ✓ Einfaches Data Binding an Managed Beans über EL (Expression Language)
  - ✓ Konvertierung und Validation
- ✓ Konfigurierbare Seitennavigation
- ✓ Unterstützung von Meldungen
- ✓ Internationalisierungs- und Zugänglichkeits-Unterstützung

Faces Servlet in web.xml

- Faces-Servlet wird als zentraler Einstiegspunkt festgelegt

```
<!-- Faces Servlet -->
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup> 1 </load-on-startup>
  </servlet>
  <!-- Faces Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>MyApp/*</url-pattern>
  </servlet-mapping>
```

JSF-Entwicklungsprozess beginnt beim UI

1. *Entwirf die UIs und binde die Datenfelder an das Datenmodell*

2. *Entwickle ein (Interaktions-)Datenmodell aus Beans*

3. *Entwirf die Seitennavigation abhängig von Ergebnissen*

4. *Führe die "Verdrahtung" im web.xml und faces-config.xml bzw. durch Annotationen durch*

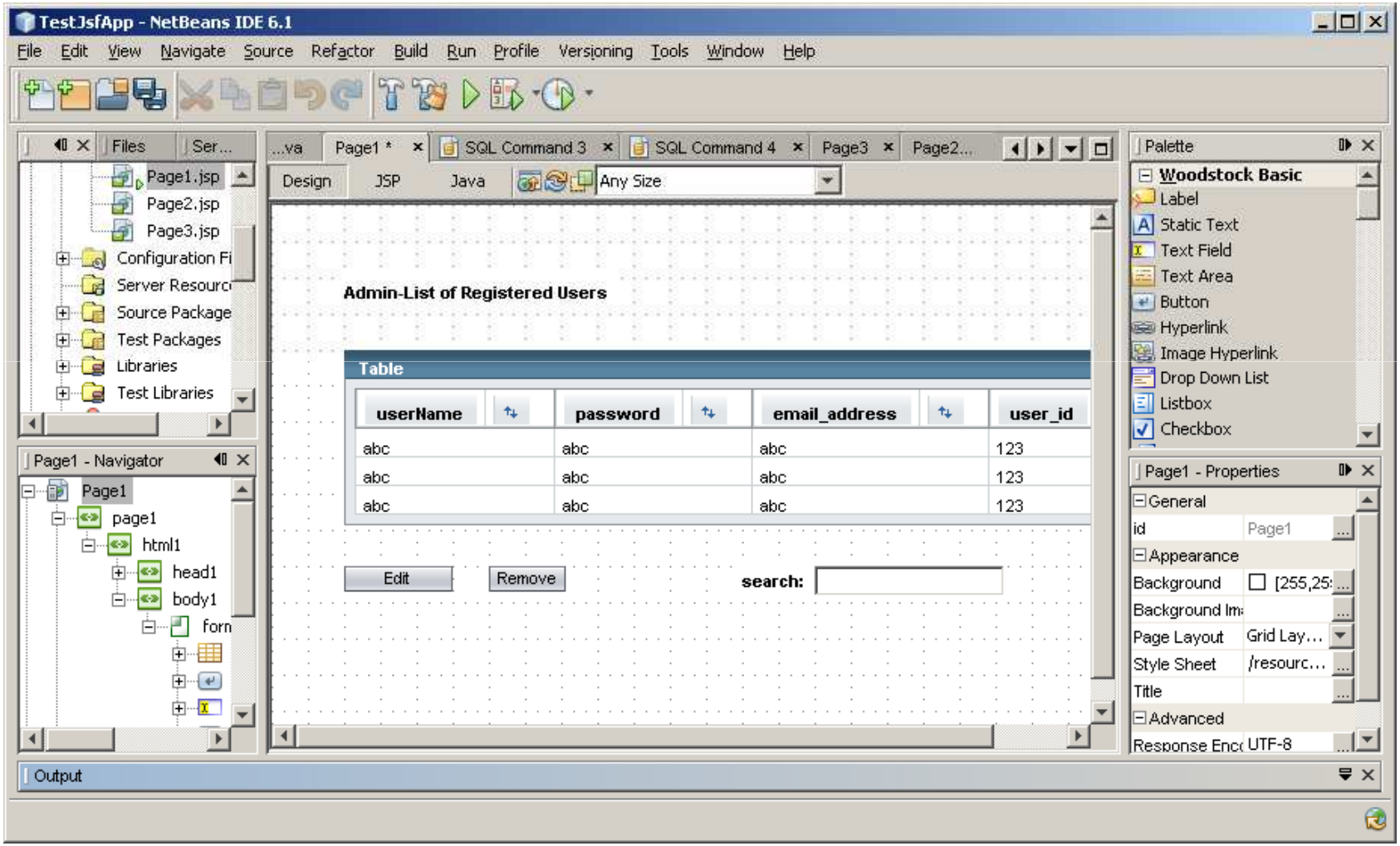
JSF-UI-Komponenten

- Ähnlich Swing-Komponenten, grafisch zusammenstellbar
- Instanziierung und Platzierung durch JSF-tags:

```
<h: form id="customer_form">
  <h: panelGrid  id="grid"  columns="2" >
    <h: outputLabel  value="First Name:" for "firstnameField"/>
    <h: inputText  id="firstnameField" value="#{customer.firstName}"/>
    …
    <h: commandButton  id="saveBtn" action="#{customer.save}"
                                  value="Save"/>
  </h:panelGrid>
</h:form>
```

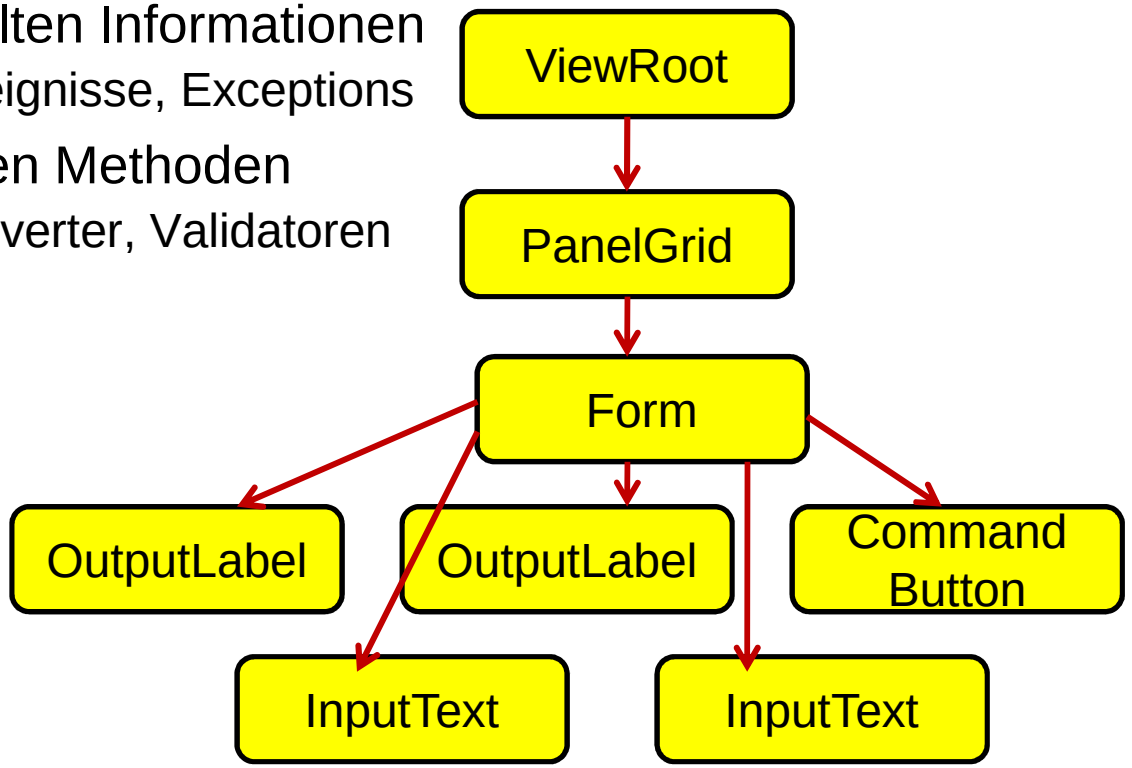
- VDL – View Definition Language
  - Facelets sind XHTML-Seiten (erscheinen im Browser mit der Endung .jsf)
  - JSP wird auch unterstützt.

Visuelle UI-Gestaltung mit JSF-Komponenten



Der Komponentenbaum

- Grundlage der Verarbeitung
  - Komponenten enthalten Informationen
    z.B. Eingaben, Ereignisse, Exceptions
  - Komponenten kennen Methoden
    z.B. Aktionen, Konverter, Validatoren



Managed Beans

- JSF instanziiert und verwaltet alle deklarierten Beans
- Managed Beans sind über Managed Properties verknüpfbar
- Deklaration in der faces-config.xml
- Alternativ durch Annotationen

```
@ManagedBean (name="address")
@SessionScoped
class AddressBean {
  String surname;
  String Address;

}
```

```
@ManagedBean (name="mail")
@SessionScoped
class MailBean {
  String mailprovider;
  String mailaddress;
  @ManagedProperty
    (value="#{address}")
  AddressBean ref;
}
```

- Mithilfe der EL ansprechbar:
  #{address.surname} #{mail.ref.surname}
- Scopes: none /request / view / session / application
- Lazy Creation

Deklaration in der faces-config.xml

```
class MailBean {
  String mailprovider;
  String mailaddress;
  AddressBean ref;

}
```

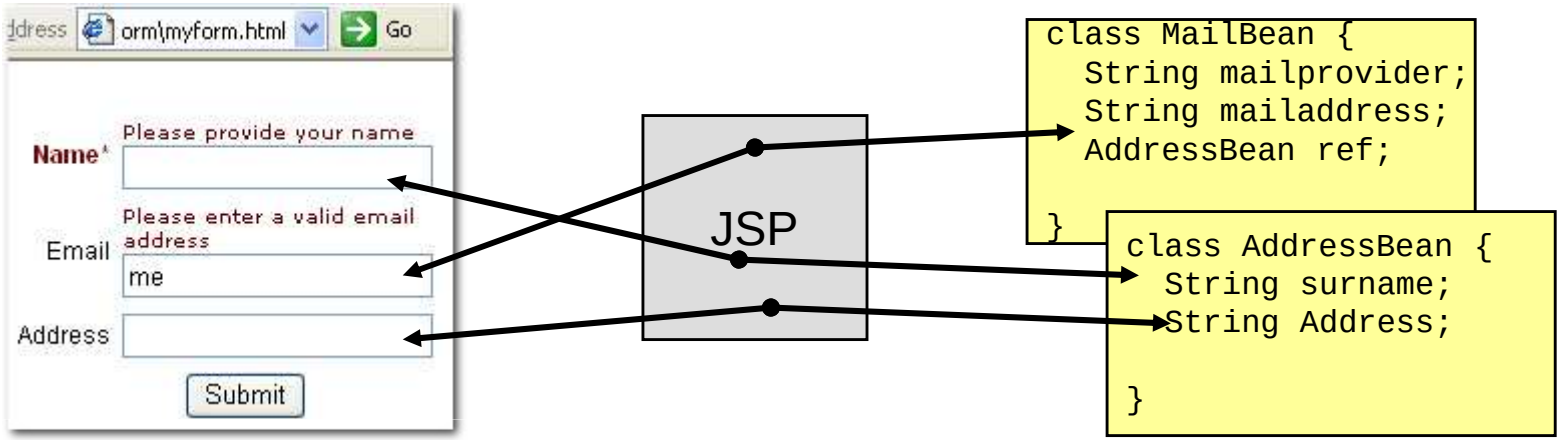
```
class AddressBean {
  String surname;
  String Address;

}
```

Deklaration von Manages Beans und Managed Properties

```
<managed-bean>
    <managed-bean-name>address</managed-bean-name>
    <managed-bean-class> firstForm.AddressBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>mail</managed-bean-name>
    <managed-bean-class> firstForm.MailBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property> <property-name>ref</property-name>
    <value>address</value> </managed-property>
</managed-bean>
```

Managed Beans und Data Binding



...

File register.jsp

```
<f:view> <h:form>
    <b>Please provide your name</b><br/>
     <h:outputText value="Name*"/></td>
     <h:inputText id="name" required="true" value="#{address.surname}" size="20">
        <f:validateLength minimum="2" maximum="12"/>
     </h:inputText>
     <h:message for="name" style="color: red;"/>
    <p> ...
</f:view>
```

ruft setter

Die Unified EL

- *Unified: JSP-EL und JSF-EL sind kompatibel (seit JSF 2.0)*

- Beispiele für Ausdrucksmöglichkeiten:

- value="#{address.surname}"
- rendered="#{address.surname != null}"
- value="#{address.zip * 100 + 55}" <!-- schlechter Stil -->
- value="Guten Tag, Herr #{address.surname}"

- action="#{address.saveAddr}" <!-- Aufruf **`saveAddr()`**; -->

Aktionen

- Definiert in Managed Beans
- Parameterlose Funktionen mit String-Rückgabe

- Anbindung als action-Attribut von

  h:commandButton, h:commandLink

```
<h: commandButton  id="saveBtn"
                   action="#{address.saveAddr}"
                        value="Save"/>
```

- Rückgabewert bestimmt die Navigation zur Folgeseite

Navigation

- Navigation ist immer die Folge einer Aktion

- statische Navigation:
  das Action-Attribut gibt direkt das Ziel an:

```
<h: commandButton  id="cancelBtn"
                   action="cancelled"
                        value="Cancel"/>
```

- dynamische Navigation:
  das Action-Attribut ist ein Aufruf, der Rückgabewert gibt das Ziel an:

```
<h: commandButton  id="saveBtn"
                   action="#{address.saveAddr}"
                        value="Save"/>
```

Statische Navigation über faces-config.xml

register.jsp enthält:

```
<h:commandButton id="submit" action="submitted" value="Submit" />
```

faces-config.xml enthält:

```
<navigation-rule>
    <from-view-id>/register.jsp</from-view-id>
    <navigation-case>
      <from-outcome>submitted</from-outcome>
      <to-view-id>/welcome.jsf</to-view-id>
    </navigation-case>
</navigation-rule>
```

Dynamische Navigation über faces-config.xml

register.jsp enthält:

```
<h:commandButton id="submit" action="#{MailBean.checkMailAddress}" value="Submit" />
```

faces-config.xml enthält:

```
<navigation-rule>
    <from-view-id>/register.jsp</from-view-id>

    <navigation-case>
    <from-action>#{MailBean.checkMailAddress}</from-action>
    <from-outcome>valid</from-outcome>
    <to-view-id>/welcome.jsf</to-view-id>
    </navigation-case>

    <navigation-case>
    <from-action>#{MailBean.checkMailAddress}</from-action>
    <from-outcome>invalid</from-outcome>
    <to-view-id>/invalidMail.jsf</to-view-id>
    </navigation-case>
</navigation-rule>
```

```
class MailBean {
  String mailprovider;
  String mailaddress;
  AddressBean ref;
  String checkMailAddress();
}
```

Vereinfachte Alternative in JSF 2.0

- statisch

register.jsp enthält:

```
<h:commandButton id="submit" action="/welcome.jsf" value="Submit" />
```

- dynamisch

register.jsp enthält:

```
<h:commandButton id="submit" action="#{MailBean.checkMailAddress}"
                 value="Submit" />
```

checkMailAddress() gibt entweder "/welcome.jsf" oder "/invalid.jsf" zurück

- In beiden Fällen reicht auch der Dateiname, z.B. "welcome", da automatisch nach einer Datei mit eine VDL-Endung gesucht wird.

Konverter

- Konvertierung:
  - HTML enthält nur Strings
  - beim Speichern in Beans: String nach Zahl, Datum, …
  - beim Lesen aus Beans: Zahl, datum,… nach String

- Standardkonverter:
  - f:convertDateTime

    Eigenschaften type, dateStyle, timeStyle, pattern, timeZone, locale

    immer nützlich: pattern="dd.MM.yyyy"
  - f:convertNumber

    Eigenschaften type, currencyCode, locale, pattern …

    immer nützlich: pattern="#.###,##" oder pattern ="#,##"

- Verwendung

```
<h:outputText  value="#{order.date}">
    <f:convertDateTime  pattern="dd/MM/yyyy" />
<outputText>
```

Validierung

- Zwei Standardwege zur Validierung:

- Bean-Validierung
  - Spezifikation der korrekten Werte durch Annotation an den Bean-Properties.
  - Wichtig: in web.xml den Parameter *INTERPRET_EMPTY_STRING_SUBMITTED_VALUES_AS_NULL* auf true setzen!

- JSF-Validierung
  - vorgegebene f:validateXXX – Tags
  - validator-Attribute an den Input Tags verweisen auf eigene Validatormethoden (in Beans)

Bean-Validierung

```
public class Address {
   @NotNull @Min(value=1000) @Max(value=9999)
   private Integer zipCode;

   @Past
   private Date registeredSince;
```

@AssertFalse, @AssertTrue

@Max, @Min,

@Digits,

@Size

@NotNull, @Null

@Past, @Future

@Pattern

JSF-Standard-Validatoren

- f:validateLength

```
<h:inputText value="#{userBean.úser_id}">
    <f:validateLength minimum="3" maximum="7" />
</h:inputText>
```

- f:validateLongRange
- f:validateDoubleRange
- f:validateRegex

- f:validateRequired

Benutzerdefinierte Validatoren

Definition

- Validator-Methode im Bean schreiben
- Signatur:

```
public void myValidate (
  FacesContext ctx, UIComponent cmp, Object val)
  throws ValidatorException;
```

- Benutzung
- mit validator-Attribut:

```
<h:inputText value="#{myBean.someVal}"
             validator="#{myBean.myValidate}" />
```

Meldungen

- JSF unterstützt die Ausgabe von Meldungen
- Klasse `javax.faces.application.FacesMessage`
- Konstruktoraufruf z.B.:

```
msg = new FacesMessage (
          FacesMessage.SEVERYTY_WARN,
          "This is my Warning Text", null);
```

- FacesMessages können als Exception-Parameter mitgegeben werden:

```
throw new ValidatorException(msg);
```

- Sie können im FacesContext gesammelt werden:

```
FacesContext.addMessage(msg);
```

Meldungen anzeigen

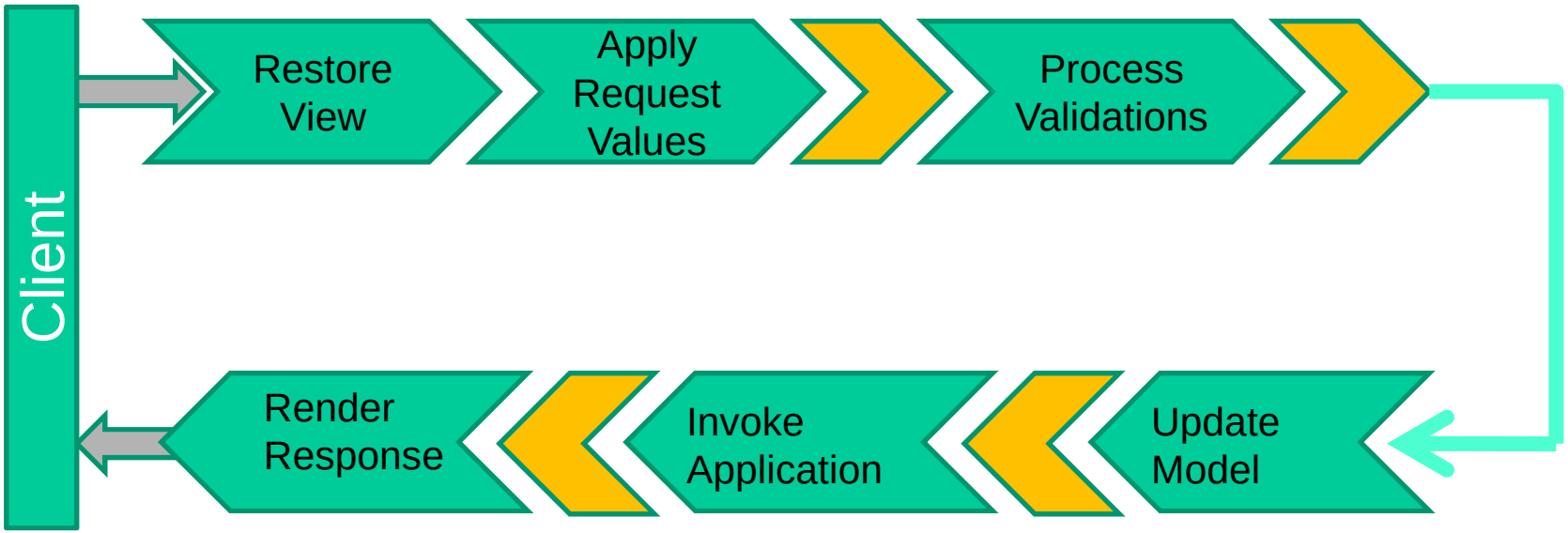
- Um Meldungen anzuzeigen, wird ein <h:messages …>-Tag benötigt
- JSF 2.0 fügt es ggf. automatisch ein

(nur im Development-Status)

```
<h:messages showSummary="false" showDetail="true"/>
```

Der große Zusammenhang: Der Lifecycle

- Eine HTTP-Anfrage an eine JSF-Seite wird in 6 Phasen abgearbeitet.
- Zwischen den Phasen können Ereignisse behandelt werden – *davon nächste Woche.*
- In bestimmten Situationen werden Phasen übersprungen.



Konfigurationsdatei oder Annotation?

- Pro Annotation:
  - Im Code erkennbar
  - keine extra-Datei

- Pro Konfigurationsdatei
  - zentrale Information
  - ohne Kompilation änderbar

- Mischung??
  - XML siegt….

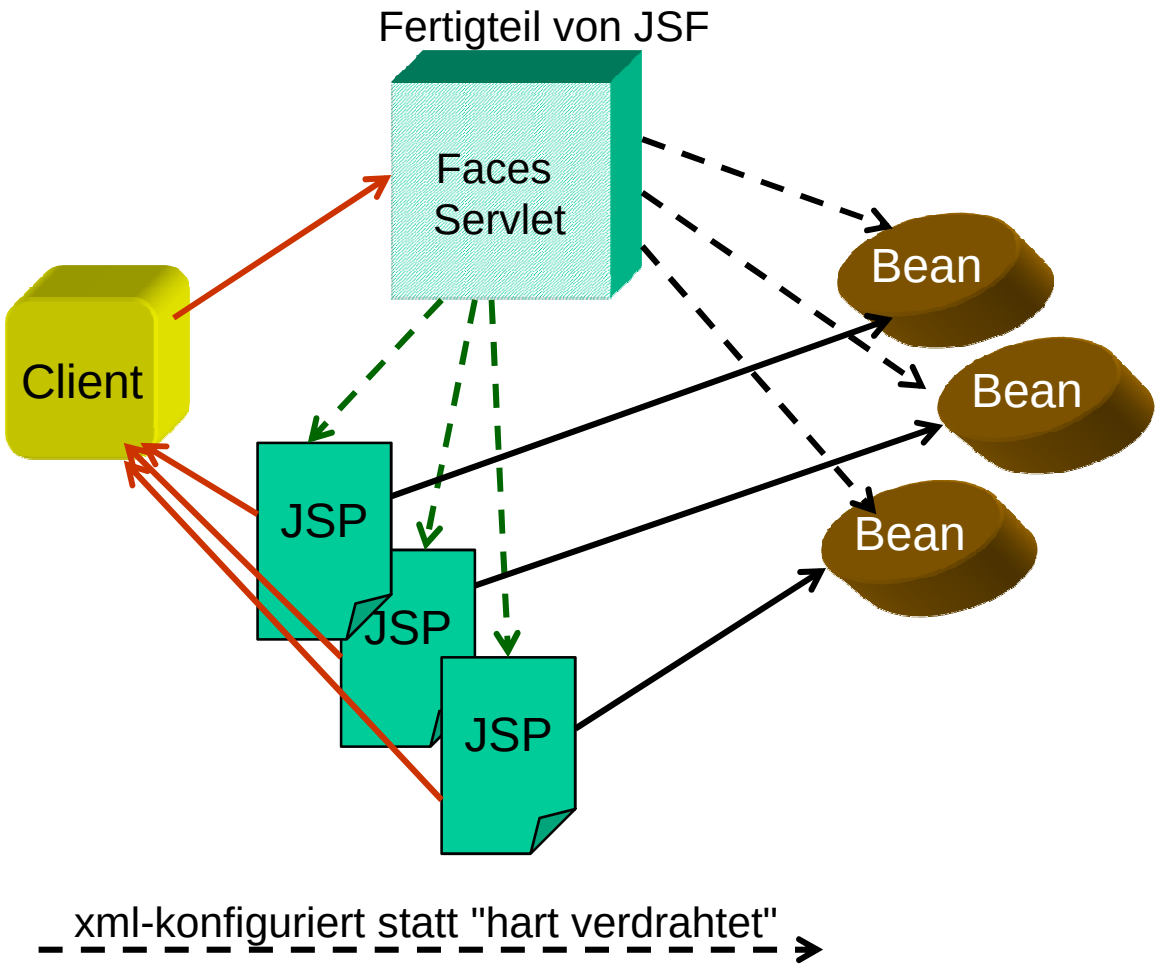
"Verdrahtung" - Konfiguration

- Deployment-Deskriptor web.xml
  - Spezifikation des FacesServlet in web.xml
  - url-mapping: Alle anfragen an das Wurzelverzeichnis werden an das FacesServlet weiter gegeben:

- JSF-Deskriptor faces-config.xml
  - Liste der Managed Beans
  - Liste der Navigationsregeln
  - gehört ins WEB-INF – Verzeichnis ("neben" web.xml)

- JSF-Bibliotheken einbinden
  - gehören ins WEB-INF – Verzeichnis,
  - Unterverzeichnis lib

Modell 2 mit JSF



- FacesServlet
  - empfängt alle Requests
  - füllt Beans
  - leitet an die passende Antwort-JSP weiter.
- FacesContext
  - instanziiert die Beans
- JSP
  - liest Beans
  - erzeugt und sendet Response

Web-Programmierung kann ja so einfach sein...

oder?