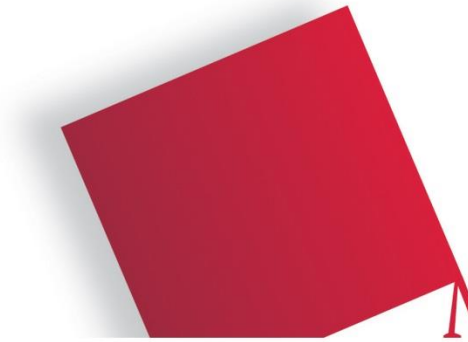


HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

Webbasierte Benutzerschnittstellen mit JSF

FWP Aktuelle Technologien zur Entwicklung verteilter Java-Anwendungen

Sommersemester 2014

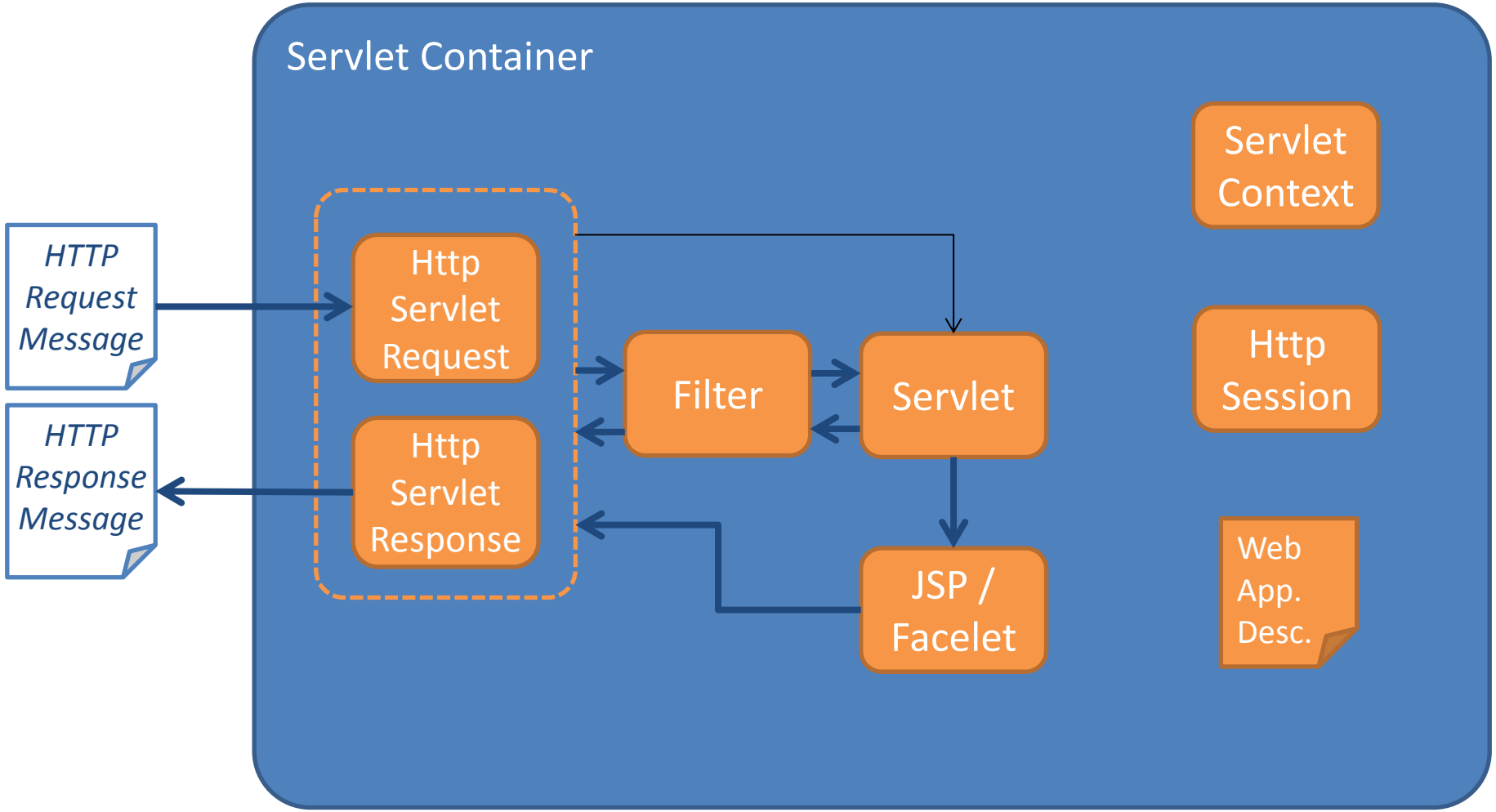
Michael Theis, Lehrbeauftragter

GRUNDBAUSTEINE WEB-BASIERTER BENUTZERSCHNITTSTELLEN

Web Applikationen mit Java

- Werden als *Web Application Archive (WAR)* verpackt
- Können eigenständig oder als Teil einer *Enterprise Application* installiert werden
- Ihre Komponenten werden über den *Web Application Descriptor (web.xml)* oder über *Annotations* konfiguriert
- *Kontextpfad* in der URL identifiziert eine Web-Applikation: *http://localhost:12345/**jeetrain**/...*

Servlet API auf einen Blick



Web Application Descriptor

- XML-Konfigurationsdatei /WEB-INF/web.xml
- Definiert alle Komponenten und Artefakte:
  - Servlet Kontext Parameter
  - Servlets und Filter und deren URLs
  - Servlet Context Listener
  - Security Constraints, Security Roles, Art der Anmeldung
- Kommt in zwei Ausprägungen:
  - hersteller-neutral: /WEB-INF/web.xml
  - hersteller-spezifisch: (z.B.) /WEB-INF/glassfish-web.xml

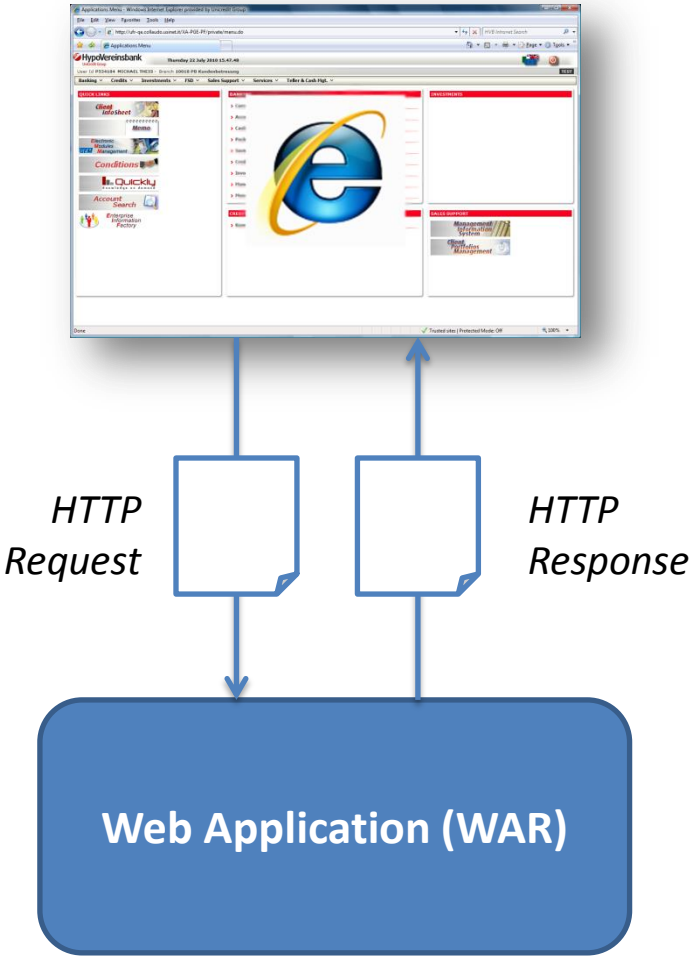
Servlet

- Endpunkte für HTTP-Nachrichten
  - Konsumieren HTTP-Requests
  - Produzieren HTTP-Responses
- Konfiguration über */WEB-INF/web.xml* oder *@WebServlet*
  - Servlet-Name, Servlet-Klasse, Servlet-URL-Mapping
- Eigene Servletklassen müssen entweder Interface *javax.servlet.Servlet* oder *javax.servlet.http.HttpServlet* implementieren

Filter

- Ermöglichen das Abfangen und Manipulieren von HTTP-Request- und HTTP-Response-Nachrichten
- Unterstützen beim Einbringen von Aspekten auf HTTP-Request- und HTTP-Response-Ebene
- Konfiguration über */WEB-INF/web.xml* oder *@WebFilter*
  - Filter-Name, Filter-Klasse, Filter-URL-Mapping
- Eigene Filterklassen müssen Interface *javax.servlet.Filter* implementieren

Request und Response



- Browser und Web-Applikation tauschen HTTP Nachrichten aus:
  - HTTP Requests lösen spezifische Aktionen innerhalb der Web-Applikation aus
  - HTTP Responses enthalten das Ergebnis der Aktion (im Allgemeinen als HTML-Dokument)
- Text-basierte HTTP-Nachrichten werden durch Objekte repräsentiert:
  - HTTPServletRequest
  - HTTPServletResponse
- *HTTPServletRequest*
  - Bestimmt Art des Zugriffs (GET/POST/…)
  - Definiert welche Aktion ausgelöst werden soll (URL)
  - Bietet Zugriff auf übergebene Parameter
- *HTTPServletResponse*
  - Bietet einen Ausgabestrom, in den (unter anderem) HTML Code geschrieben werden kann

Session

- Repräsentiert durch *javax.servlet.http.HttpSession*
- Identifizieren einen Benutzer über mehrere Requests hinweg
- Informationen zum Benutzer können als Sessionattribute in der Session gespeichert werden
- HTTP-Sessions sind gebunden an einen bestimmten Benutzer und eine bestimmte Web-Applikation

Lebenszyklus einer HTTP Session

- Neue Sessions werden erzeugt beim ersten Aufruf von *HttpServletRequest.getSession()*
- Servlet Container identifiziert existierende Sessions über ein Cookie identifiziert
- Session existiert
  - entweder bis zur expliziten Freigabe über *HttpSession.invalidate()*
  - oder bis zur impliziten Freigabe durch den Container nach Ablauf des HTTP-Session-Timeouts

Möglichkeiten der Datenhaltung

- Web-Applikationen müssen Daten über einen bestimmten Zeitraum halten
  - Zwischen aufeinanderfolgenden HTTP-Requests
  - Solange ein Benutzer mit der Web-Applikation arbeitet
- Es gibt drei Möglichkeiten zur Datenspeicherung:
  - Request-Attribute / request scope
  - Session-Attribute / session scope
  - Servlet-Context-Attribute / application scope
- Manche Frameworks bieten erweiterte Scopes

Gedanken zur Datenhaltung

- Sinnvolle Datenhaltung ist Voraussetzung für skalierbare Applikationen
  - Halten Sie Daten so kurz wie möglich vor
  - Denken Sie darüber nach, was Sie in Sessions speichern wollen und wie lange die Daten dort bleiben sollen
  - Benutzer verlassen Web-Applikationen u.U. unerwartet
- Nutzen Sie Frameworks mit Datenmanagement
- Seien Sie darauf vorbereitet, dass mehrere Browserfenster die gleiche Session nutzen

Java Server Pages (JSP)

- Bringen HTML und Java zusammen
  - HTML-Code und Java-Code lassen sich in einer Datei mischen
- Aus JSPs werden on-demand von einem JSP Compiler Java Sourcen generiert und kompiliert
- Generierte JSP-Klassen entsprechen einem Servlet

JSP Tags

- Erweitern den Sprachraum von HTML
- Werden bei jedem Anzeigen einer JSP durchlaufen
- Ermöglichen die Ausführung komplexer Präsentations-Logik
- Können dynamisch HTML-Code in eine Seite einfügen
- Werden zusammengefasst in JSP Tag Libraries
- Pro Tag ein Eintrag im Tag Library Deskriptor (TLD) und eine Tag Handler Klasse

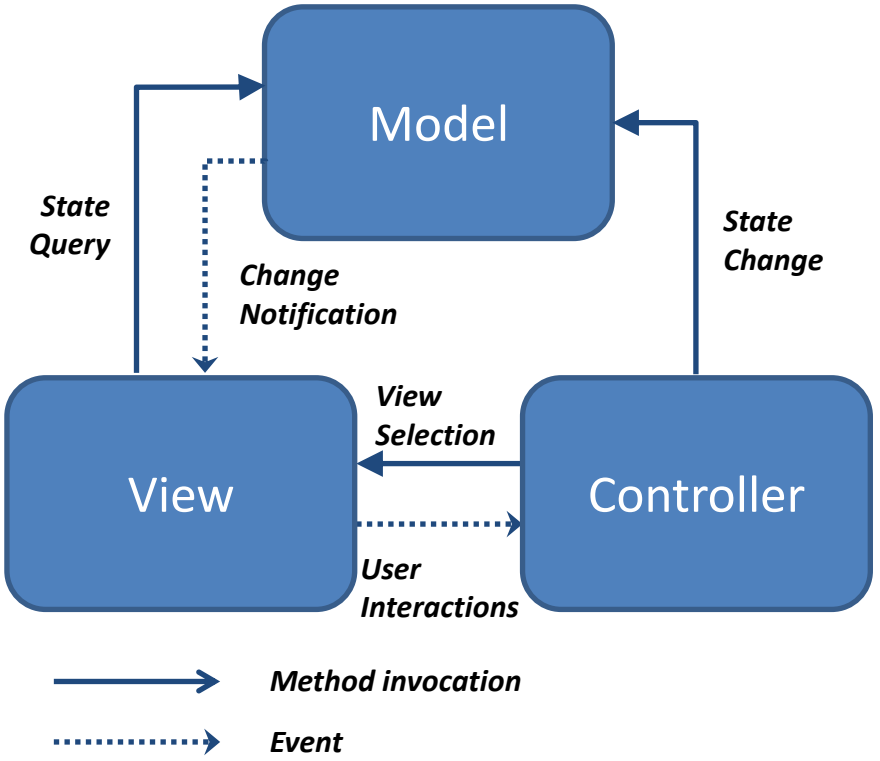
JSP Standard Tag Library (JSTL)

- Stellt immer wieder benötigte Basis-Funktionalität in Form einfacher Tags zur Verfügung:
  - Ausgabe von Werten aus Request und Session: *c:out*
  - Iteration über Arrays oder Collections: *c:forEach*
  - Bedingte Sprünge: *c:if* *c:choose*/*c:when*/*c:otherwise*
  - Internationalisierung: *fmt:message*, *fmt:format**
  - Manipulation von XML-Dokumenten: *x:**

BASISARCHITEKTUR WEB-BASIERTER APPLIKATIONEN

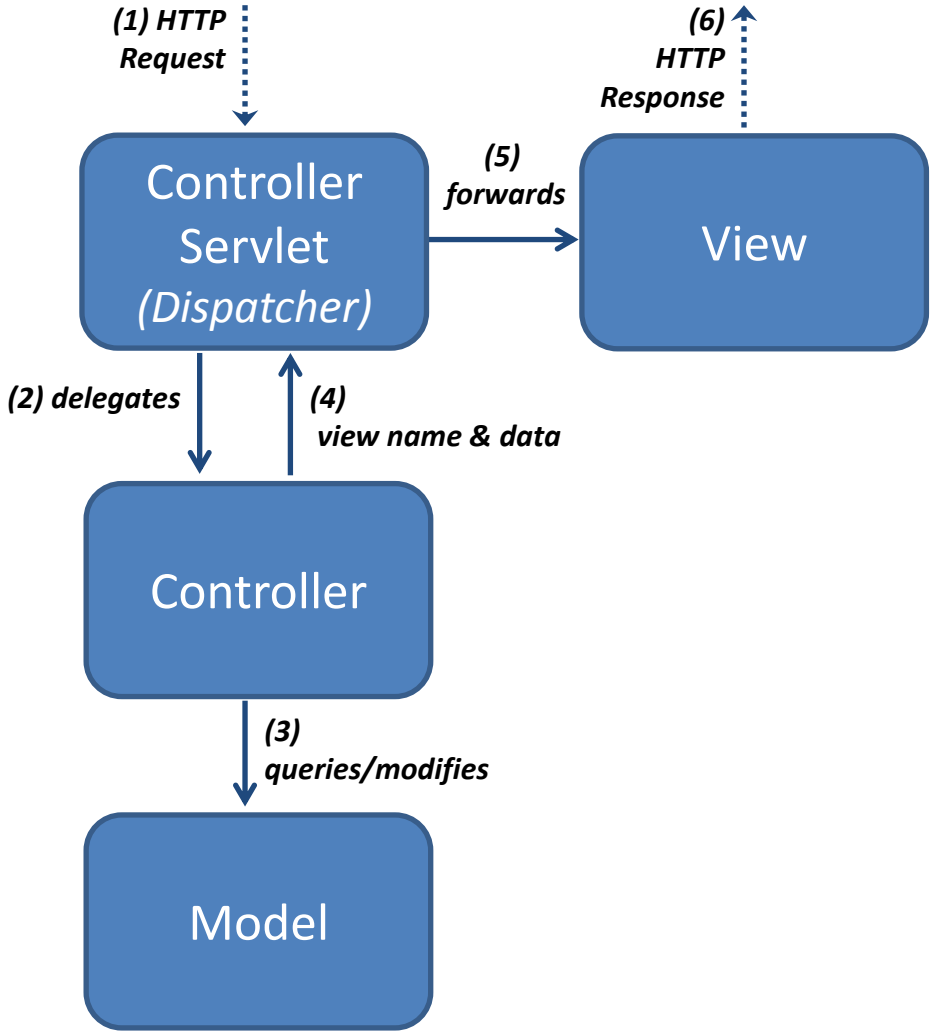
Model-View-Controller

- Regelt die Gliederung einer Web-Applikation in Artefakte mit wohl-definierten Rollen und Verantwortlichkeiten
- ***Model***: Repräsentiert die Daten inkl. der Regeln, nach denen auf diese Daten zugegriffen werden darf bzw. wie diese Daten verändert werden dürfen.
- ***View***: Stellt die im Model enthaltenen Daten für einen menschlichen Benutzer lesbar dar. Ermöglicht dem Benutzer Interaktion mit dem Model.
- ***Controller***: Übersetzt die Interaktionen des Benutzers mit dem View in Aktionen, die durch das Model ausgeführt werden sollen und dabei ggf. den Zustand des Models ändern.



Model 2 Architektur

- **Motivation**: Ereignisbasierte Benachrichtigungen funktionieren im Web nicht besonders gut
- Die Model 2 Architektur versucht dieses Problem mit einer ver-einfachten MVC-Adaption zu lösen:
  - Ein ***Controller Servlet*** leitet eingehende Requests basierend auf deren URL und Parametern an einen Controller weiter.
  - Der ***Controller*** übernimmt den gewünschten Zugriff auf das Model, fügt dem aktuellen Request die anzuzeigenden Daten hinzu und liefert den nächsten View zurück.
  - Das Controller Servlet leitet den Request auf den nächsten View um.
  - Der ***View*** stellt die anzuzeigenden Daten dar.



SECURITY IN WEB-APPLIKATIONEN

Security Constraints

- Stellen sicher, dass geschützte Bereiche nur von Benutzern mit den passenden Rechten betreten werden
- Deklarative Sicherheit über *web.xml* oder *@ServletSecurity*
- Nur authentisierte und autorisierte Benutzer können geschützte Bereiche betreten
- Nicht authentisierte Benutzer werden automatisch zur Anmeldung gezwungen

Security Roles

- Repräsentieren logische Zugriffsrechte einer Anwendung
- Müssen auf physische Rollen/Gruppen abgebildet werden („Admin“ entspricht „JTR00123“)
- Bestimmen in Verbindung mit Security Constraints die erwarteten Zugriffsrechte
- Können programmatisch über *HttpServletRequest.isUserInRole(String)* abgefragt werden

Basic Authentication

- Einfachster Weg der Authentisierung

```
<security-constraint>
  <display-name>ProtectedPages</display-name>
  <web-resource-collection>
    <web-resource-name>ProtectedPages</web-resource-name>
    <url-pattern>/ping.jsp</url-pattern>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint><role-name>Authenticated</role-name></auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<login-config><auth-method>BASIC</auth-method></login-config>
<security-role><role-name>Authenticated</role-name></security-role>
```

Was will ich schützen?

Erforderliche Rolle

Verschlüsselte Kommunikation

Basic authentication

- Browser übernimmt Abfrage der Logindaten über Popup

Form-based Login

- Anmeldung über ein selbst erstelltes Formular

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/login/login.jsp</form-login-page>
    <form-error-page>/login/loginFailed.jsp</form-error-page>
  </form-login-config>
</login-config>
```

- Anmeldung wird vom Servletcontainer über konfigurierte Login-Seite erzwungen
- Login-Seite muss eine vordefinierte Aktion auslösen und die Eingabefelder müssen bestimmte Namen haben

Aufbau eines Login-Formulars

- Um korrekt zu funktionieren, muss die Login-Seite den folgenden Namenskonventionen genügen:
  - Ausgelöste Aktion muss *j_security_check* heißen
  - Eingabefeld für den Benutzernamen muss *j_username* heißen
  - Eingabefeld für das Passwort muss *j_password* heißen

```
<form id="loginForm" name="loginForm" action="j_security_check"
method="post">
    <label id="userNameLabel" for="j_username"> User name:</label>
    <input id="j_username" type="text" name="j_username" size="16" />
    <label id="passwordLabel" for="j_password"> Password:</label>
    <input id="j_password" type="password" name="j_password" size="16" />
    <input id="loginButton" type="submit" name="loginButton" value="Login"/>
</form>
```

Programmatische Sicherheit

- Über *javax.servlet.HttpServletRequest*:
  - *getUserPrincipal()* liefert einen *java.security.Principal* zurück, der den angemeldeten Benutzer repräsentiert:

```
Principal principal = request.getUserPrincipal();
if (principal != null) {
    String authenticatedUserId = principal.getName();
}
```

  - *isUserInRole()* prüft, ob der angemeldete Benutzer eine bestimmte Rolle besitzt:

```
if (request.isUserInRole("Authenticated")) {
    // do something only accessible to authenticated users
} else {
    // throw exception indicating that authentication is required
    throw new IllegalStateException("Requires authentication!");
}
```

WEB-BASIERTE BENUTZERSCHNITT-STELLEN MIT JAVA SERVER FACES

Was ist Java Server Faces (JSF)?

- Standard Java-Framework zum Bauen von Benutzeroberflächen für Webapplikationen
- Design unterstützt einfache Entwicklung:
  - Komponentenorientierter Entwicklungsansatz unabhängig von konkreten Endgeräten
  - Vereinfachtes Management von Applikationsdaten
  - Automatische Verwaltung von Zuständen
  - Bietet das Beste aus der Welt der Webapplikations-entwicklung über eine einheitliche, verständliche API

Design-Ziele von JSF (I)

- Standard-UI-Komponenten-Framework, von Entwicklungstools wirksam einsetzbar
- Einfache, leicht-gewichtige Basisklassen für UI-Komponenten, Komponentenzustände und Eingabeereignisse
- Gemeinsamen UI-Komponenten, die als Basis für neue Komponenten verwendet werden können

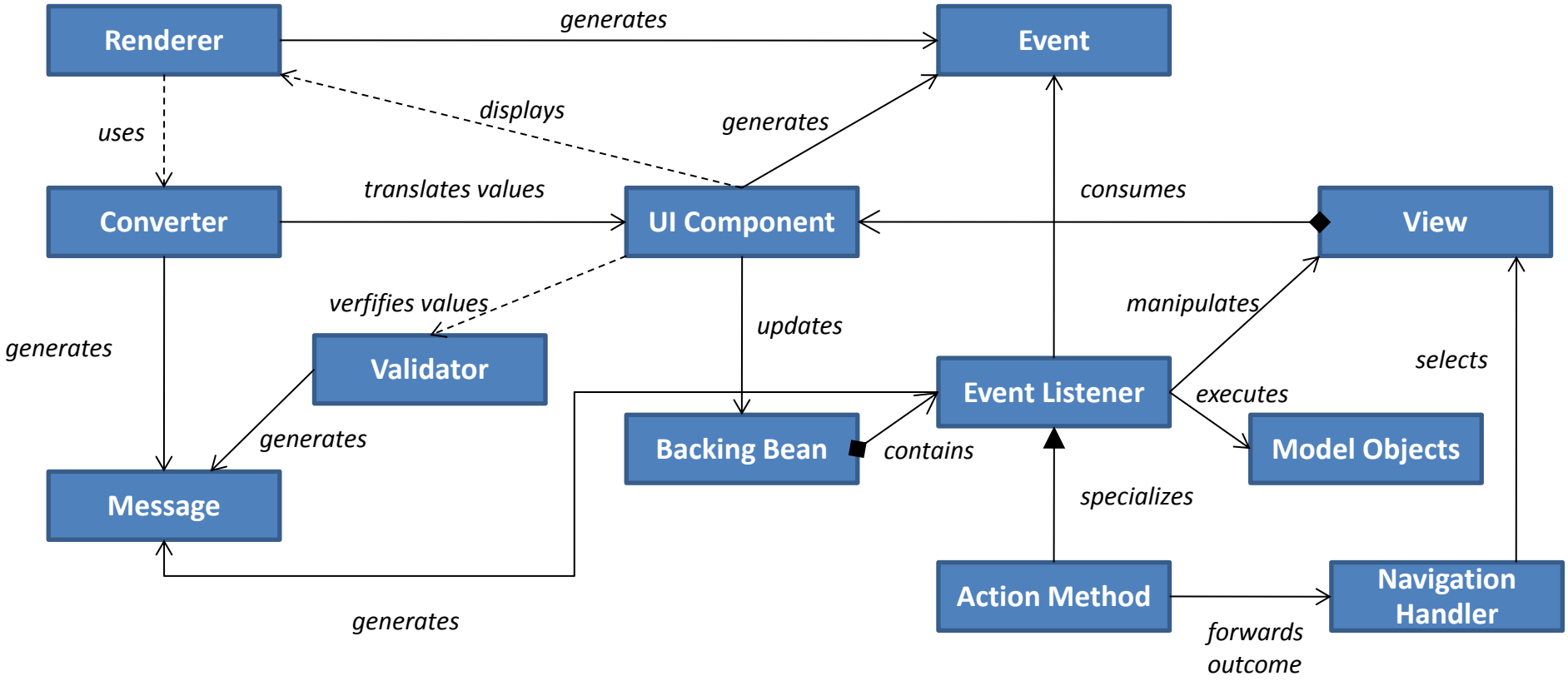
Design-Ziele von JSF (II)

- APIs für die Eingabevalidierung inlusive der Unterstützung für clientseitige Validierung
- Model für die Internationalisierung und Lokalisierung der Benutzerschnittstelle
- Automatische Generierung der zum Endgerät passenden Ausgabe
- Automatische Generierung der Ausgabe bietet Erweiterungspunkte für barrierefreien Zugriff

Überarbeitungen des Designs

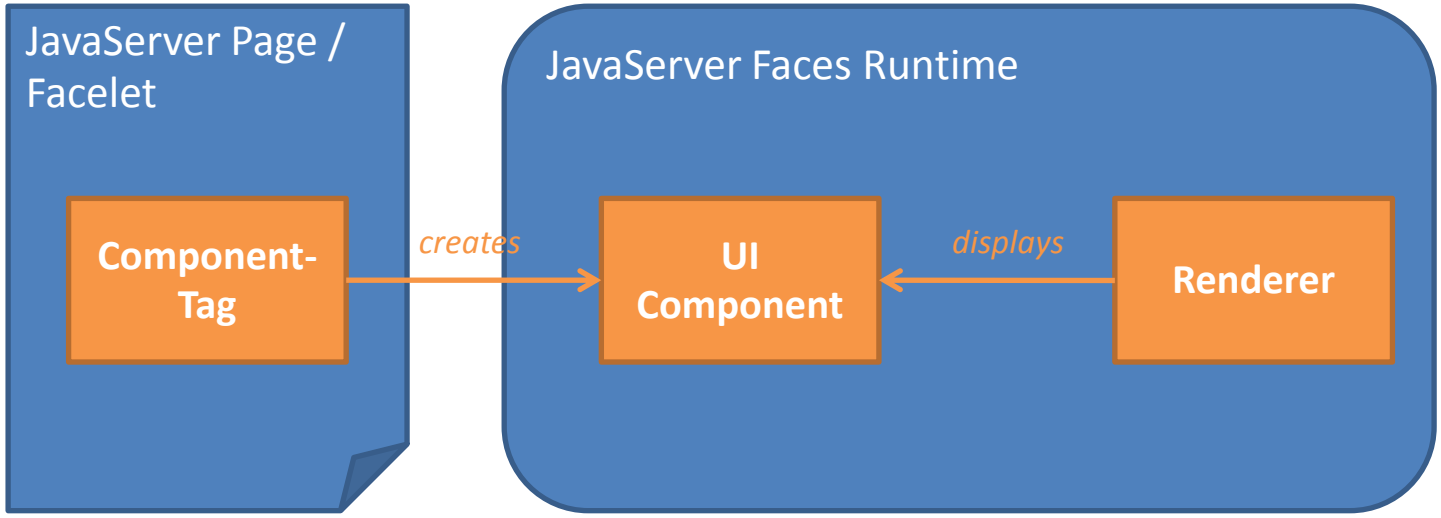
- JSF 1.2
  - Gemeinsame Unified Expression Language für JSP/JSF
- JSF 2.0
  - Facelets als neue Standard-View-Technologie
  - Erstklassige Unterstützung von AJAX
  - Composite Components
  - Verbesserte Validierung durch Bean Validation (JSR303)
  - Erstklassige Unterstützung von Ressourcen
  - Annotations statt XML + Convention over Configuration

JSF basiert auf Komponenten



Quelle: JavaServer Faces in Action, Kito Mann

Dreifaltigkeit als Grundprinzip

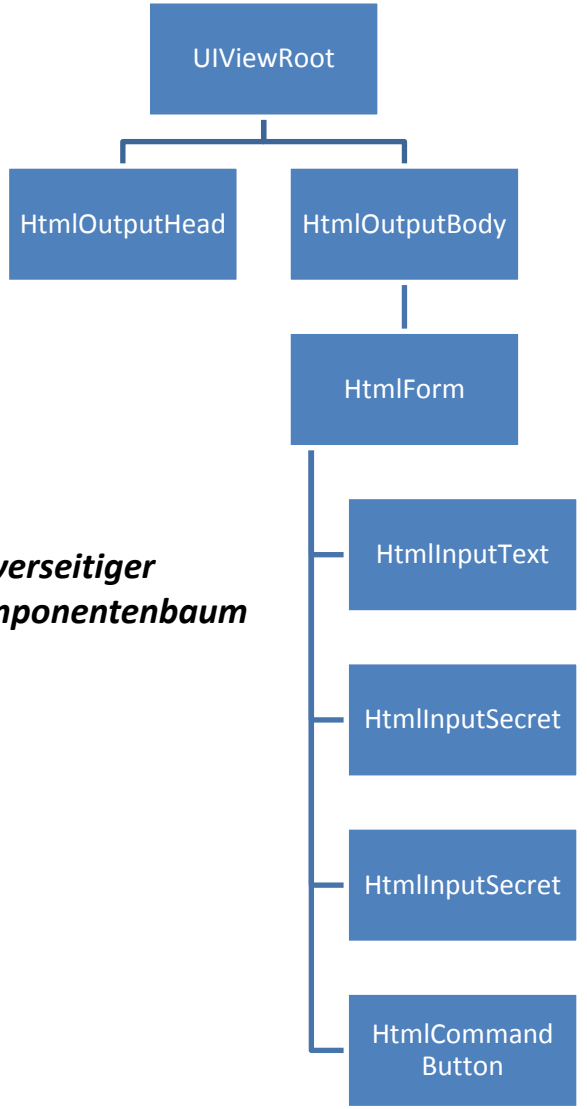


Serverseitiger Komponentenbaum

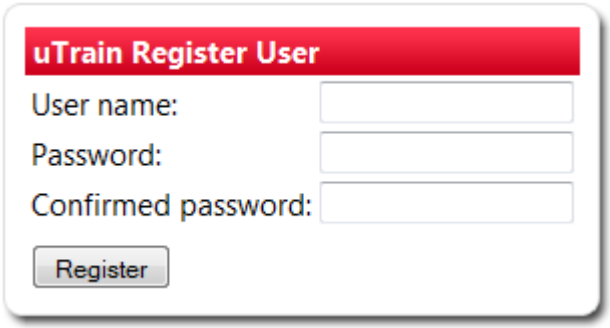
```
<h:form id="registerUserForm">
  <h:inputText
     id="userName" ... />
  <h:inputSecret
     id="password" ... />
  <h:inputSecret
     id="confirmedPassword" .../>
  <h:commandButton
     id="registerButton" ... />
</h:form>
```

Facelets or JSP view mit Komponententags

(1) Erstellen des Komponentenbaumes



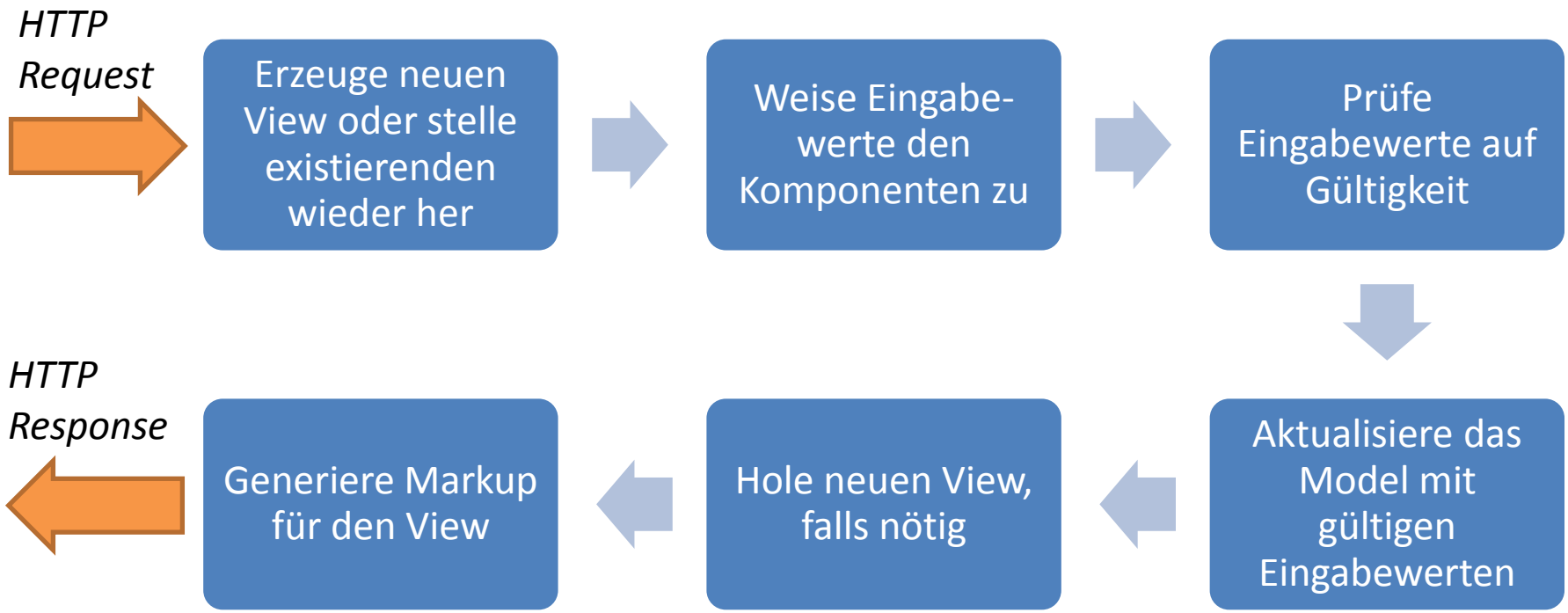
Serverseitiger Komponentenbaum



Im Browser angezeigter View

(2) Generieren des HTML Codes

Phasen der Requestverarbeitung



Faces Servlet

- Verarbeitet alle an JSF gerichteten HTTP-Requests
- Folgende URL-Muster identifizieren an JSF gerichtete HTTP-Requests:
  - */faces/** oder **.jsf*
- Initialisiert beim Start der Web-Applikation die JSF-Laufzeitumgebung
  - Unter Berücksichtigung der JSF-spezifischen Kontext-Parameter
  - Unter Berücksichtigung der JSF-Konfigurationsdatei */WEB-INF/faces-config.xml*

Faces Context

- Abstrakte Klasse, über welche mit der JSF-Laufzeitumgebung kommuniziert werden kann:
  - Zugriff auf Request/Response über *ExternalContext*
  - Zugriff auf all konfigurierten JSF-Artefakte
  - Zugriff auf den Komponentenbaum (*UIViewRoot*)
  - Direktes Schreiben der HTTP Response-Nachricht
  - Error-Handling (*FacesMessage*)
- Instanzen werden über die statische Factory-methode *FacesContext.getCurrentInstance()* ermittelt

Managed Beans

- Einfache POJO-Klasse + *@Named*/*@ManagedBean*
- Kapseln Präsentationslogik:
  - Validiere alle Benutzereingaben
  - Bilde Benutzeraktionen auf Aufrufe von Geschäfts-komponenten ab
  - Ermittle und stelle Geschäftsdaten zur Visualisierung durch Views bereit
  - Halte Daten über mehrere Requests hinweg
  - Entscheide welcher View als nächster anzuzeigen ist
  - Behandle alle Fehler, die dazwischen aufgetreten sind

Lebensdauer eines Managed Beans

- Managed Bean Instanzen werden beim ersten Zugriff erzeugt und dann wiederverwendet
- Gültigkeitsbereich bestimmt deren Lebensdauer:
  - *RequestScoped*: für die Dauer des aktuellen Requests
  - *SessionScoped*: für die Dauer der aktuellen Session
  - *ApplicationScoped*: solange die Applikation läuft
  - *ViewScoped*: solange der aktuelle View angezeigt wird (nur JSF)
  - *ConversationScoped*: für die Dauer der aktuellen Konversation (nur CDI)

Managed Beans und Views

- Kommunikation zwischen Managed Beans und Views erfolgt über Ausdrücke in den Views
- Als Sprache für die Ausdrücke dient die *Unified Expression Language*
- Über *Value Expressions* lassen sich Daten aus Managed Beans lesen und schreiben
- Über *Method Expressions* lassen sich Methoden aus den Managed Beans aufrufen

Unified Expression Language

- Erlaubt Autoren, von Web-Seiten über einfache Ausdrücke auf Daten in Java-Objekten zuzugreifen
- Allgemeine Syntax: *#{<Ausdruck>}*
- Einfache Ausdrücke können Felder oder Methoden in Java-Objekten/Java-Objekt-Bäumen referenzieren: *#{userEditor.user.name}*
- Komplexe Ausdrücke reichen von Abfragen von logischen Bedingungen bis zu Aufrufen von Methoden mit beliebiger Signatur

Value Expressions

- Zustandsbehaftete JSF Komponenten besitzen ein Attribut *value*, mit dem Daten in Managed Beans referenziert werden können
  - Referenzierung erfolgt über eine Value Expression
  - Werte werden beim Rendern eines Views aus dem referenzierten Feld gelesen
  - Werte werden beim Submit eines Formulars in referenzierte Feld geschrieben

```
<h:inputText id="userName„
    value="#{userEditor.user.userName}" ... />
```

Method Expressions (Actions)

- Kommandobasierte JSF Komponenten besitzen ein Attribut *action*, mit dem Methoden in Managed Beans referenziert werden können
  - Referenzierung erfolgt über eine Method Expression
  - Action-Methoden sind parameterlos und liefern einen String-Wert zurück, der den nächsten View bestimmt
  - Aufruf der Methode erfolgt beim Submit eines Formulares

```
<h:commandButton id="registerButton“
   action="#{userEditor.registerUser}"…/>
```

Method Expressions (Events)

- Ereignisbasierte JSF Komponenten besitzen Attribute, mit dem Listener-Methoden in Managed Beans referenziert werden können
  - Referenzierung erfolgt über eine Method Expression
  - Art des Ereignisses bestimmt die Signatur der Methode
  - Aufruf der Methode erfolgt beim (ggf. partiellen, über AJAX ausgelösten) Submit eines Formulares

Facelets als neue Viewtechnologie

„Facelets is a powerful but lightweight page declaration language that is used to build JavaServer Faces view using XHTML style templates to build component trees“
The JavaEE 6 Tutorial, 2. Ausgabe

- Server-seitiges Templating welches die Komposition von Views aus wiederverwendbaren Teilen ermöglicht
- Entwickelt unter der vollständigen Berücksichtigung von JSF
- Views werden in Standard-XHTML geschrieben
- Bieten die Möglichkeit zur einfachen Erstellung eigener Tag-Libraries
- Bestehende JSF and JSTL Tag-Libraries können weiterhin genutzt werden
- Unterstützen die Unified Expression Language
- Erzwingen die strikte Trennung nach MVC-Paradigma, da Java-Code nicht mehr in Views verwendet werden kann

Templating

- Ermöglicht die Modularisierung von Facelets in wiederverwendbare Teile basierend auf objekt-orientierten Konzepten:
  - *Komposition*: Existierende Facelets können zu neuen Facelets zusammengesetzt werden
  - *Vererbung*: Template-Clients können existierende Templates erweitern
  - *Überschreiben*: Template-Clients können Teile von Templates überschreiben

Templates

- Facelet-Templates sind einfache Facelet-Dokumente
  - Definieren überschreibbare Templateteile (Platzhalter) mit *ui:insert*
  - Inkludieren andere Facelet-Dokumente als Templateteile mit *ui:include*
- Dienen als Vorlagen für die Template-Clients
  - Änderungen am Template wirken sich auf alle Clients aus

Template Clients

- Template Clients sind einfache Facelet-Dokumente
  - Deklarieren über *ui:composition*, dass sie auf einem Template beruhen
  - Geben über das Attribut *template* an, auf welches Template sie sich beziehen
  - Überschreiben mit *ui:define* gezielt gleichnamige Platzhalter aus dem Template

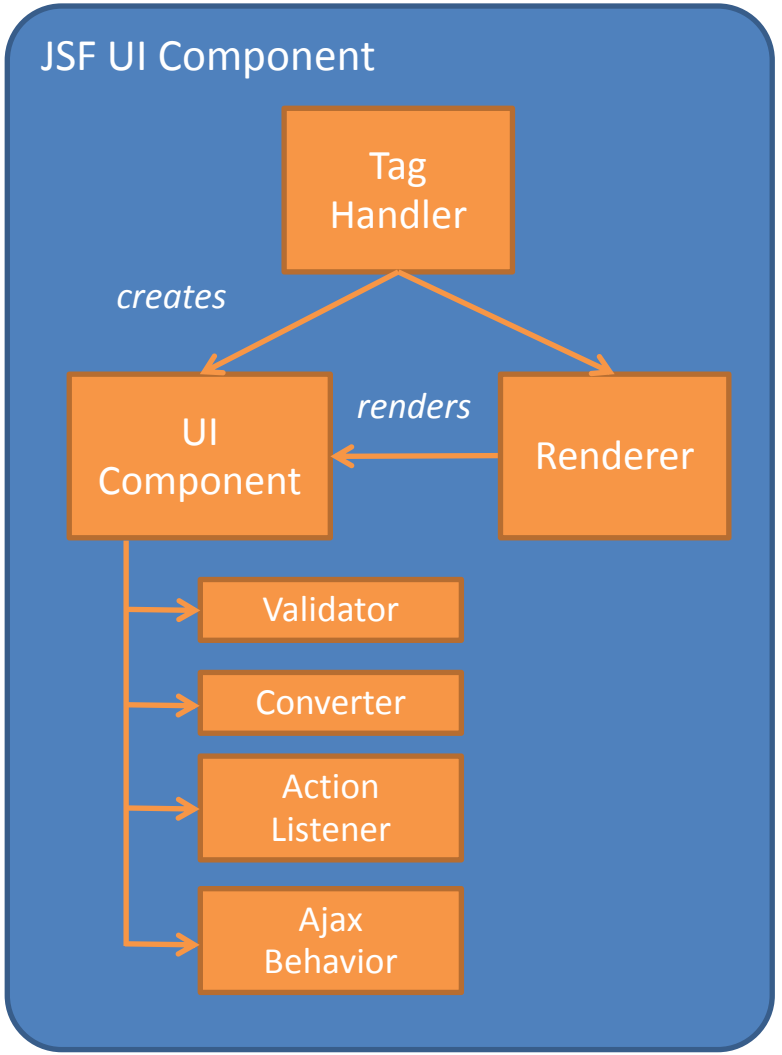
JSF UI Komponenten

A **user interface component** *(UI component) is a specific type of component that displays user interface content which the user can modify over time. This content ranges from simple input fields or buttons to more complex items, such as trees or datagrids.*
(Source: "JavaServer Faces 2.0 The Complete Reference", Ed Burns & Chris Schalk, McGraw-Hill 2010, p135)

- JSF kommt mit einem erweiterbaren Komponentenmodell für Benutzerschnittstellen
- Basierend auf diesem Komponentenmodell bietet JSF mehr als 30 vorgefertigte UI-Komponenten
- Diese Standard-HTML-UI-Komponenten sind gebunden an den Namensraum *http://java.sun.com/jsf/html* und den Präfix *h*

Bausteine einer UI Komponente

- ***UIComponent***: Klasse, welche die abstrakte Semantik der Komponente unabhängig von der Darstellung auf einem spezifischen Endnutzer-Gerät repräsentiert.
- ***Renderer***: Eine optional Klasse, die den Markup-Code einer Komponente für ein spezifisches Endnutzer-Gerät erzeugt.
- ***Tag Handler***: Klasse, die es erlaubt, eine bestimmte Komponente in eine Web-Seite zu bringen.
- ***Attached Objects***: Eine optionale Sammlung von Hilfskomponenten wie *Konvertern*, *Validatoren*, *Action-Listenern* etc, welche zusätzliche Funktionalität zur Verfügung stellen.



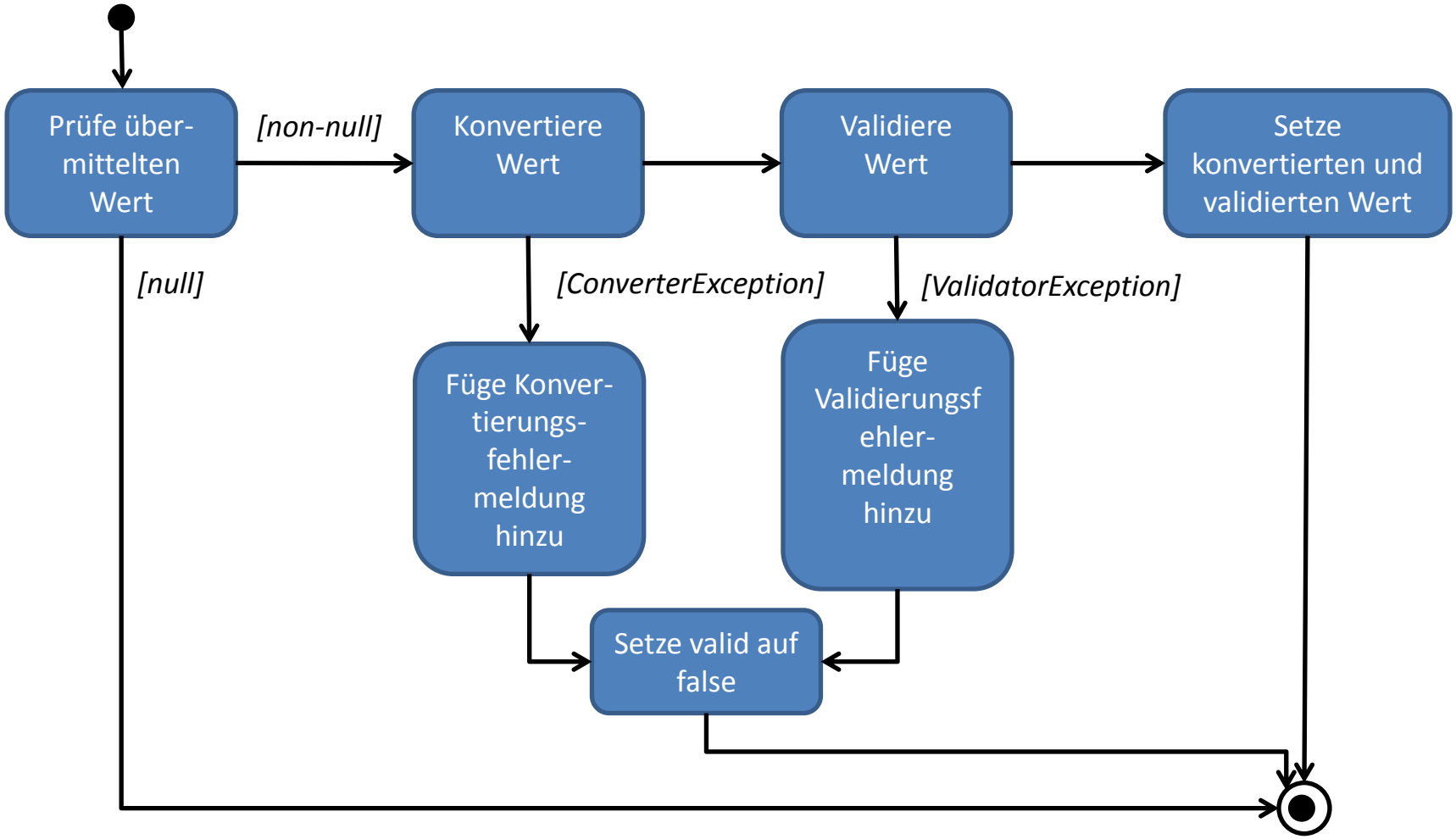
Gemeinsame Attribute (Auszug)

Attribut	Beschreibung
id	Eindeutiger Bezeichner der Komponente
value	Wert der Komponente; entweder Literal oder EL-Ausdruck
immediate	Steuert, ob ein Komponentenwert konvertiert und valdiert werden muss, bevor die Aktion auf der Komponente ausgeführt werden kann
rendered	Steuert, ob für Komponente Markup-Code erzeugt werden soll
required	Markiert eine Eingabekomponente als Pflichtfeld
requiredMessage	Optionale spezifische Fehlermeldung bei fehlendem Pflichtfeld
validatorMessage	Optionale spezifische Fehlermeldung bei fehlgeschlagener Validierung

Konverter und Validatoren

- HTML ist eine text-basierte Sprache wogegen Java eine Vielzahl von Datentypen unterstützt
  - Konvertierung von String-Werten in den gewünschten Java-Typ und umgekehrt notwendig
- Datenkonvertierung erfolgt in *Konvertern*
- Datenvalidierung erfolgt in *Validatoren*
- Konverter und Validatoren werden an UI-Komponenten gehängt
  - Implizit durch JSF oder explizit über Tags

Konvertierung und Validierung



Implizite und explizite Konvertierung

- JSF erkennt automatisch die erforderlichen Konvertierungen und hängt die entsprechenden Konverter an die UI-Komponenten
- Leider funktioniert dieser Automatismus in mindestens zwei Fällen nicht (explizite Angabe von Konvertern erforderlich)
  - Datum/Uhrzeit basierend auf *java.util.Date*
  - Beträge basierend auf *java.math.BigDecimal* oder *double*

Validierung mit Bean Validation

- *JSR 303 Bean Validation* definiert ein einfaches Validierungsmodell
  - Kann auf beliebige POJOs angewendet werden
  - Einsatz nicht auf die Präsentationsschicht beschränkt
- Validierungsregeln werden durch Annotationen an Feldern definiert (@NotNull => Pflichtfeld)
- Eigene Annotationen und Validierungshandler sind leicht zu implementieren
- JSF unterstützt Bean Validation automatisch

Bean Validation Annotationen

Constraint	Description
@AssertFalse, @AssertTrue	Spezifiziert gültige Werte für boolean Felder
@DecimalMax, @DecimalMin	Spezifiziert gültigen Wertebereich für Dezimalfelder
@Digits	Spezifiziert gültige Anzahl von Stellen in BigDecimal-Feldern
@Max, @Min	Spezifiziert gültigen Wertebereich für int-Felder
@NotNull	Der Wert des Feldes darf nicht null sein (= Pflichtfeld)
@Null	Der Wert des Feldes muss null sein
@Future, @Past	Spezifiziert den gütligen Wertebereich für Date und Calendar-Felder
@Pattern	Spezifiert einen regulären Ausdruck, zu dem ein String-Wert passen muss
@Size	Spezifiziert eine zulässige Größe für einen String, eine Collection oder eine Map

Fehlerbehandlung

- Im Kontext von JSF kann die Behandlung von Fehlern in zwei Bereiche unterteilt werden
  - Hinzufügen von Fehlermeldungen zum FacesContext (explizit in Managed Beans oder implizit in Konvertern und Validatoren)
  - Anzeigen von Fehlermeldungen in Views

Erzeugen von Meldungen

- Klasse *FacesMessage* repräsentiert Meldungen mit den folgenden Attributen
  - severity (FATAL, ERROR, WARNING, INFO)
  - summary
  - detail
- FacesMessage-Objekte können über *FacesContext.addMessage()* zur Anzeige vorgemerkt werden
  - Mit optionaler Angabe der betroffenen Komponente

Anzeigen von Meldungen

- Mit *h:message* kann eine einzelne Meldung zu einer bestimmten Komponente direkt neben der Komponente ausgeben werden
- Mit *h:messages* können alle Fehlermeldungen an einer Stelle im View ausgegeben werden

Navigation

- Navigation funktioniert in JSF im Allgemeinen so:
  - Konvertierungsfehler oder Validierungsfehler => Anzeige des letzten Views
  - Eine Aktionsmethode liefert kein Ergebnis => Anzeige des letzten Views
  - Eine Aktionsmethode liefert ein Ergebnis => der NavigationHandler versucht das Ergebnis auf die URL eines Views abzubilden
    - View lässt sich ermitteln => Anzeige des neuen Views
    - View lässt sich nicht ermitteln => Anzeige des letzten Views

Navigationsarten

- *Implizite Navigation* (seit JSF 2.0): hard-codierte, logische Seitennamen in Views oder als Ergebnis von Aktionsmethoden
- *Regel-basierte Navigation*: Navigationsregeln in faces-config.xml

Fragen?

ANHANG

Quellen

- Eric Jendrock et. al.
  The Java EE 7 Tutorial Part III Kapitel 7-16
  http://docs.oracle.com/javaee/7/tutorial/doc/jsf-intro.htm
  Oracle September 2013
- Ed Burns, Chris Schalk
  JavaServer Faces 2.0: The Complete Reference
  McGraw-Hill 2010
  ISBN 978-0-07-162509-8

Kontakt

Michael Theis
Lehrbeauftragter Hochschule München

email michael.theis@hm.edu
mobile + 49 170 5403805
web http://www.tschutschu.de/Lehrauftrag.html