HOCHSCHULE DER MEDIEN

Design Patterns | MVC

Marcus Köhler
Markus Merath
Axel Reusch

Agenda

- Allgemeines
- Aufgabenverteilung
- Alltagsbeispiel
- Beurteilung
- Konkretes Code-Beispiel
- Einsatz im Web
- Webbeispiel
- Fazit & Ausblick

Marcus Köhler

Axel Reusch

Markus Merath

Allgemeines

Das Problem

- In Anwendungen waren die Anwendungsdaten mit dem GUI-Code verwoben
  - Das **Design**, die **Daten** und die **Logik** wurden in einer zentralen Datei unter gebracht

Die Folgen des Problems

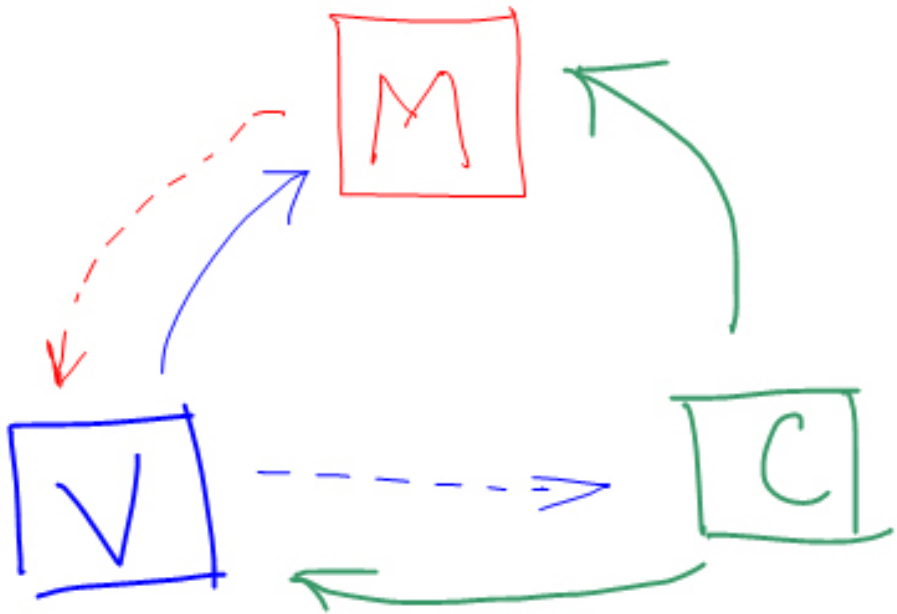
- Unübersichtlichkeit
- Schwere Wartbarkeit
- Schlechte Erweiterungsmöglichkeit
- Schlechte Widerverwendungsmöglichkeit des Codes

Was wollen wir ?

- Ein Muster durch das bei der Implementierung eine klare Trennung von **Datenhaltung**, **Design** und **Logik** entsteht.

- **Lösung der Probleme wie**
  - Unübersichtlichkeit
  - Schwere Wartbarkeit
  - Schlechte Erweiterung
  - Schlechte Struktur des Programms
  - Schlechte Arbeitsteilung

Wir stellen vor:

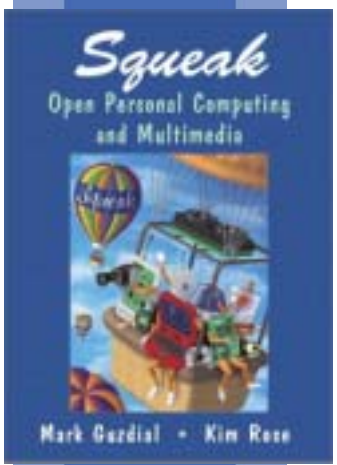


- **Das Model-View-Controller Pattern**

Erste Definitionen

- **Definition 1:**

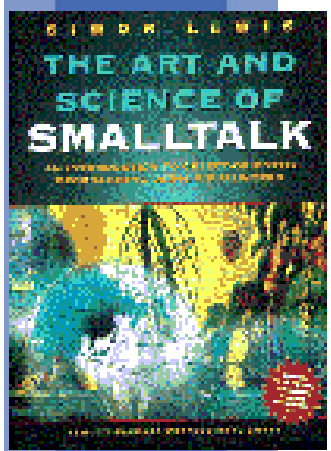
- (nach *Mark Guzdial* in seinem Buch „**Squeak**“): Das MVC Modell beschreibt im Grunde genommen eine Benutzerschnittstelle in Form von einem „**Model**“ der realen Welt, welches durch ein „**View**“ präsentiert wird, wobei Benutzereingaben durch ein oder mehrere „**Controller**“ gehandhabt werden



Erste Definitionen

- **Definition 2:**

- (nach *Simon Lewis* in seinem Buch „**The Art and Science of Smalltalk**“): Das MVC-Modell ist ein grundlegender Architekturbaustein von Smalltalk, in welchem die Funktionalität einer Applikation mit einer graphischen Benutzerschnittstelle (GUI) in die drei Bereiche „**Model**“, „**View**“ und „**Controller**“ gegliedert wird



Die Geschichte

- MVC wurde etwa 1978 im Xerox PARC (Palo Alto Research Center) im Zusammenhang mit Smalltalk-80 entwickelt

- Die Entwicklung von MVC wird **Trygve Reenskaug** (Norwegen) zugerechnet

Die Geschichte

- Erste Veröffentlichung über MVC:

„A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80“

- Von **Glenn Krasner** und **Stephan Pope** im *Journal of Object-Oriented Programming* (JOOP)
- im August/September 1988

Die Geschichte

- Der Siegeszug begann in Smalltalk
- Übertragung in Java/Swing
- Verwendung des Musters heute ebenfalls in Webanwendungen

- Andere Namen des Patterns
  - PAC – **Presentation-Abstraction-Control**
  - ICM – **Interface-Control-Model**

Beschreibung von MVC (klassisch)

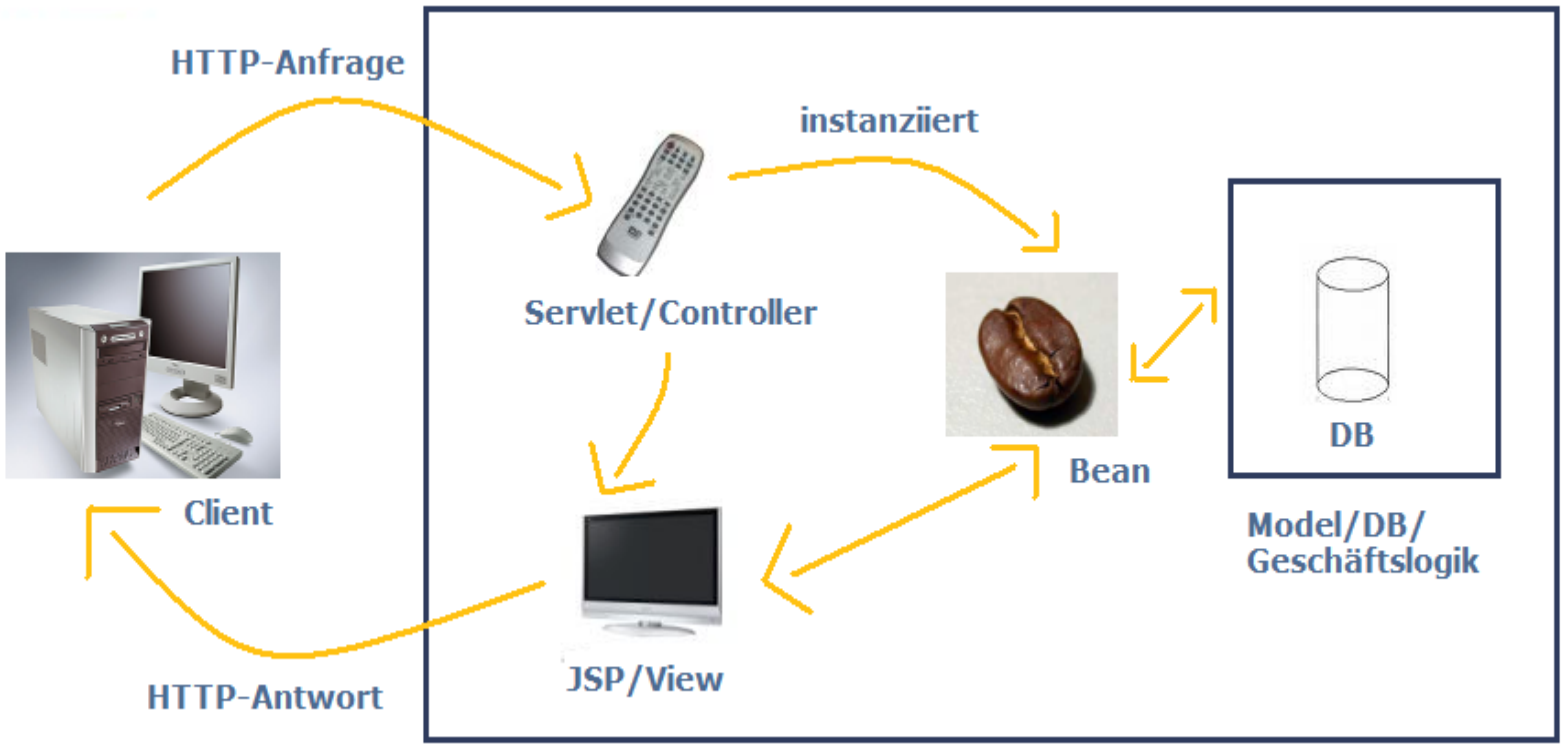
- Ziel: die Präsentation eines Programms von seiner Logik und den Daten zu trennen
- Das **Model** entspricht der Verarbeitung, die **View** der Ausgabe, sowie der **Controller** der Eingabe
- Model enthält die Kernfunktionalität und die Daten der Anwendung.

Beschreibung von MVC (klassisch)

- View stellt Informationen für den Benutzer graphisch dar
- Controller verarbeitet die Benutzereingabe
- Jeder dieser Komponenten sollte als eigener Baustein implementiert werden

MVC und das Web

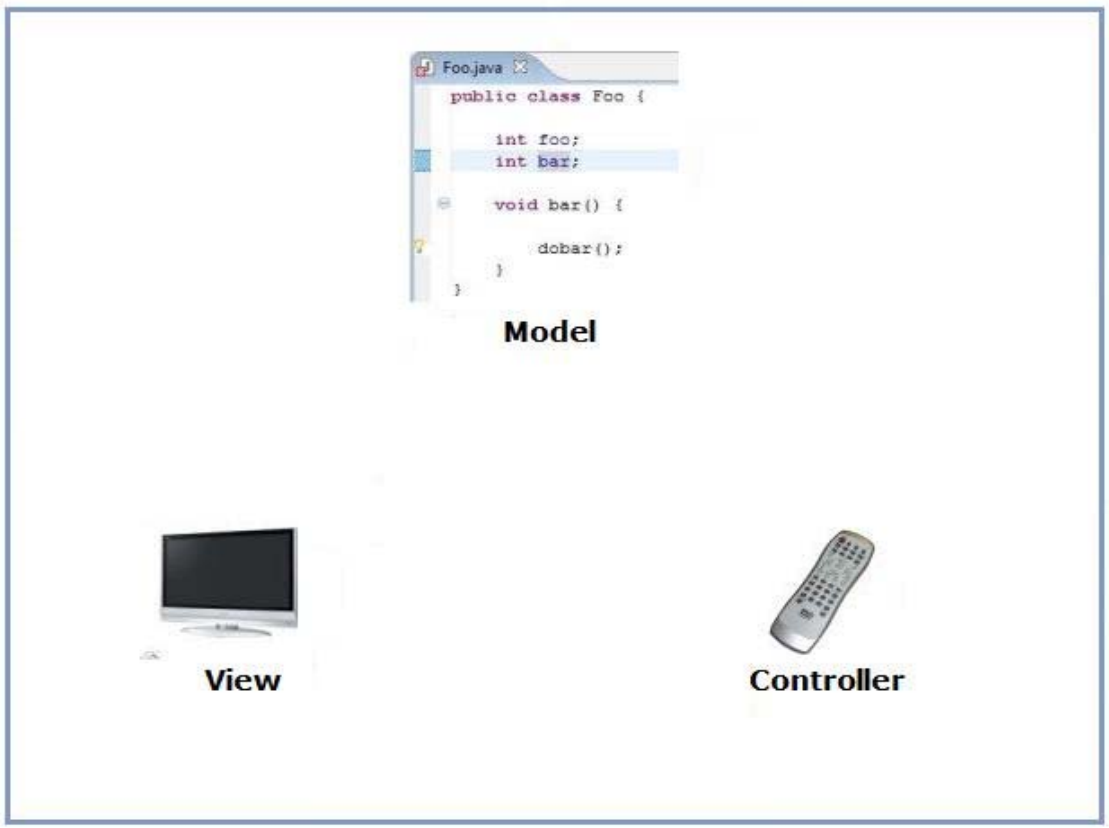
- Beispiel einer Anfrage



Aufgabenverteilung

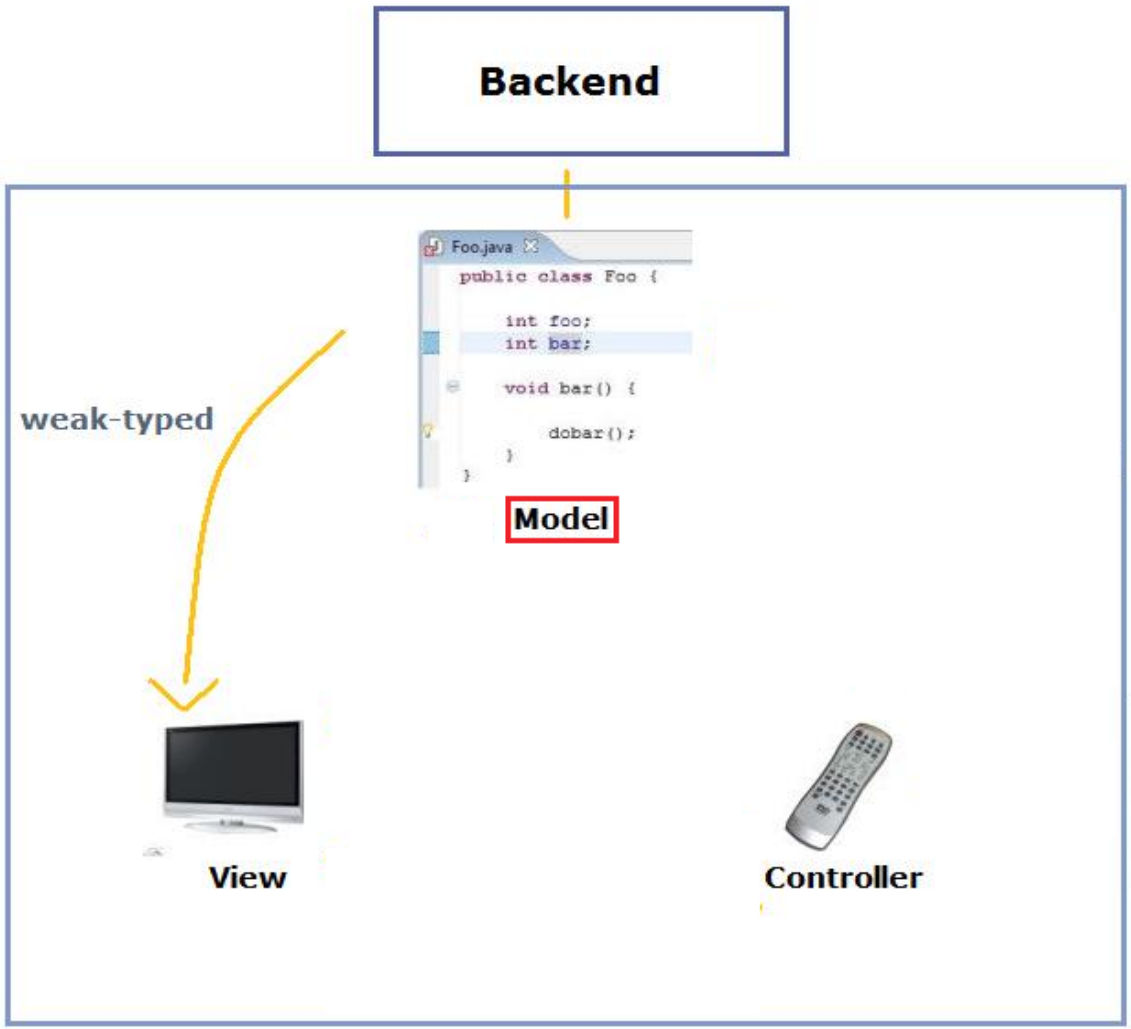
Aufgabenverteilung

Die Komponenten im Überblick



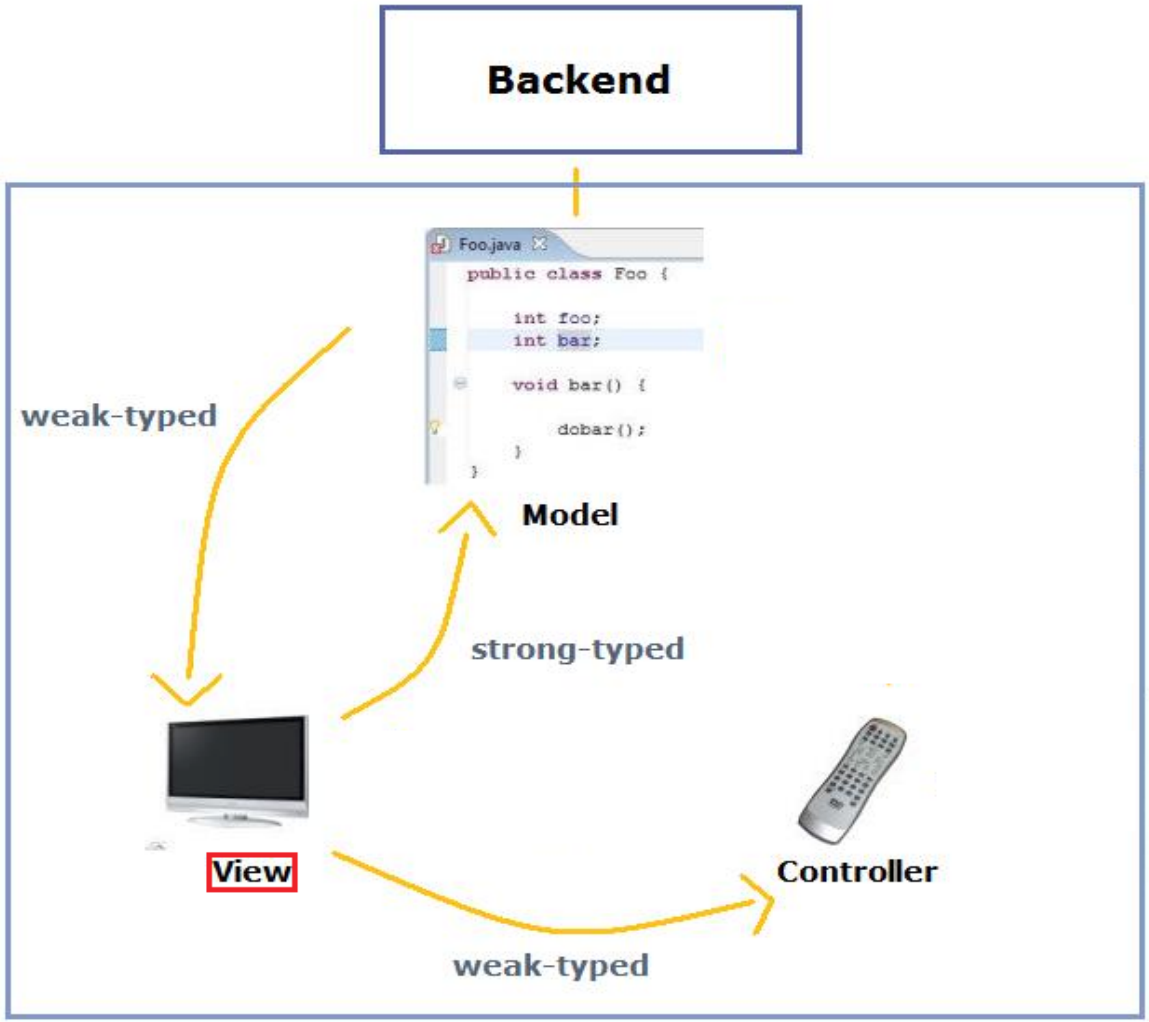
Aufgabenverteilung

Die Komponenten im Überblick: **Model**



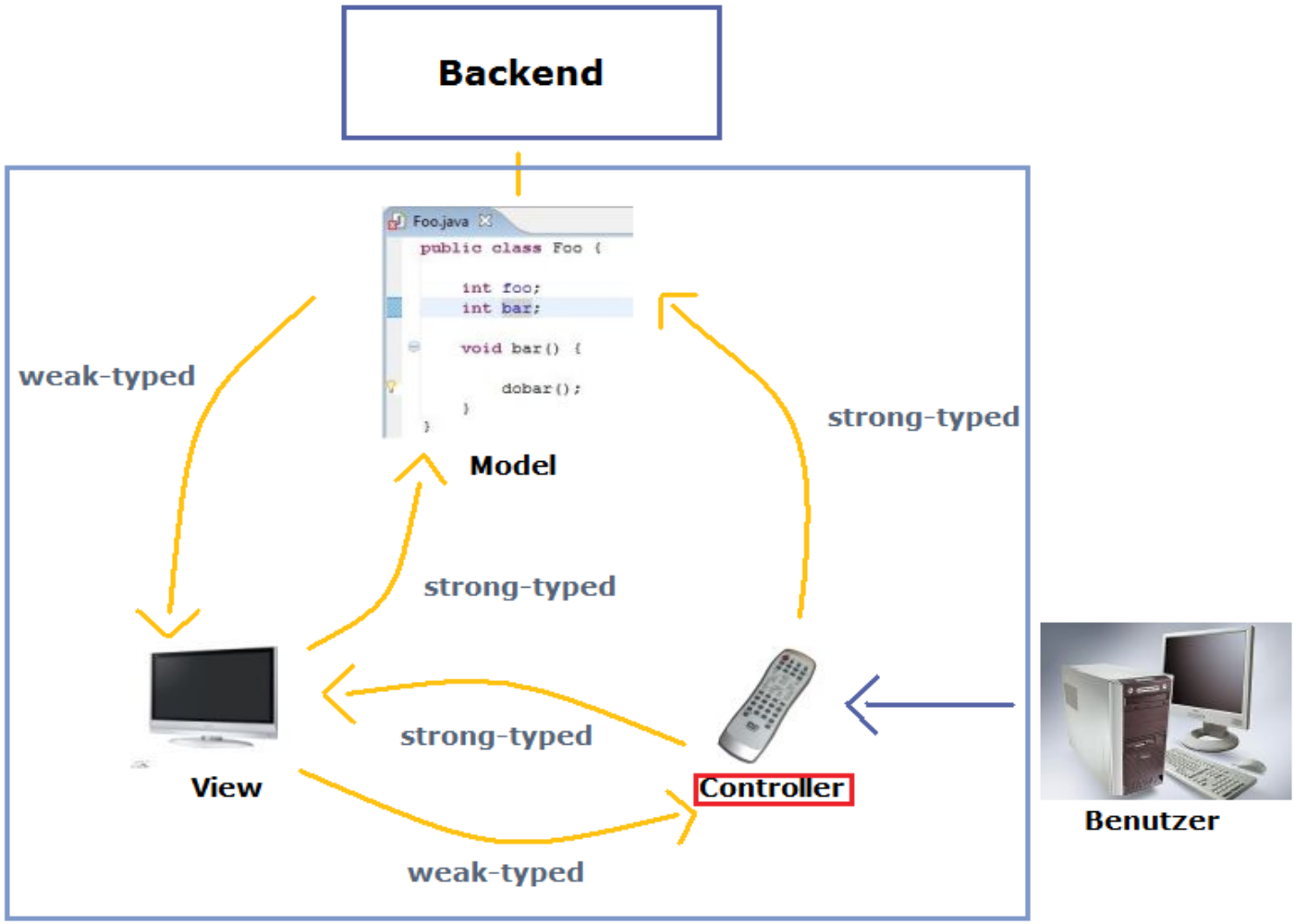
Aufgabenverteilung

Die Komponenten im Überblick: **View**



Aufgabenverteilung

Die Komponenten im Überblick: **Controller**



Zusammensetzung von MVC

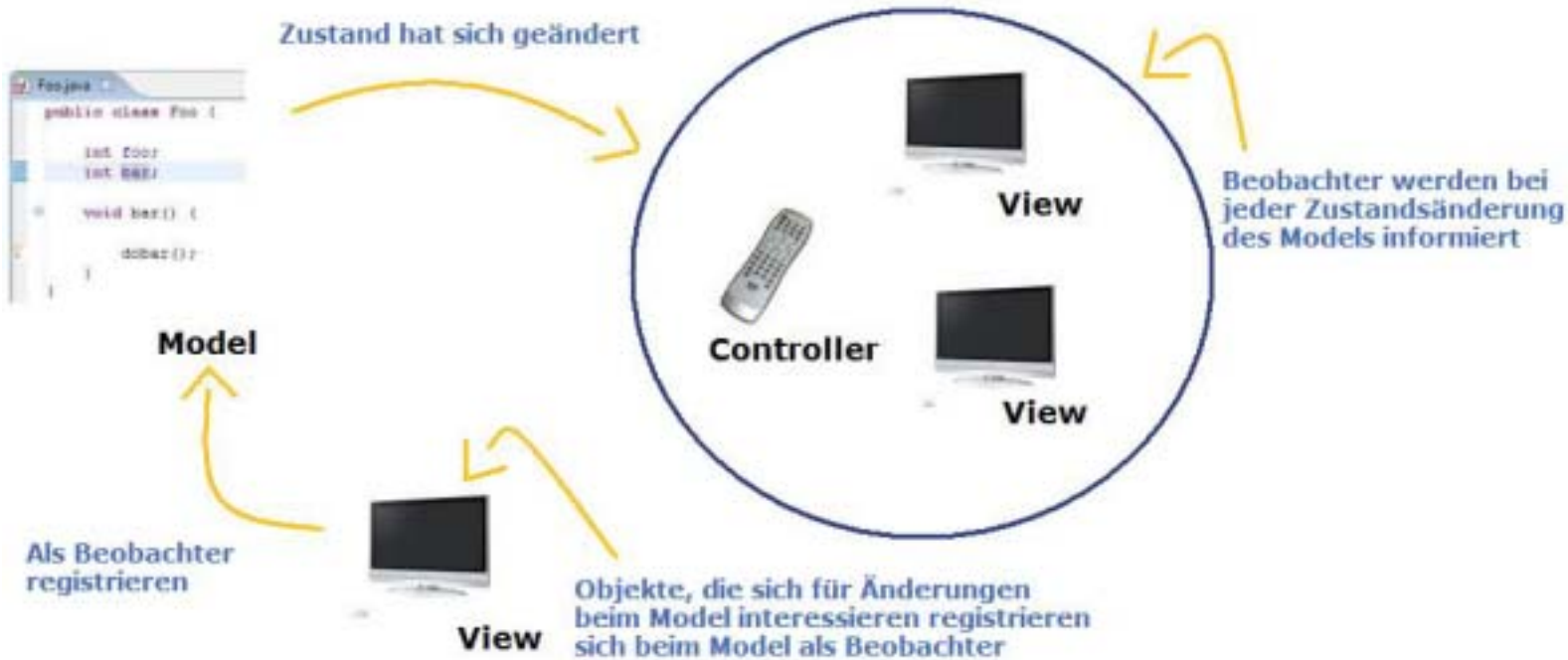
- MVC ist im Prinzip nichts anderes als ein Satz von verschiedenen Mustern, die in diesem Entwurf zusammenarbeiten:

- **Model:** Observer Pattern

- **View:** Composite Pattern,
  Strategy Pattern

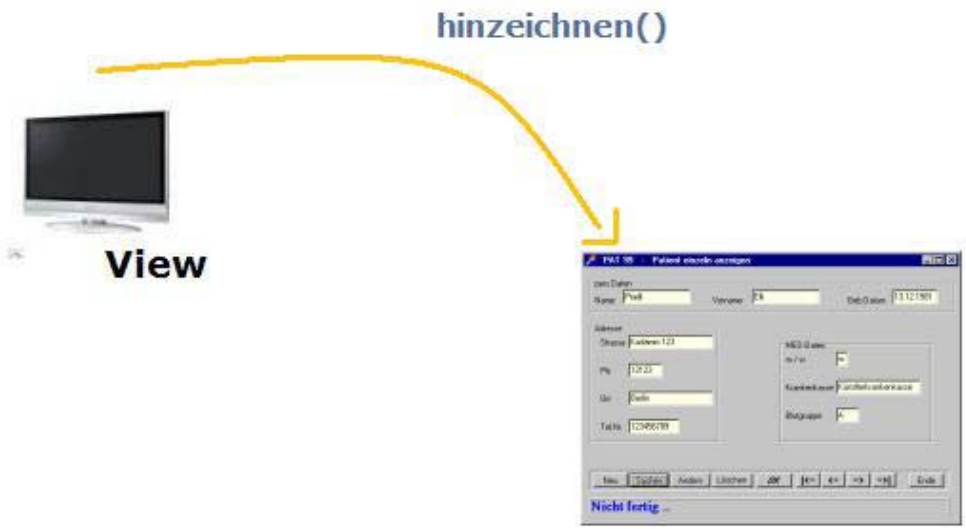
- **Controller:** Strategy Pattern

Zusammensetzung von MVC



Observer-Pattern

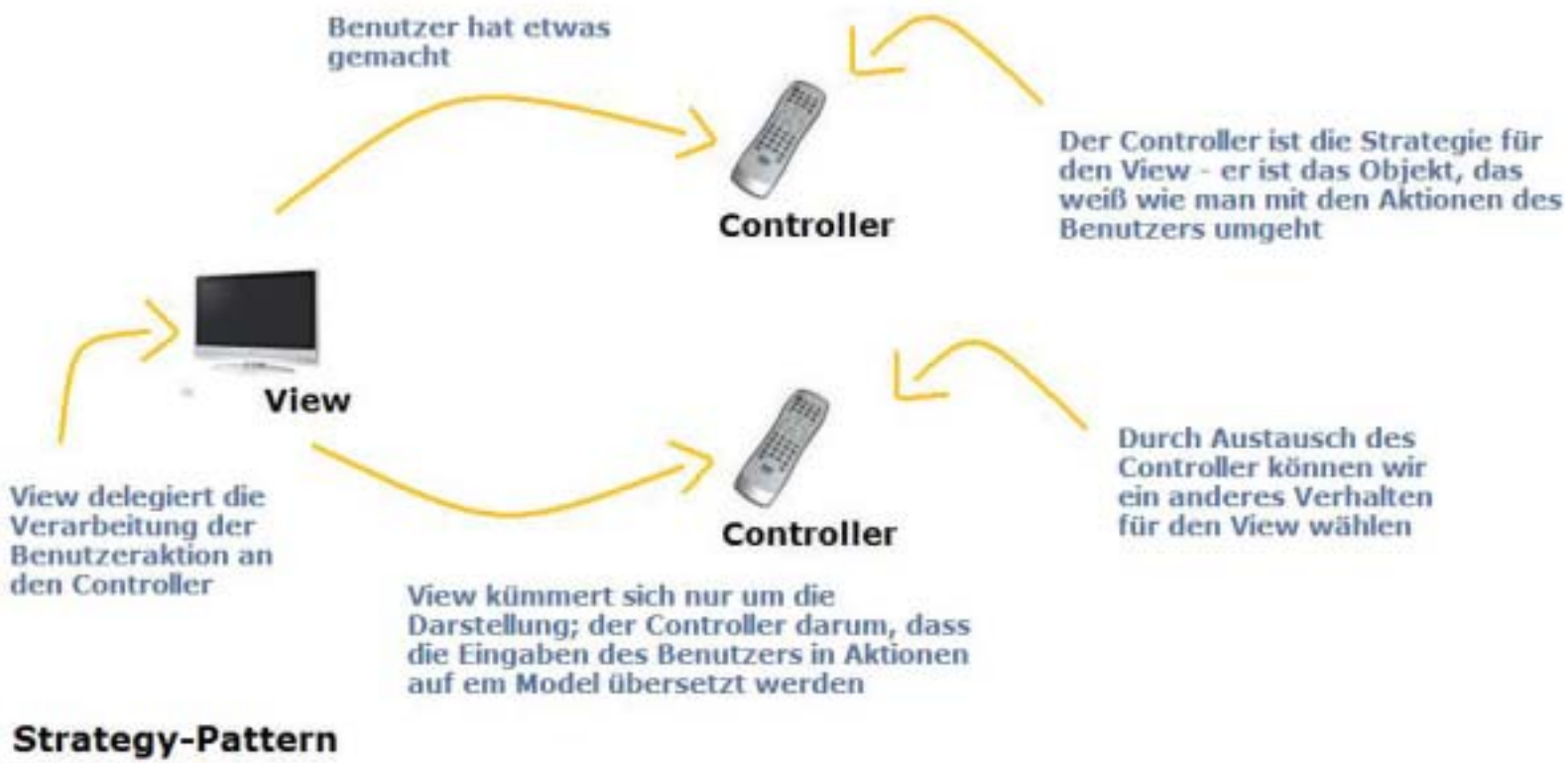
Zusammensetzung von MVC



Der View ist ein Kompositum aus GUI-Komponenten (Labels, Buttons)
Die oberste Komponente enthält andere Komponenten, die weitere Komponenten enthalten kann

Composite-Pattern

Zusammensetzung von MVC



Design Patterns | MVC

Marcus Köhler
Axel Reusch
Markus Merath

Agenda

- Allgemeines
- Aufgabenverteilung

Marcus Köhler

- Alltagsbeispiel
- Beurteilung
- Konkretes Code-Beispiel

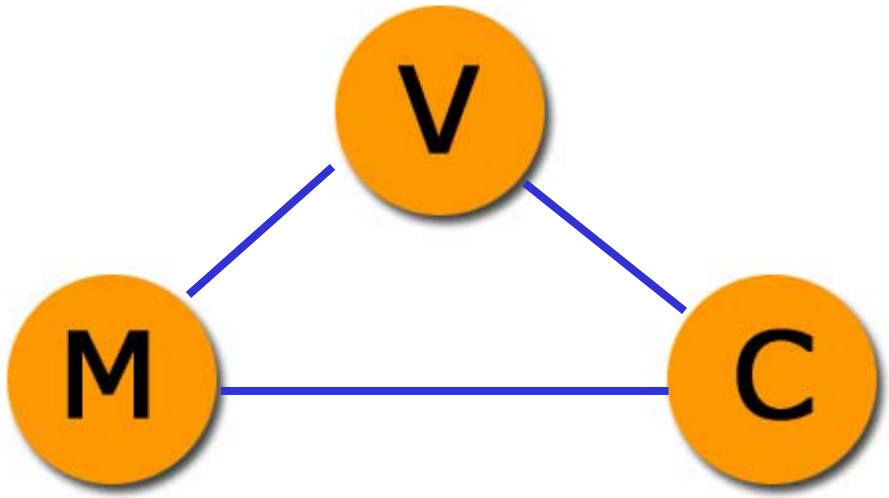
Axel Reusch

- Einsatz im Web
- Webbeispiel
- Fazit & Ausblick

Markus Merath

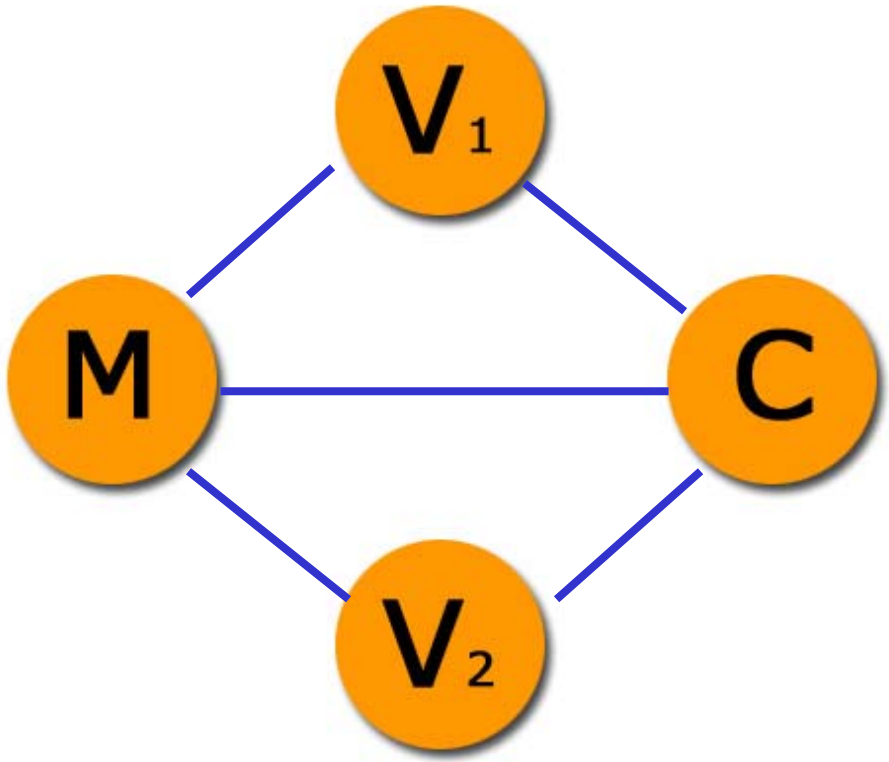
Alltagsbeispiel

- MVC im Alltag



Alltagsbeispiel

- MVC im Alltag
  - Ansprechen von 2 Views



Alltagsbeispiel

- MVC im „Alltag“
  - Alltag in der „Matrix“

  - View 1: Neo in der Großstadt

Alltagsbeispiel

- MVC im „Alltag“
  - Alltag in der „Matrix“

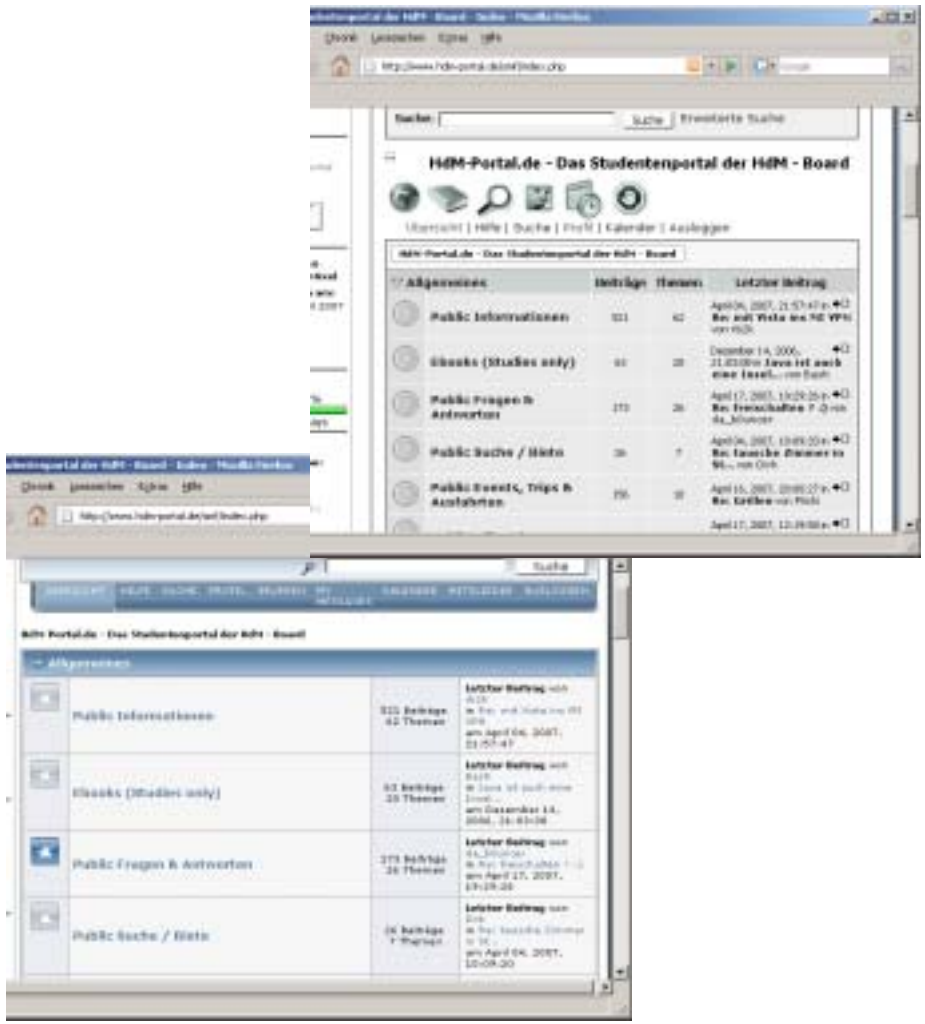
  - View 2: Neo auf‘m Land

Alltagsbeispiel

- MVC im „Alltag“
  - Alltag in der „Matrix“
    - Nur die visuelle Umgebung ändert sich
    - Daten und Methoden bleiben gleich!
      - View1: Stadt
      - View2: Land
      - Model: Personen, Gegenstände
      - Controller: Interaktion der Personen mit Gegenständen / Personen

Alltagsbeispiel

- Weitere Beispiele



Beurteilung

- Vorteile
  - Bessere Wartbarkeit
  - Bessere Erweiterbarkeit
  - Bessere Übersichtlichkeit
  - Modularisierung

Beurteilung

- Nachteile
  - Evtl. Performance-Verlust
  - Höhere Komplexität

Konkretes Code-Beispiel

- Kleines Count-Programm
  - View: Ergebnis
  - Anderer View: Farbe
  - Controller:
    Counting, Kommunikation
  - Model: momentaner Datenbestand

- >> im Eclipse

Design Patterns | MVC

Web MVC

Marcus Köhler
Axel Reusch
Markus Merath

Agenda

- Allgemeines
- Aufgabenverteilung
- Alltagsbeispiel
- Beurteilung
- Konkretes Code-Beispiel
- Einsatz im Web
- Webbeispiel
- Fazit & Ausblick

Marcus Köhler

Axel Reusch

Markus Merath

Inhalt

- Inhalt
  - Umgebung
  - Ablauf
  - Bestandteile in einem Beispiel
    - Model
    - View
    - Controller
  - Beispiel Demo

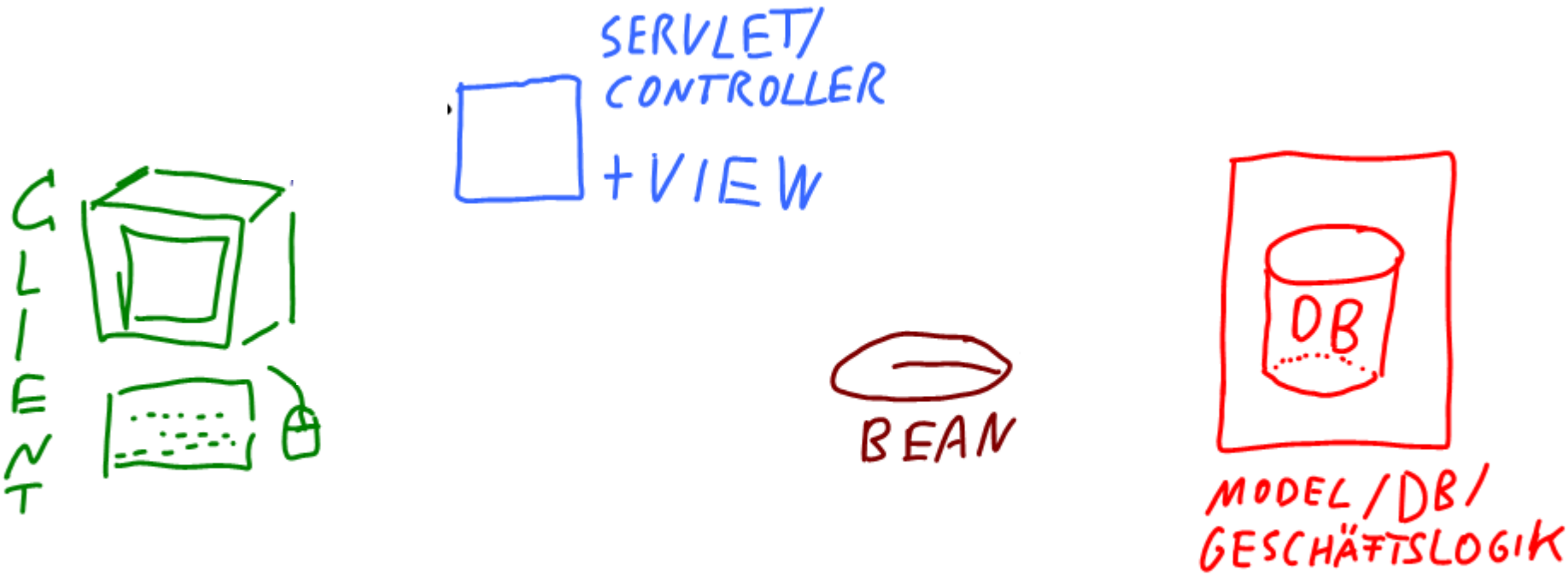
Das Web / MVC Entwicklung

- früher: statisches HTML
  - offensichtlich: zu unflexibel
- CGI und Skriptsprachen
  - zu wenig strukturiert
  - zu viel Mehrfacharbeit
- J2EE, Frameworks, Application Server
  - Entwicklung noch aktiv (Java Server Faces!)
  - Model 1 & 2 Architekturen: MVC Variationen

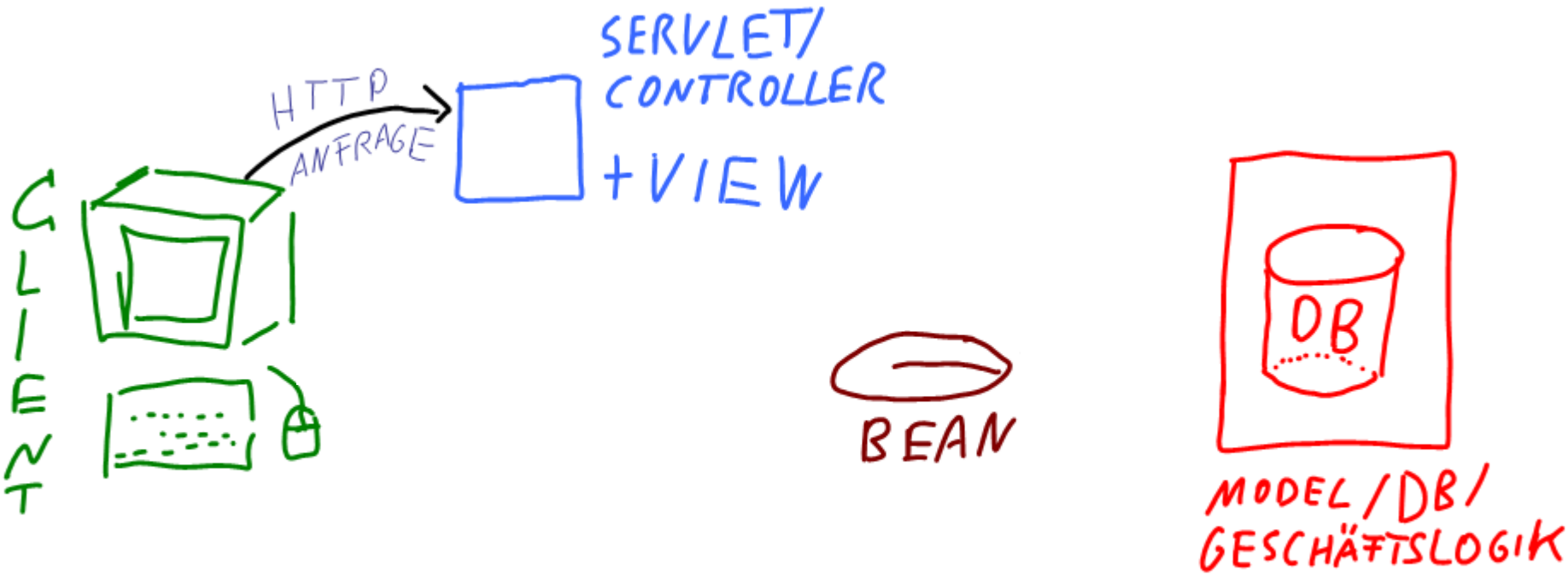
Das Web / Model 1

- "seiten-zentrisch"
- Trennung von View und Model z. Bsp. über Beans
  - provoziert aber doch Vermengung
  - Arbeitstrennung trotzdem problematisch
- Status über Get/Post
- traditionelle, harte Verlinkung von JSPs
  - starke Kopplung :(
- kein zentraler Controller
  - bspw. Input-Validierung auf *jeder* Seite

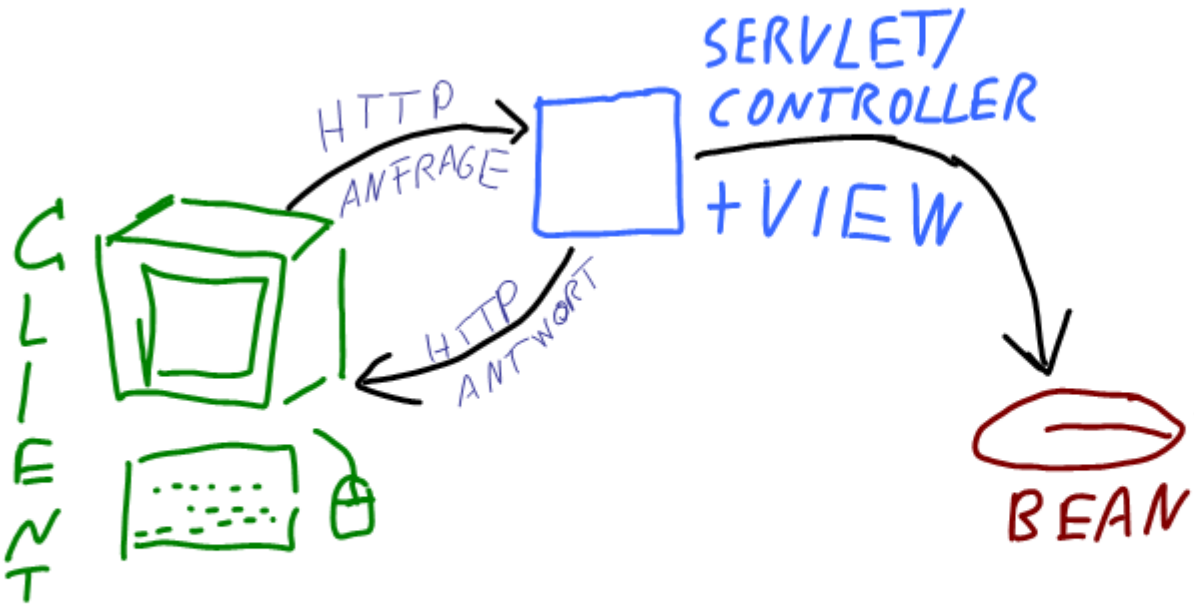
Der Ablauf im Model 1



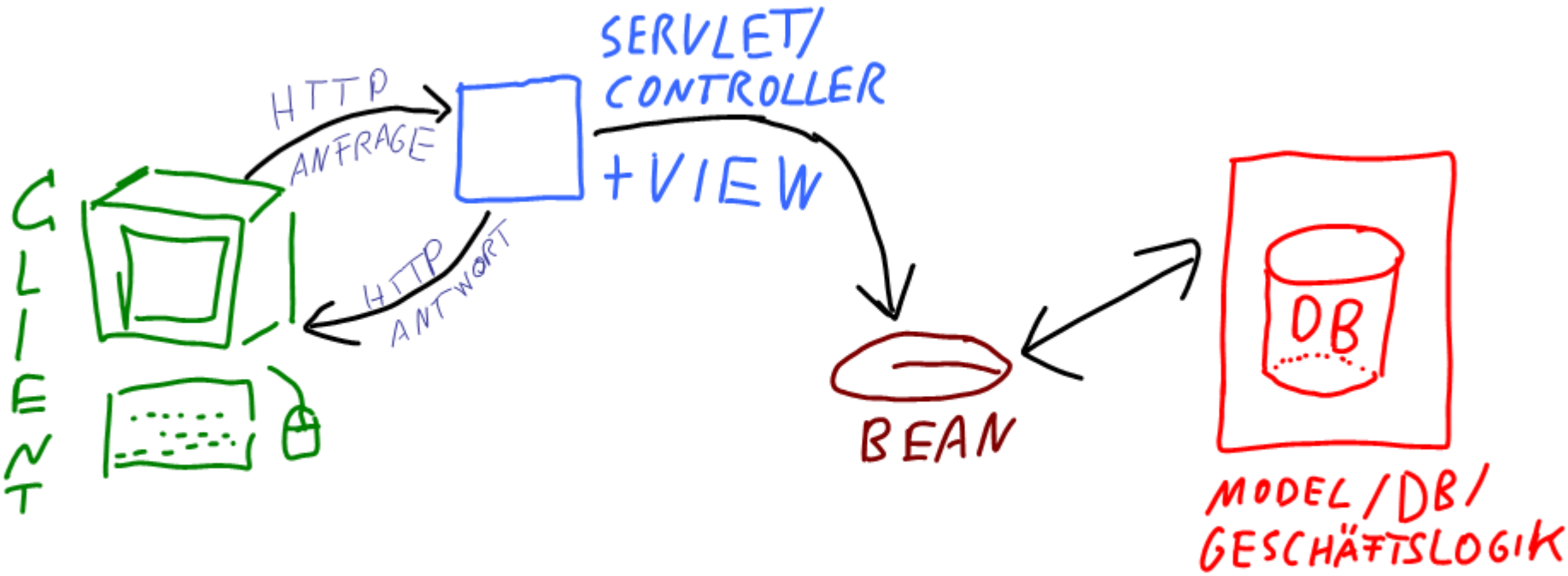
Der Ablauf im Model 1



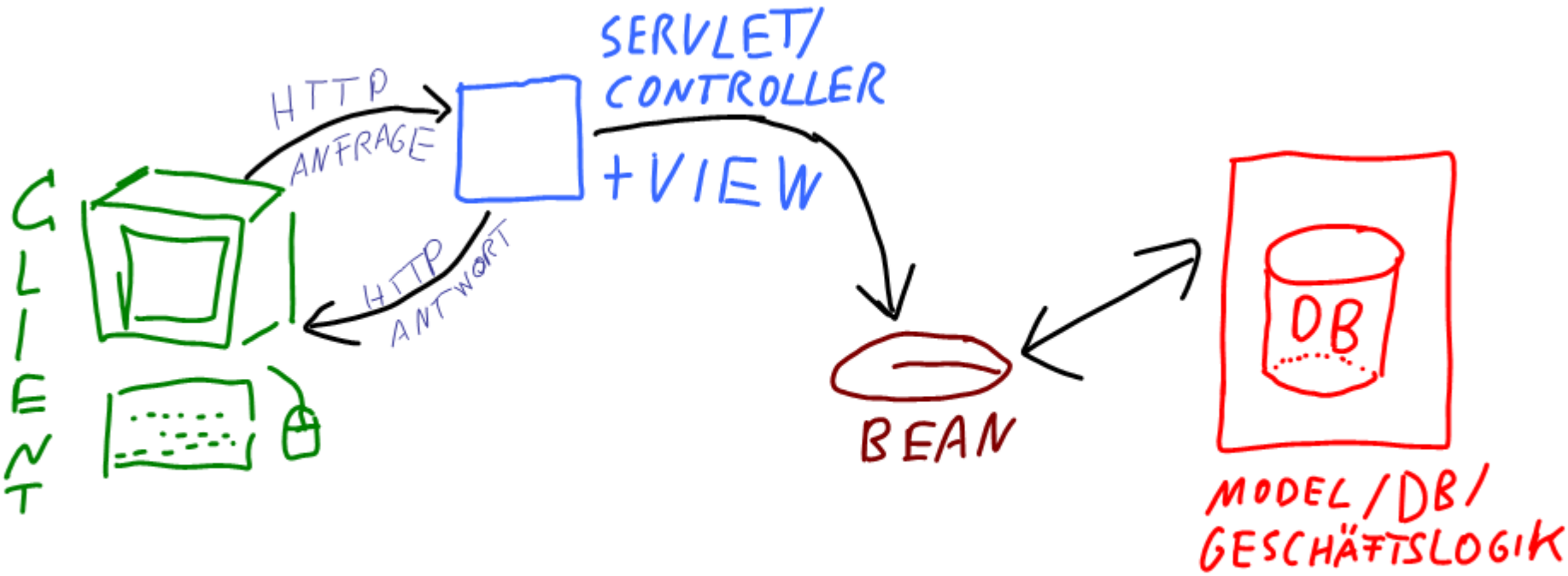
Der Ablauf im Model 1



Der Ablauf im Model 1



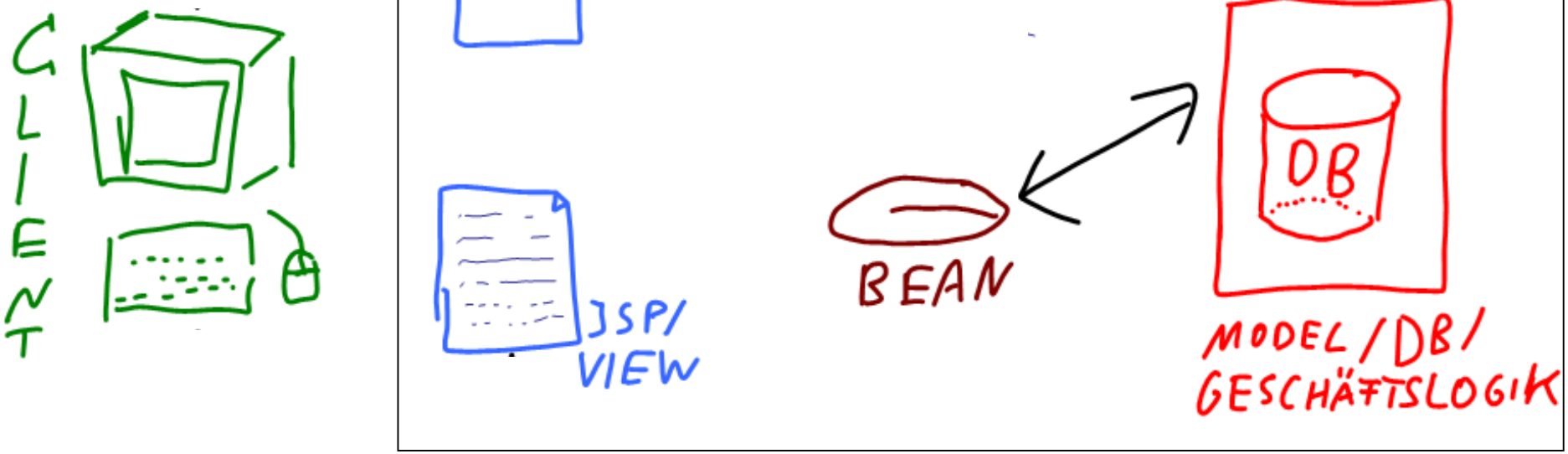
Der Ablauf im Model 1



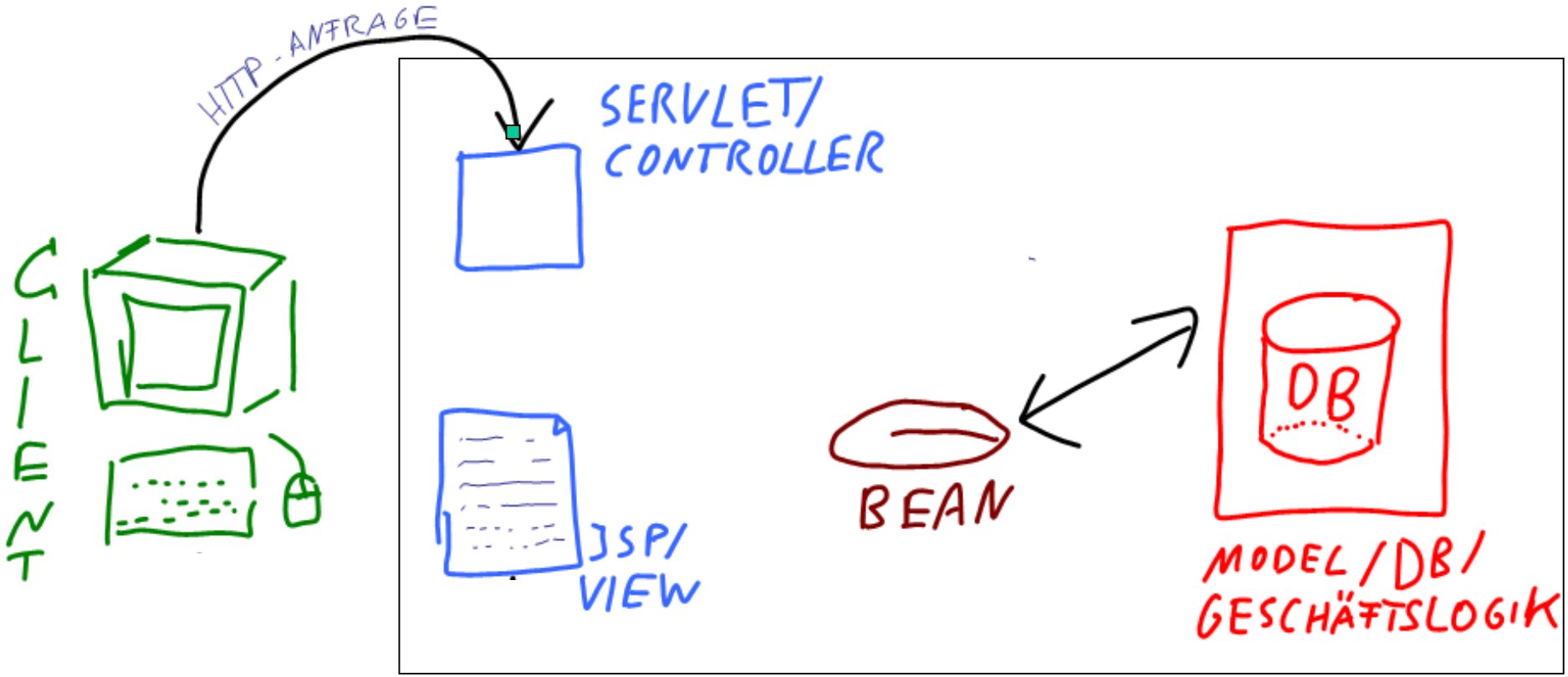
Das Web / Model 2

- empfohlen für mittlere bis große Projekte
- "servlet-zentrisch"
- ein zentrales Controller-Servlet arbeitet als Dispatcher
  - relevante Patterns: Front Controller, Application Controller
  - validiert Eingabe, wählt Sprache, etc.
  - wählt View aus
- Views sind JSPs
  - interagiert mit Model (typischerweise JavaBean(s))
  - gibt Response zurück an Servlet
- z.Bsp. Struts oder JavaServer Faces Framework

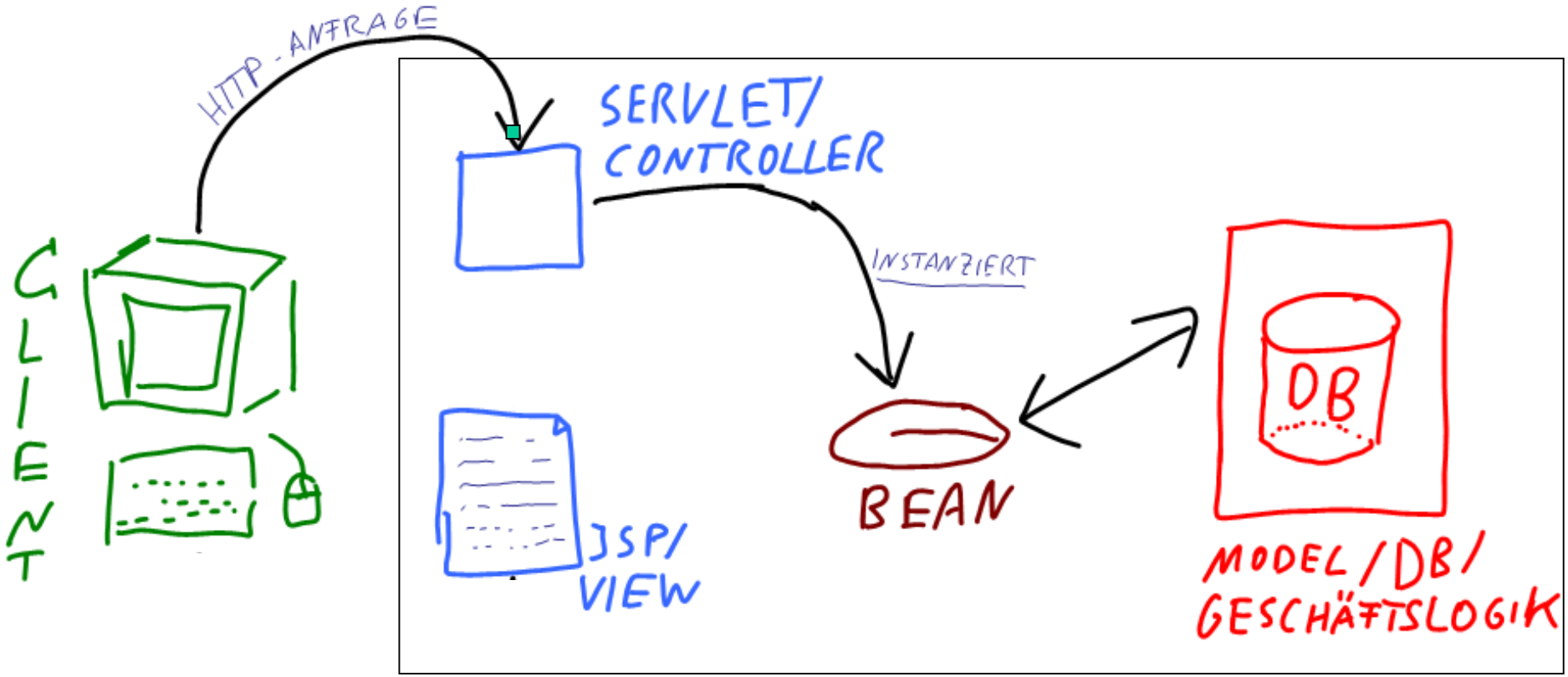
Der Ablauf im Model 2



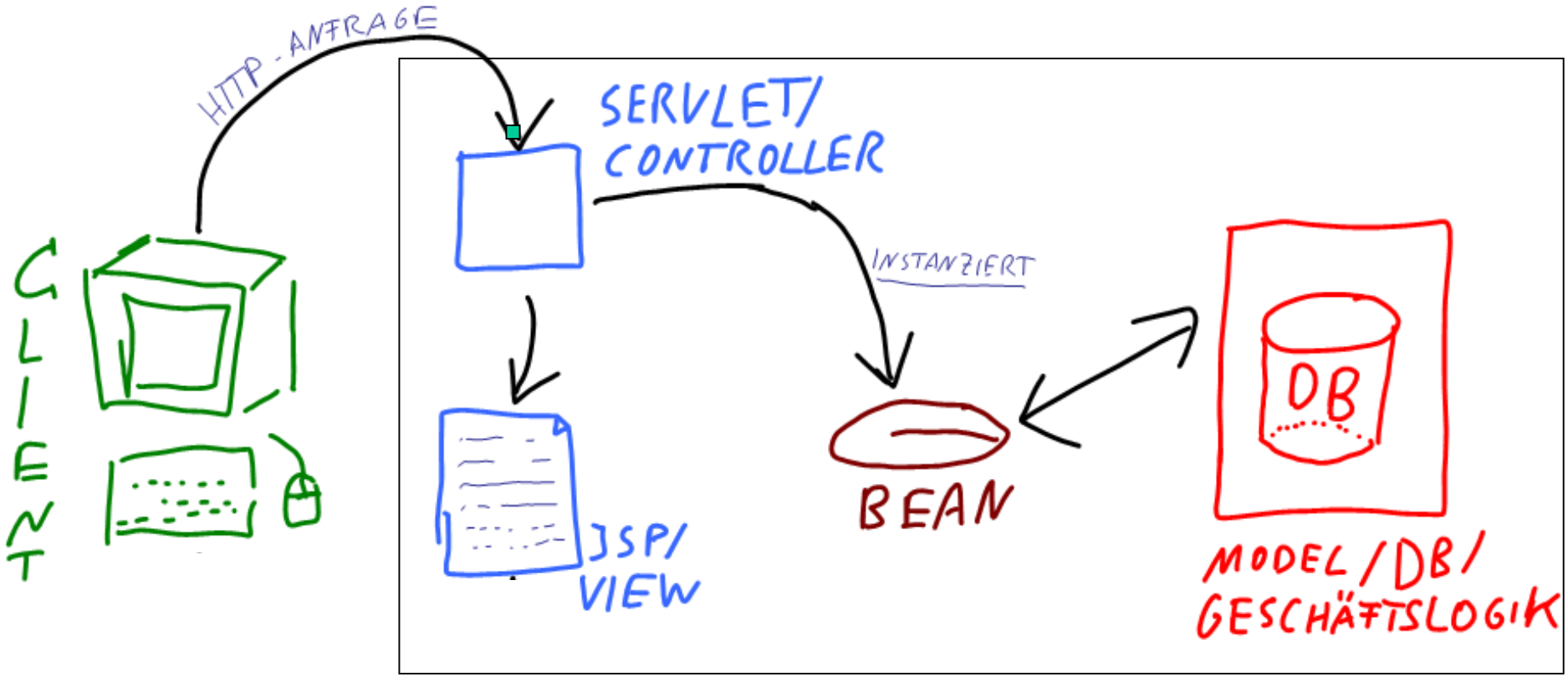
Der Ablauf im Model 2



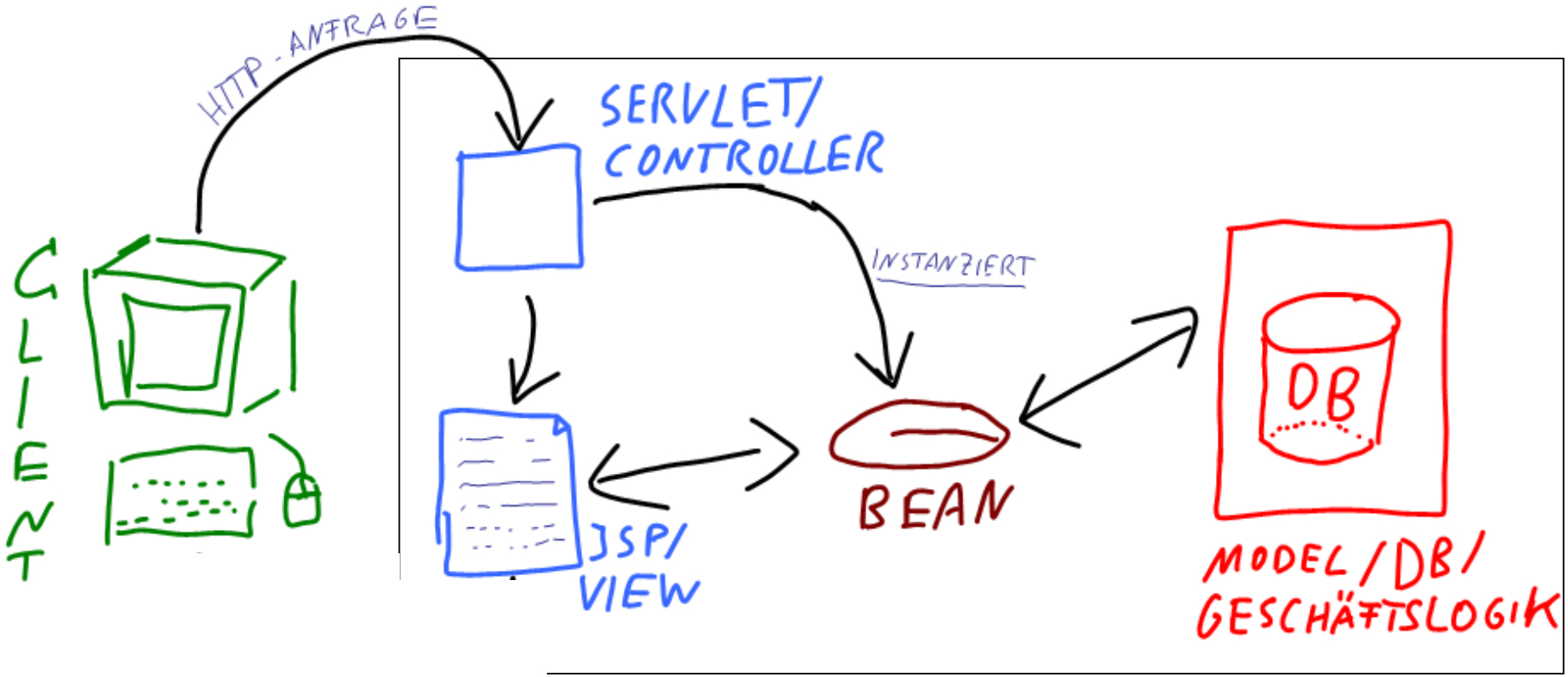
Der Ablauf im Model 2



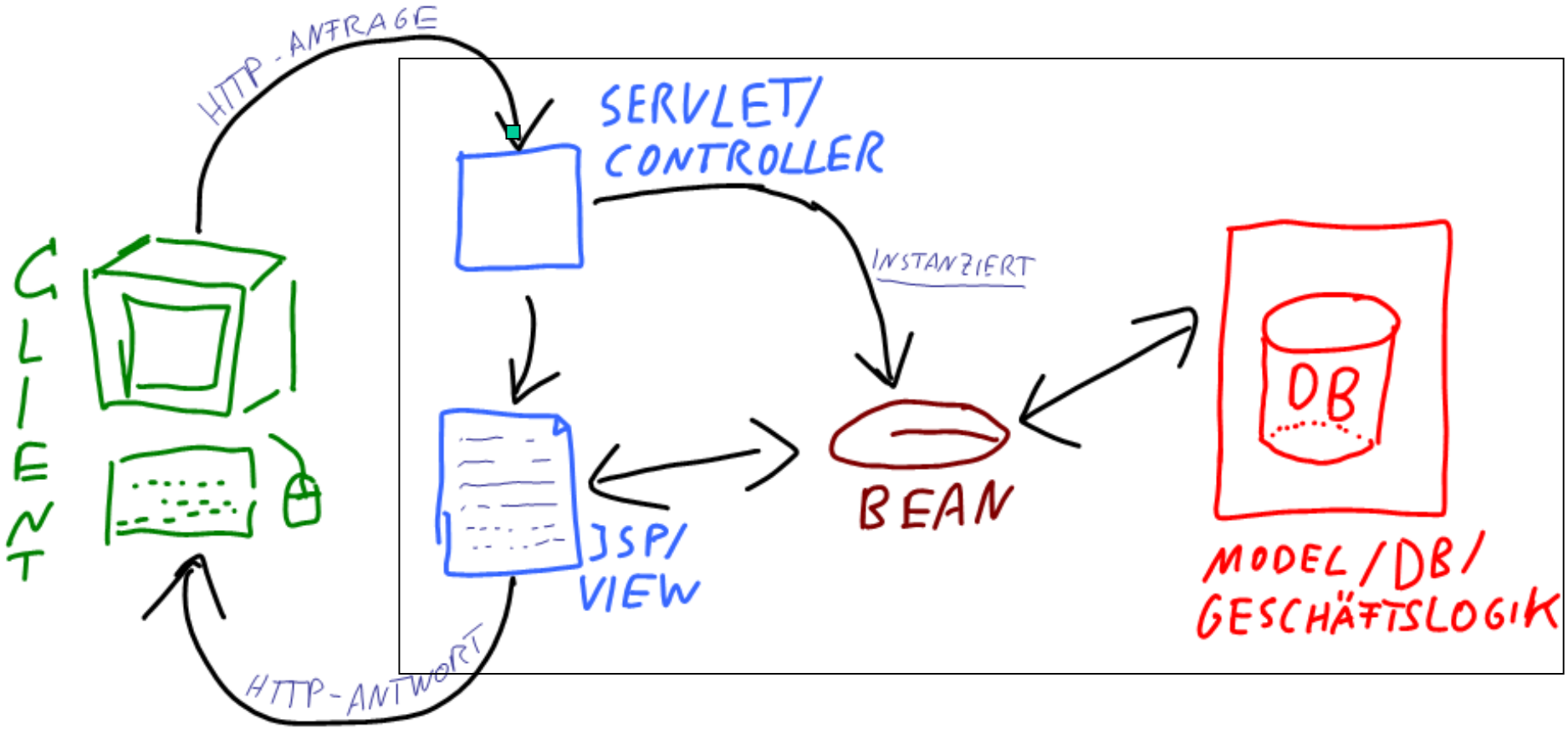
Der Ablauf im Model 2



Der Ablauf im Model 2



Der Ablauf im Model 2

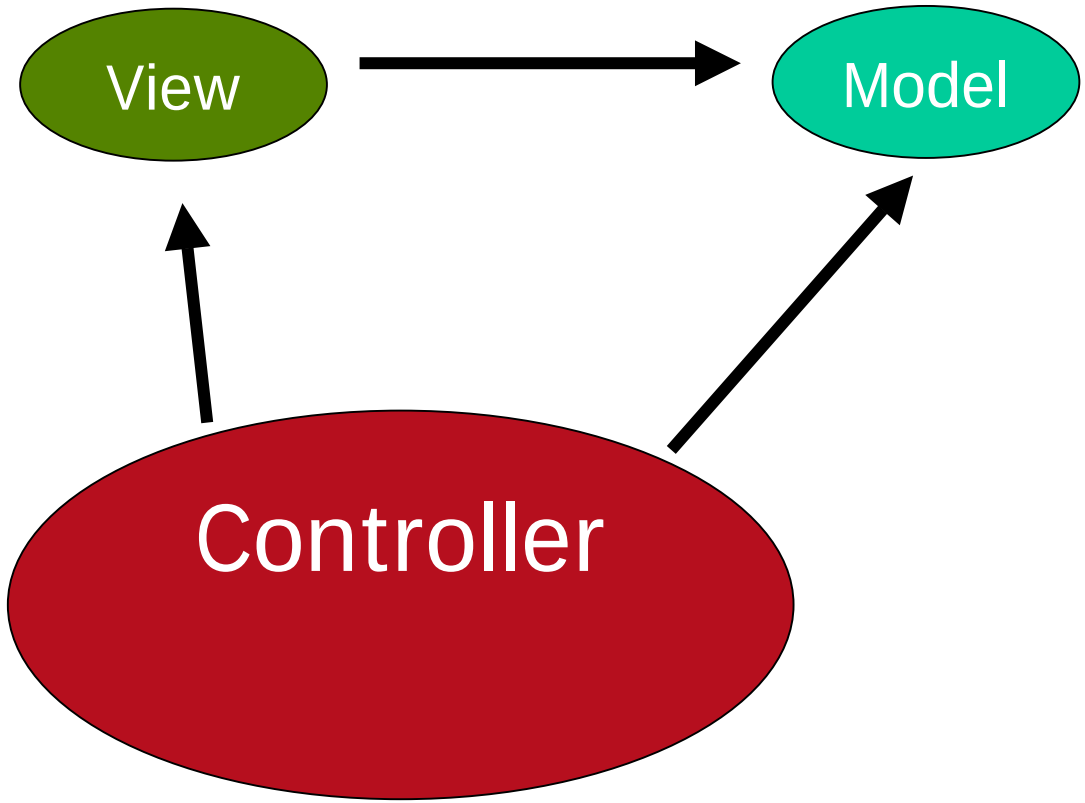


MVC im Web

- Beispielszenario
  - Beispiel Applikation welche eine Kantenlänge eines Quadrates durch Benutzereingabe aufnehmen kann und entweder den Flächeninhalt generiert.
  - Beispiel ist im JavaServer Faces Framework angelegt.

Die Komponenten

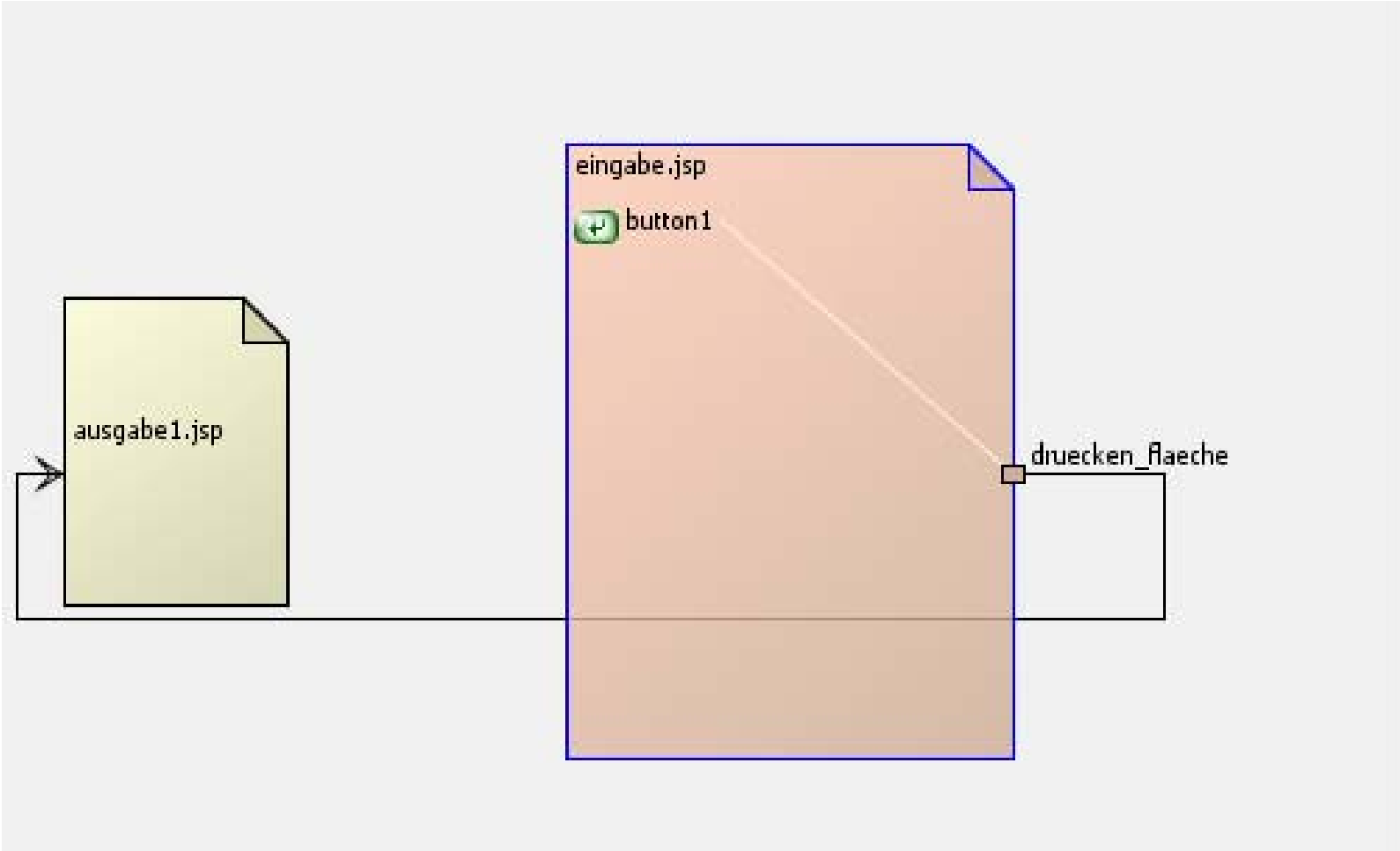
- faces-config.xml



Die Komponenten

- Anwendungskonfigurationsdatei faces-config.xml:
  - JSF-spezifische Konfigurationsdatei
  - Enthält Angaben zum Beanmanagement
  - Definiert die Navigation
  - Verschiedene weitere Angaben zur JSF-Anwendung

Die Komponenten Faces-config.xml

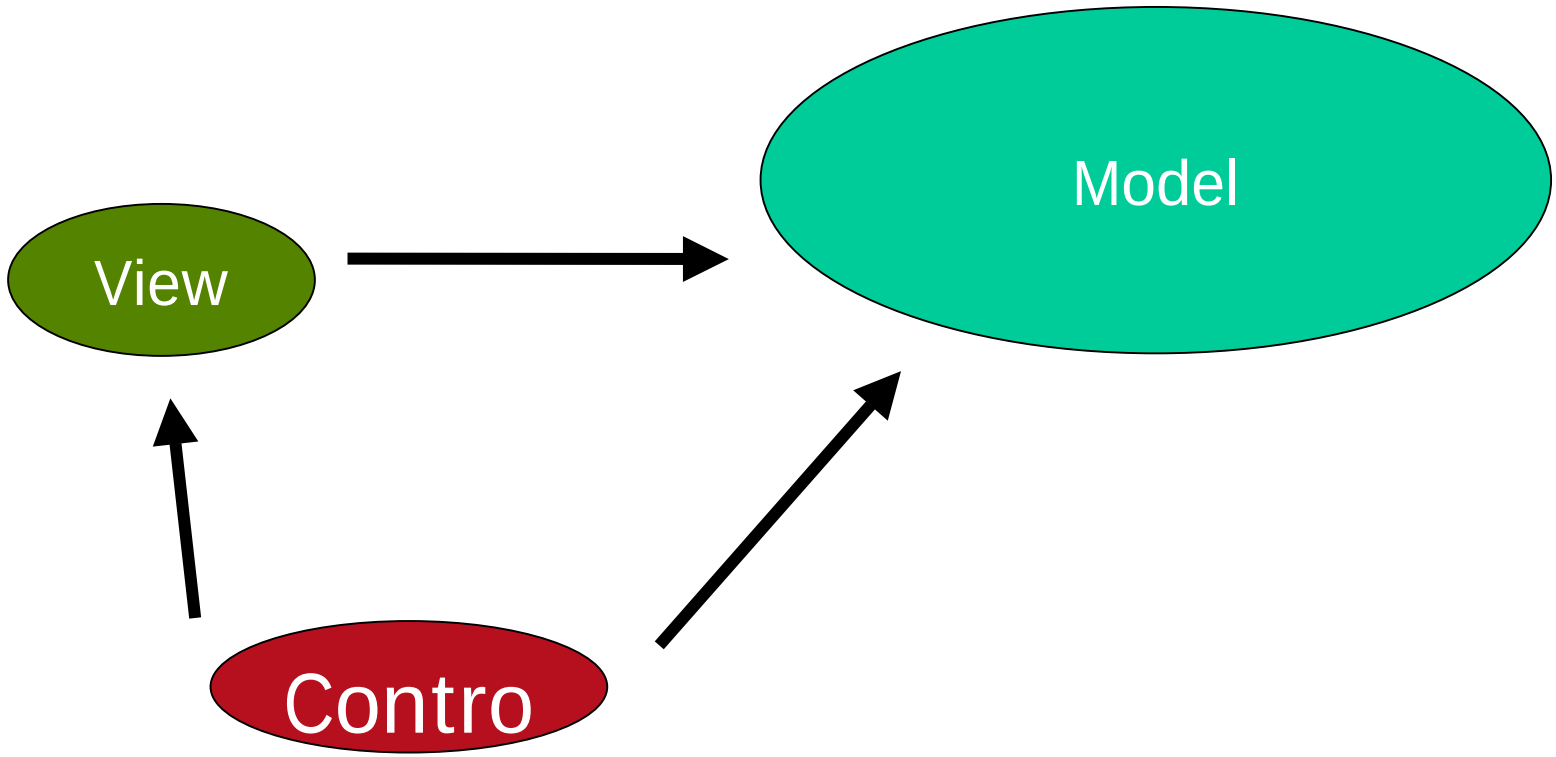


Die Komponenten Faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
  <managed-bean>
    <managed-bean-name>Playground</managed-bean-name>
    <managed-bean-class>test.PlaygroundBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  <navigation-rule>
    <from-view-id>/eingabe.jsp</from-view-id>
    <navigation-case>
      <from-outcome>druecken_flaeche</from-outcome>
      <to-view-id>/ausgabe.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

Die Komponenten

- Bean

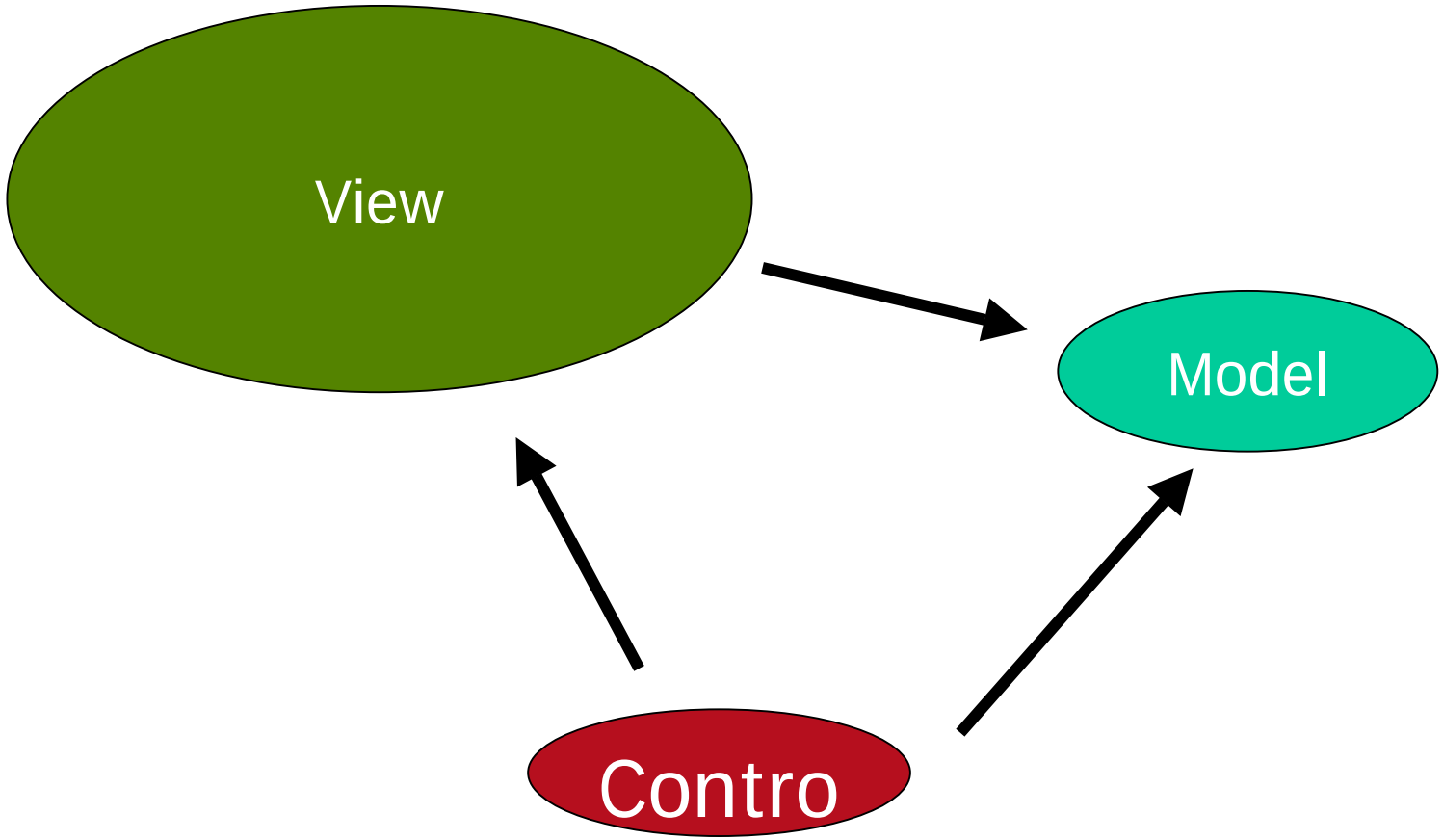


Die Komponenten / die Bean

```
package test;
public class PlaygroundBean{
private int length;
private int area;
public PlaygroundBean(){
}
public void setArea(int i) {
this.area = i;
}
public int getArea() {
return this.length * this.length;
}
public void setLength(int i) {
this.length = i;
}
public int getLength() {
return this.length;}}
```

Die Komponenten

- View



Die Komponenten / JSP-Eingabe

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <f:view>
   <h:outputLabel>……
   <h:form id="inputForm">
     <h:outputLabel>……
     <h:inputText value="#{Playground.length}"/>
     <h:commandButton value="Berechnen" action=„flaeche" />
   </h:form>
 </f:view>
```

Die Komponenten / JSP-Ausgabe

```
<%@ tagliburi="http://java.sun.com/jsf/html" prefix="h" %>
<%@ tagliburi="http://java.sun.com/jsf/core" prefix="f" %>
 <f:view>
   <h:form id="inputForm">
   <h:outputLabel>.....
   <h:outputText value="#{Playground.length}"/>

      <h:outputText value="#{Playground.area}"/>
   </h:form>
  </f:view>
```

MVC im Web

- Beispiel
  - Ansicht im Browser und Eclipse

MVC im Web

- Zusammenfassung
  - Ist in der Praxis sehr üblich
  - Notwendig bei unterschiedlichen Ausgabegeräten
  - Sehr übersichtlich
  - Gut wartbar
  - Sehr flexibel
  - Wieder benutzbar
  - Lohnt sich erst ab einer gewissen Größe

MVC im Web

- Vielen Dank für eure Aufmerksamkeit
- Zeit für Fragen!!!