FH Bielefeld
University of
Applied Sciences

Campus Minden

Webbasierte Anwendungen

SS 2018

Web-Entwicklungsmuster

Dozent: B. Sc. Florian Fehring
mailto: florian.fehring@fh-bielefeld.de

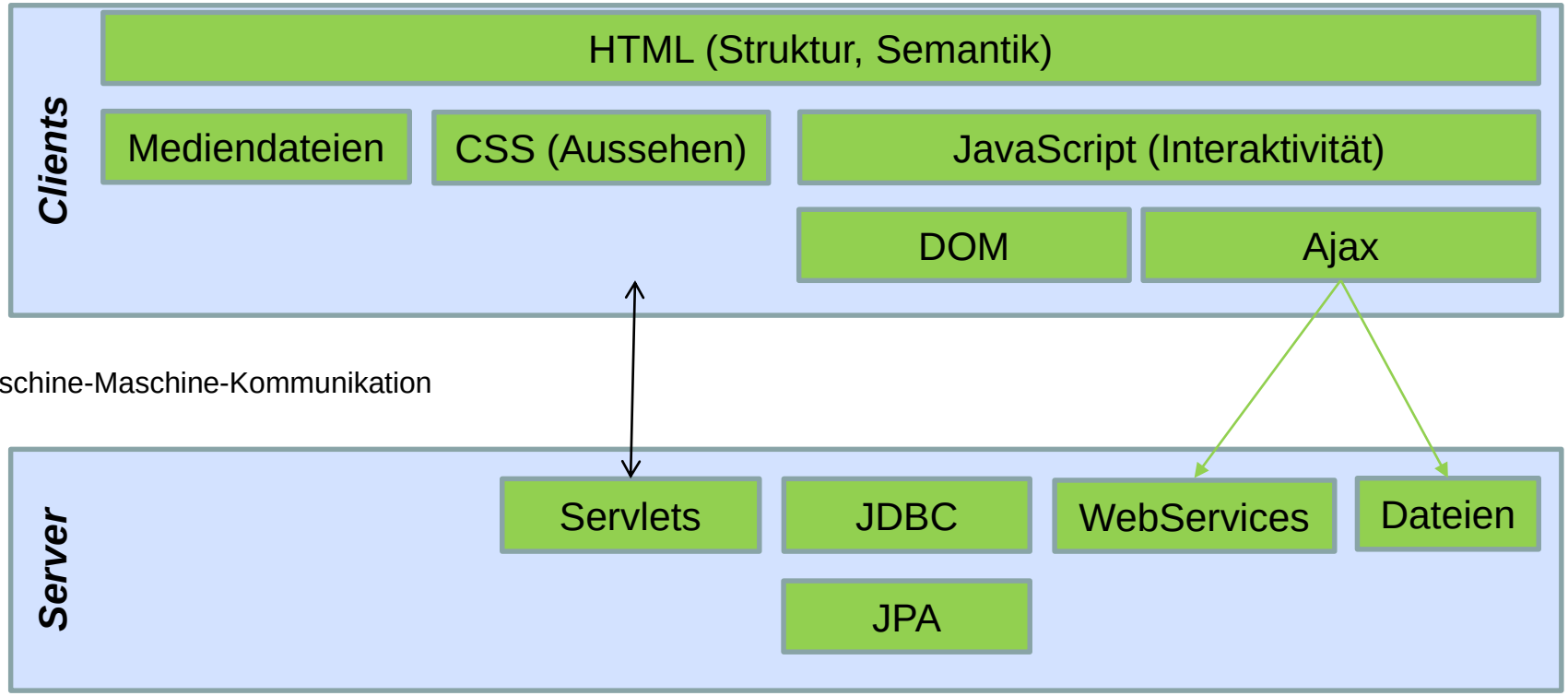
Studiengang Informatik

Entwicklungsmuster

1. **Kontext und Motivation**
2. Kategorien
3. Model 1 und Model 2 Muster
4. MV*-Muster
5. Darüber hinaus
6. Projekt

Problemfelder

Mensch-Maschine-Kommunikation



Maschine-Maschine-Kommunikation

Entwicklungsmuster

1. Kontext und Motivation
2. **Kategorien**
3. Model 1 und Model 2 Muster
4. MV*-Muster
5. Darüber hinaus
6. Projekt

Kategorien

Definition: **Entwicklungsmuster sind Schablonen für die Umsetzung von Problemlösungen.**

Arten von Mustern:

- Architekturmuster
  - Bilden Lösungen für die Organisation und Kommunikation von Softwarekomponenten. Sie bilden somit Strukturen in größerem Rahmen ab.
    - Verteilungsmuster — Festlegung der Aufteilung einer Software auf Hardware
    - Strukturmuster — Schichten- und Reihenfolgen-Festlegung
    - Interaktivitätsmuster — Verarbeitung von Benutzerinteraktionen
    - Adaptionsmuster — Erweiterungs- und Anpassungsfähigkeit von Softwaresystemen
- Entwurfsmuster
  - Bilden Lösungen für die Implementierung von häufig anzutreffenden Problemen. Sie bilden Strukturen in kleinerem Rahmen ab.
    - Erzeugungsmuster — Entkopplung von Erzeugung und Nutzung / Repräsentation
    - Strukturmuster — Beziehungen zwischen Klassen
    - Verhaltensmuster — Beschreibung von Verhalten und Abläufen
    - Muster für objektrelationale Abbildung — Abbildungen für das OR-Mapping

Komponenten

Definition: **Komponenten in einem Entwicklungsmuster sind alle verwendeten Softwareteile. In Web-Anwendungen umfasst dies auch Client- und Server-Software**

Komponenten:

- Fertig Implementierte
  - Bibliotheken und Frameworks
  - Browser
  - Application-Server
  - Datenbank-Server
- Zu Implementierende
  - Eine Einheit von:
    - HTML-Dateien
    - (Java) Klassen
    - (Javascript) Klassen
    - CSS-Dateien
    - Bilder und andere Ressourcen

Entwicklungsmuster

1. Kontext und Motivation
2. Kategorien
3. **Model 1 und Model 2 Muster**
4. MV*-Muster
5. Darüber hinaus
6. Projekt

Model 1 und Model 2 - Muster

Definition: **Model 1 und Model 2 sind mehrschichtige Architekturen die aus dem JSP Umfeld stammen.**

Model 1:

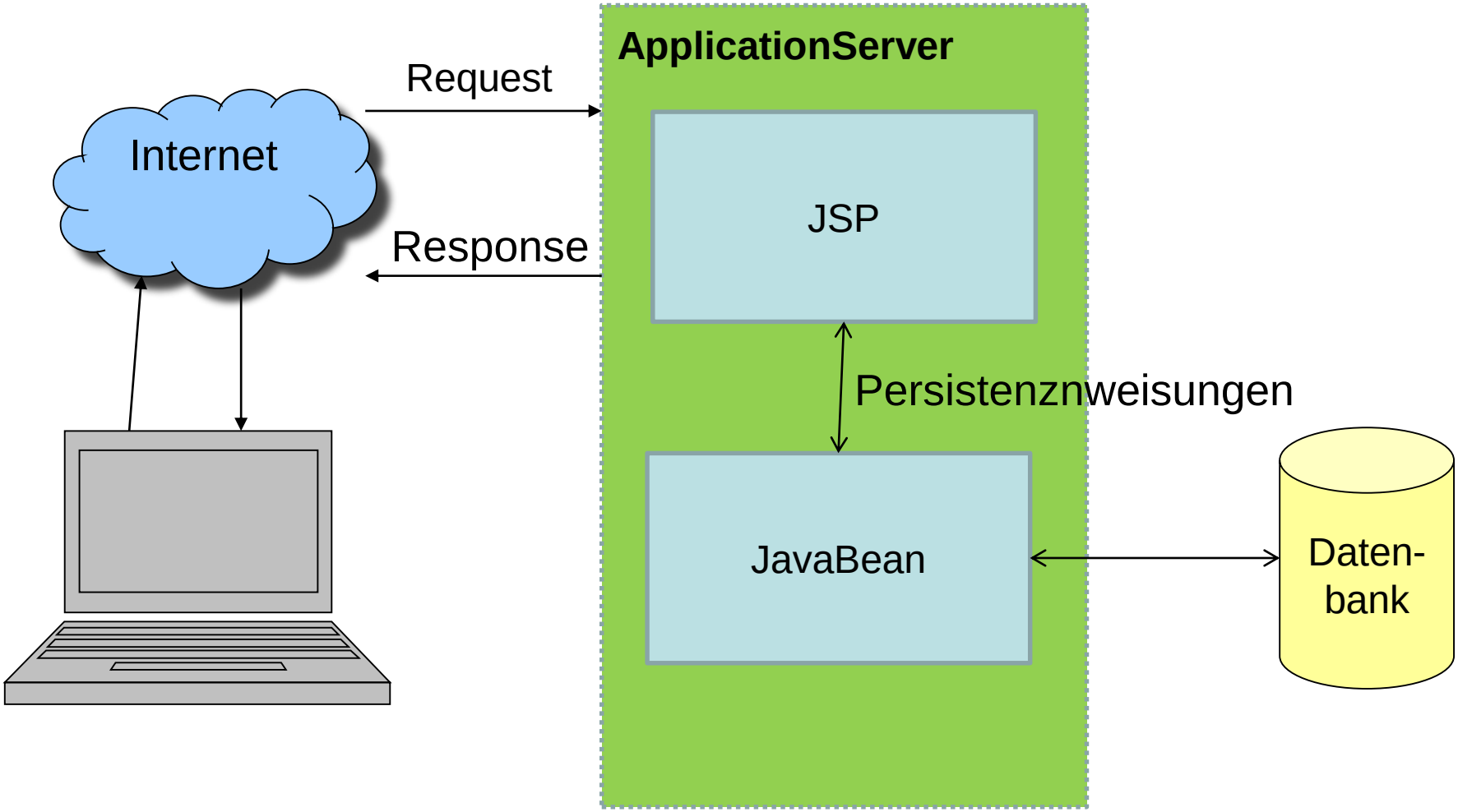
- Für jede Webseite gibt es eine Komponente
  - Enthält Code zur Steuerung (Parameter verarbeiten, Links, Redirects,…)
  - Enthält Code zur Darstellung des Ergebnisses (HTML-Code, Bilder,…)
- Geschäftslogik liegt gewöhnlich außerhalb (weitere Klassen)
  - ➢ saubere Trennung von Präsentation und Geschäftslogik möglich

Model 2:

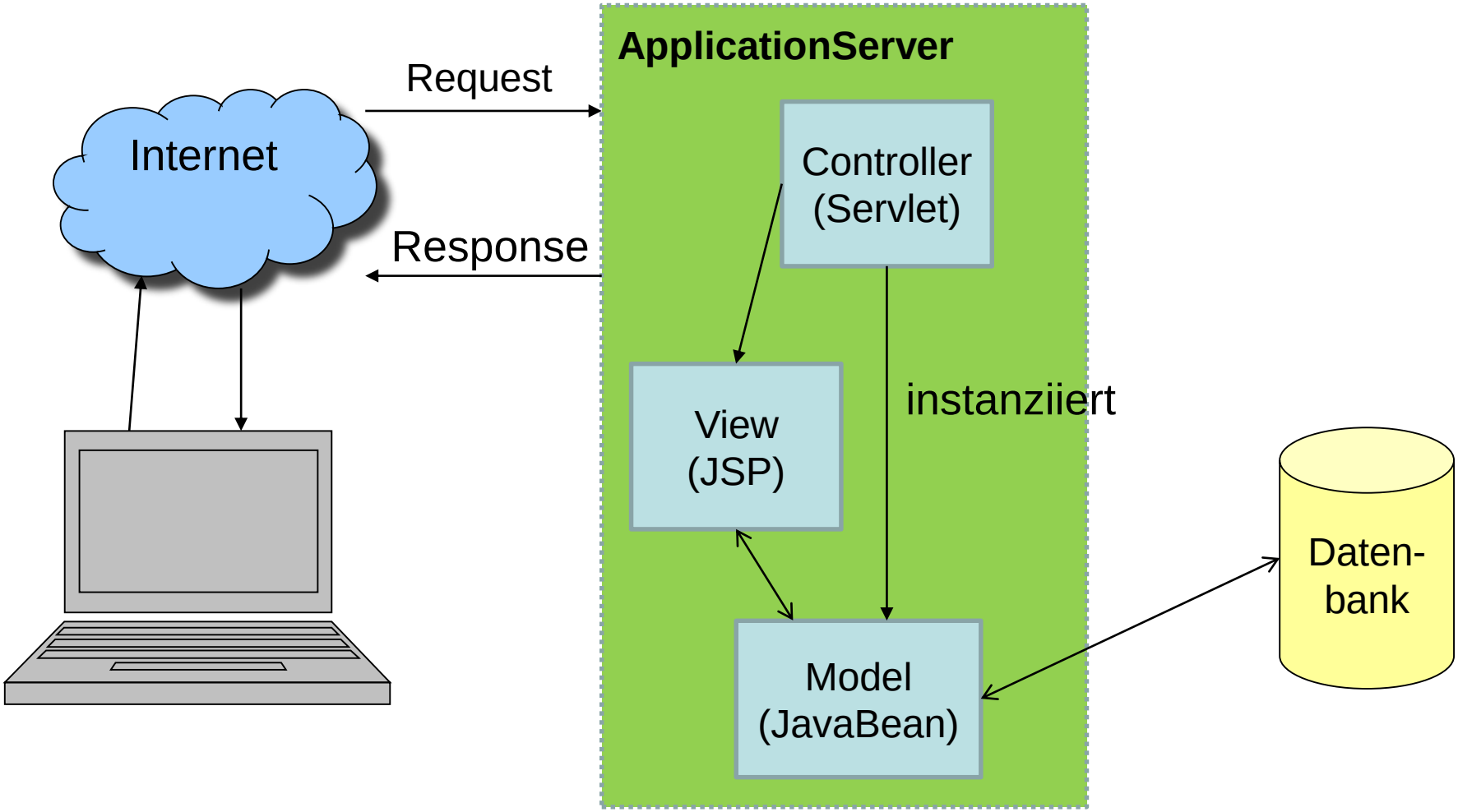
- keine seitenzentrierte Sichtweise mehr
- Für jede Seite gibt es einen Controller
  - Empfängt die Anfrage vom Client
  - Führt Logik zur Beschaffung des Inhalts aus
  - Sagt dem View, welchen Inhalt er aufnehmen soll
  - Leitet den fertigen View an den Client weiter

Beide Muster wurden in einer frühen Version von JSP beschrieben.

Model 1


ApplicationServer
Request
Internet
Response
JSP
Persistenznweisungen
JavaBean
Daten-
bank

Model 2


ApplicationServer
Request
Internet
Response
Controller
(Servlet)
View
(JSP)
instanziiert
Model
(JavaBean)
Daten-
bank

Entwicklungsmuster

1. Kontext und Motivation
2. Kategorien
3. Model 1 und Model 2 Muster
4. **MV*-Muster**
5. Darüber hinaus
6. Projekt

MV* Muster

Definition: **MV* ist eine Familie von ähnlichen Architekturmustern für graphische Oberflächen.**

Typisch für ***graphische Oberflächen*** ist, dass es Elemente zur ***Eingabe*** gibt, die zur ***Bearbeitung der eigentlichen Inhaltsobjekte*** führen, die dann eventuell zu ***Änderung der Anzeige*** führen. Das MVC trennt die drei Aufgaben.

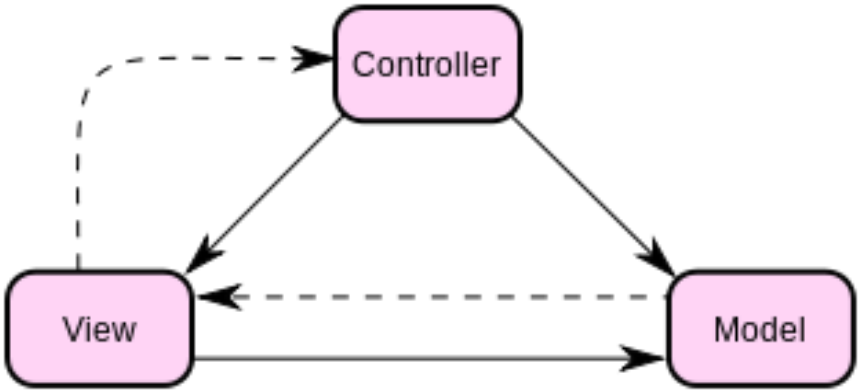
Idee: Controller steuert Änderungen des Modells, Modell teilt allen Views mit, dass eine Änderung aufgetreten ist.

Die Muster der MV*-familie werden nicht nur in Webapplikationen eingesetzt. Allen Mustern dieser Familie ist die Nutzung von Model und View gemeinsam.

MV*-familiy Muster in Webapplikationen:

- Model View Controller
- Model View Presenter
- Model View ViewModel

MV* Muster I - Model View Controller



Model

- enthält die Daten, z.T. auch Geschäftslogik
- unabhängig von view und controller
- Bekanntgabe von Änderungen im Modell mit observer-pattern vom model zur view

View

- Stellt Daten aus dem Modell dar und nimmt Benutzerinteraktionen entgegen
- kennt Controller und Modell (gestrichelte Linien sind Assoziationen)
- wird über veränderte Daten vom observer-pattern informiert und ruft diese dann ab.

Controller

- verwaltet eine oder mehrere Präsentationen
- nimmt von view Benutzerinteraktionen entgegen, wertet diese aus und steuert entsprechend :
  - z.B. Ändern der View, Verschieben von Fenstern
  - Weiterleiten an das Model (z.B. Benutzereingaben)

MV* Muster I - Model View Controller

Keine Vorgaben für

- Platzierung der Geschäftslogik (teilweise im Controller, teilweise im Model)
- Eingabevallidierung (teilweise im View, teilweise im Model)

Verteilungsmuster

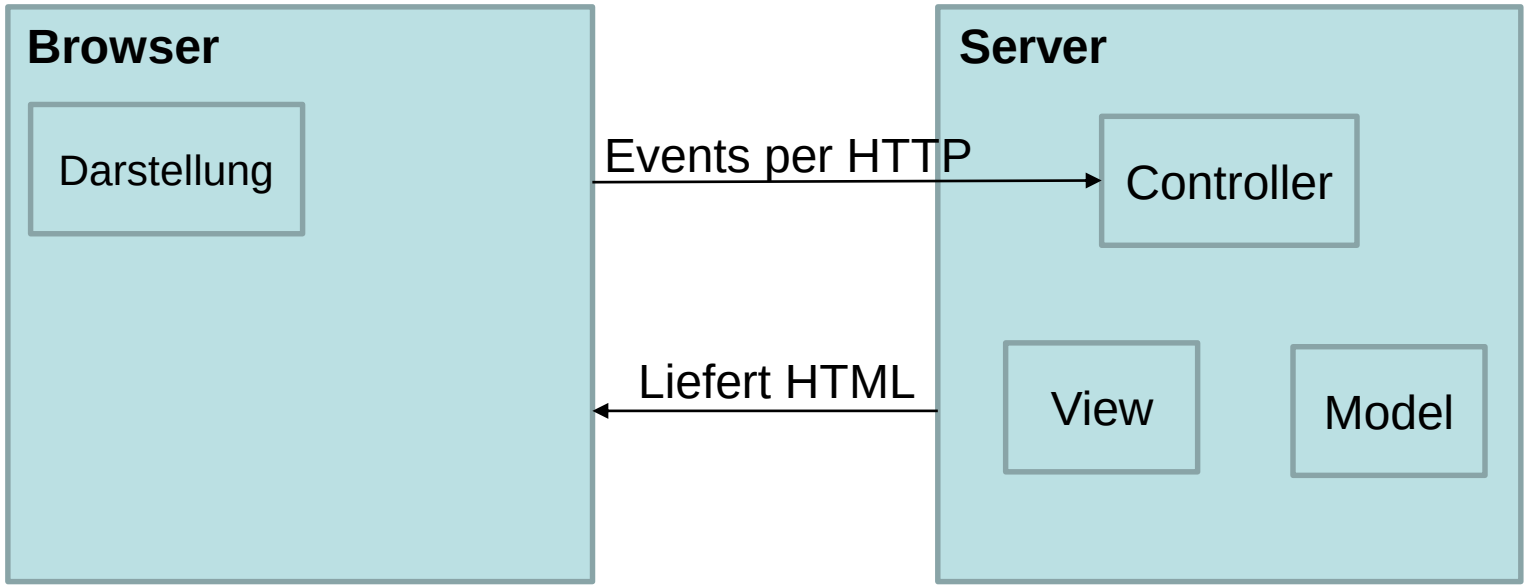
- Durch das im Web vorherrschende Client-Server Modell müssen auch die Komponenten des MVCs auf Client und Server aufgeteilt werden
- Der Browser übernimmt dabei einige Aufgaben des Views und des Controllers
  - Darstellung und Ereignisempfangen
- Die Server-Software übernimmt ebenfalls einige Aufgaben des Controllers
  - Umwandlung der Parameter in Variablen

Es sind verschiedene Aufteilungen möglich.

Besonderheiten im MVC im Webumfeld:

- Observer-Pattern ist nicht (leicht) umsetzbar
  - Durch den Request-Response-Zyklus von HTTP kann der Server keine Observer benachrichtigen
  - Client muss ständig nach Änderungen nachfragen
  - Neuere Entwicklungen mit Push-Nachrichten und Websockets
- Hyperlinks mit Controllereigenschaft
  - Hyperlinks legen das Ziel einer Anfrage fest, sind somit auch Controller

MV* Muster I - Model View Controller



Server zentriertes MVC

- Browser ist nur „Darstellungsgerät“
- Browser bekommt die vom Server vorgefertigte View zur Anzeige
- Browser leitet Events an den Controller auf der Server-Seite weiter
- Controller lädt Model, baut View, …

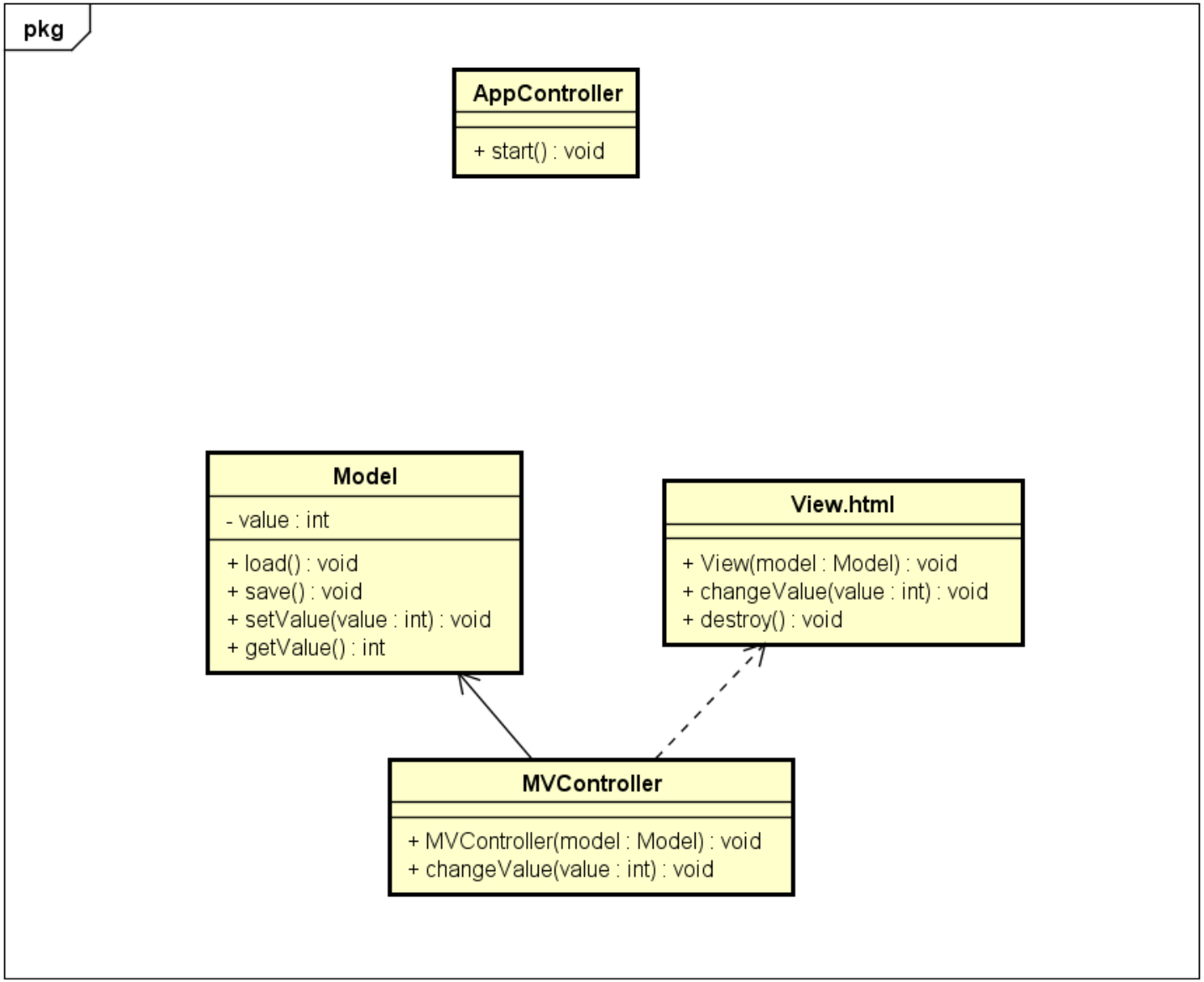
Vorteil:

- Sehr geringe Anforderungen an den Client

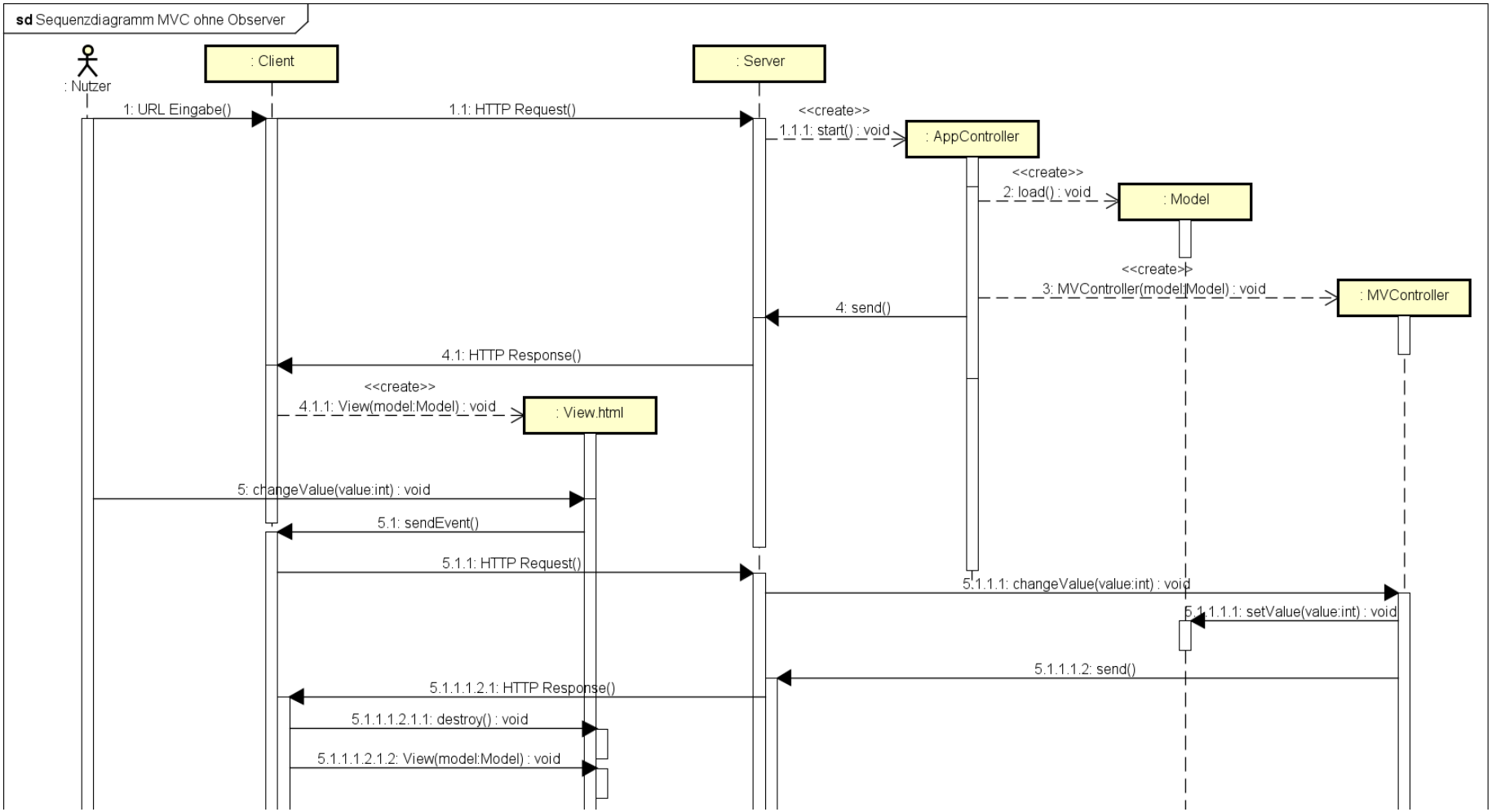
Nachteil:

- Client muss immer online sein
- Umsetzung des observer-Patterns nicht möglich

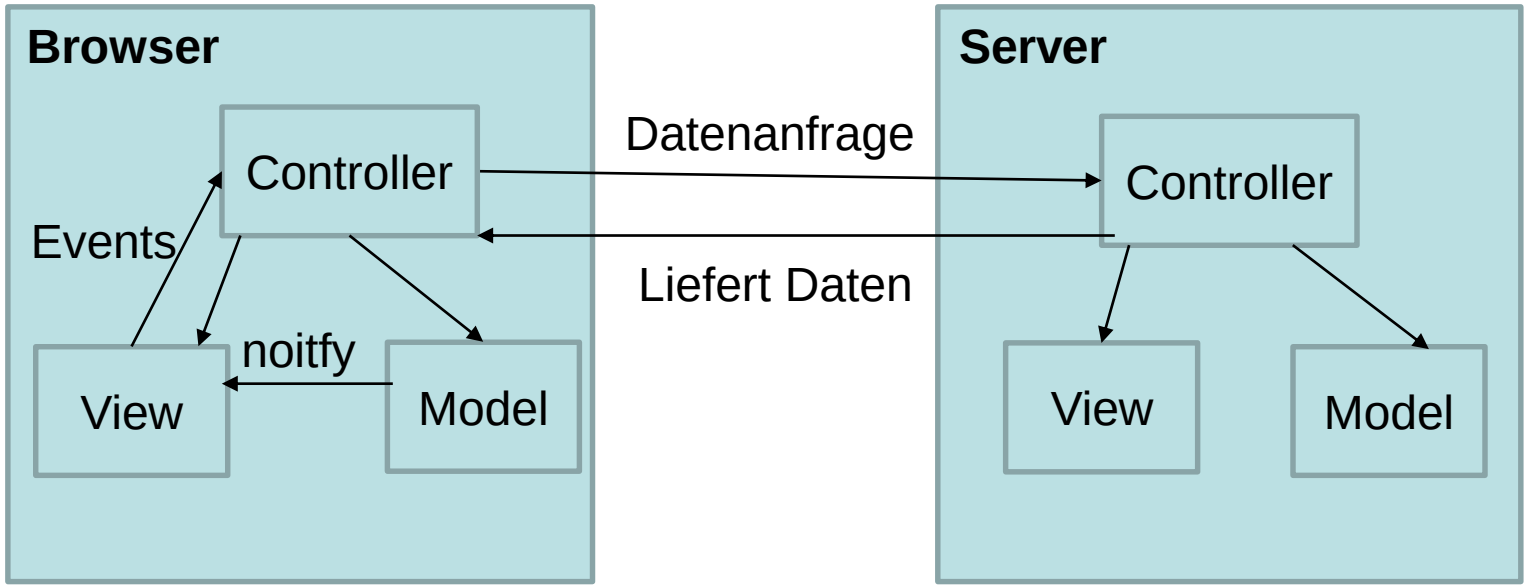
MV* Muster I - Model View Controller



MV* Muster I - Model View Controller



MV* Muster I - Model View Controller



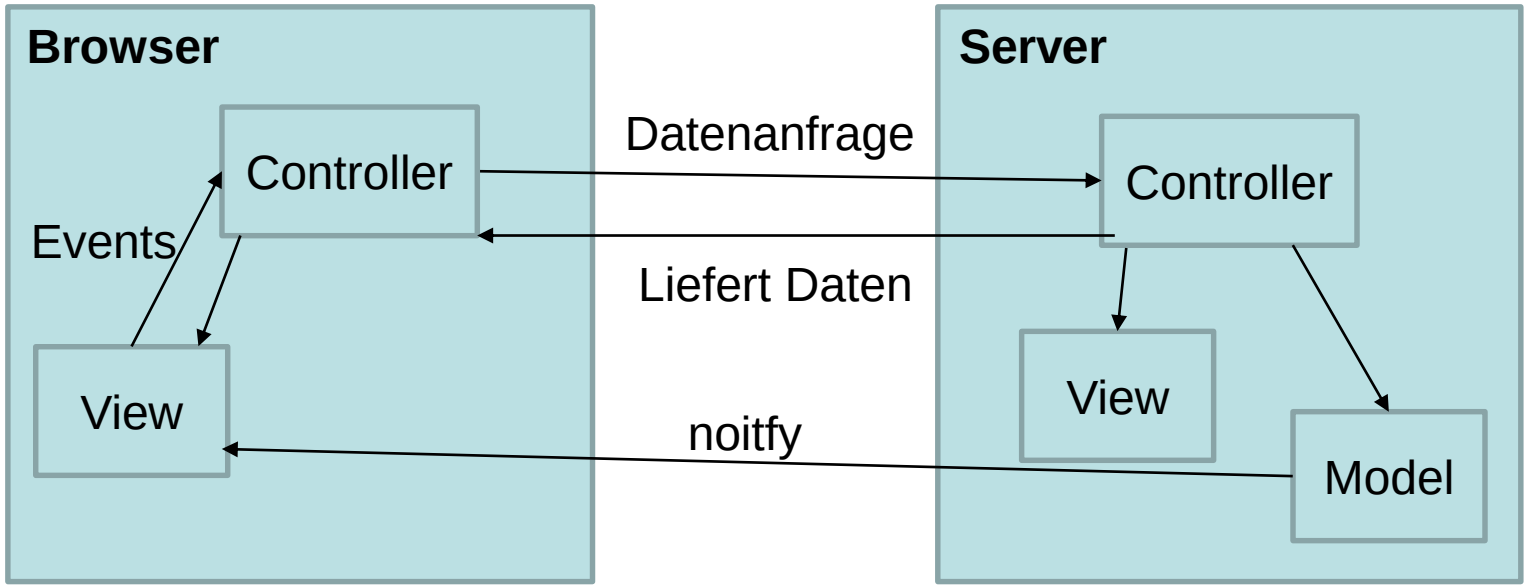
Client-Server aufgeteiltes MVC

- Einmal geladene Views werden im Browser manipuliert
- Client-Controller fragt Server nach Daten, Berechnungen und Seitenwechsel
- Server-Controller lädt und liefert Model und View und führt Geschäftslogik aus

Vorteil:

- Weniger Datentransfehr zwischen Client und Server
- Offline-Funktionalität möglich

MV* Muster I - Model View Controller



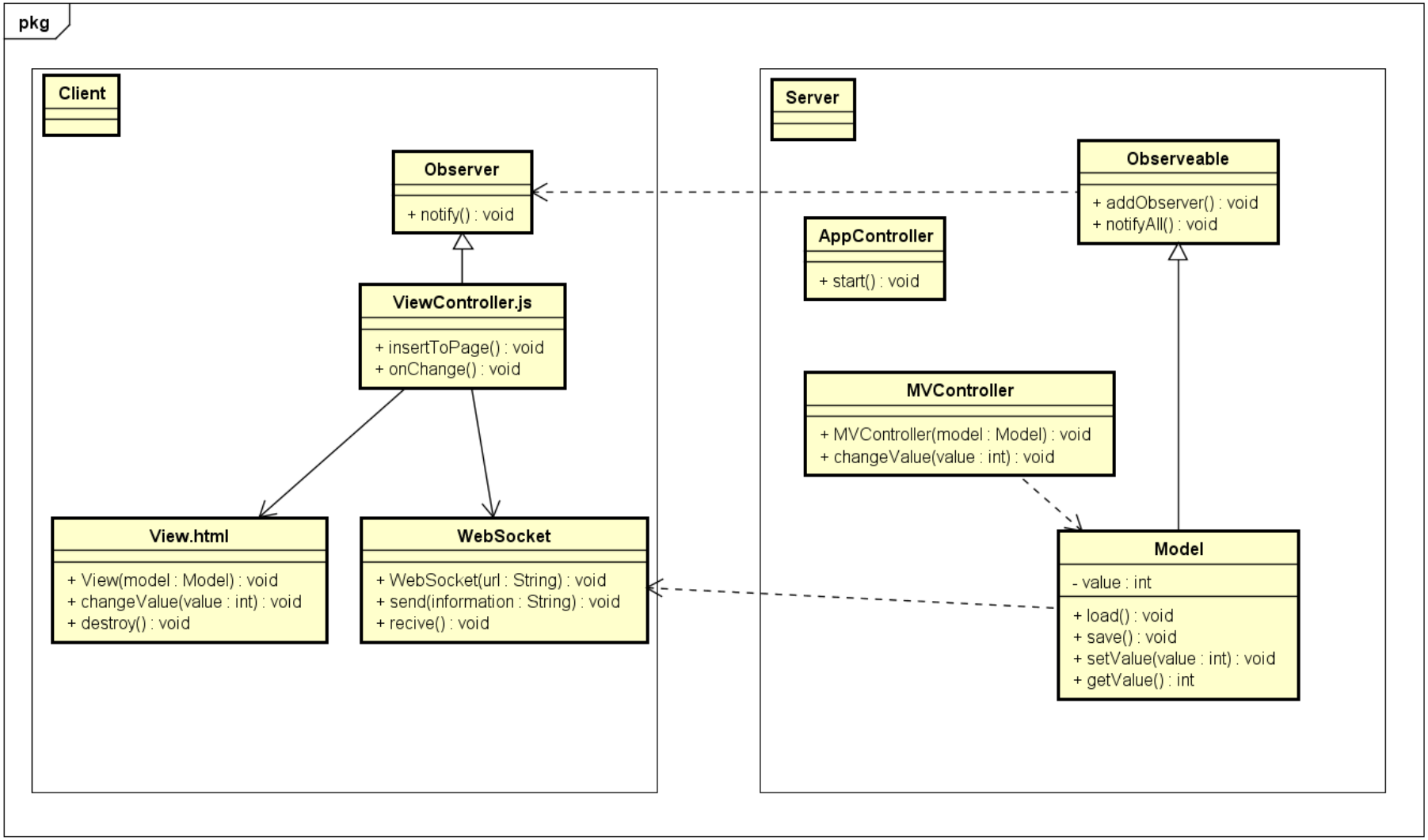
Volles Web-MVC mit WebSockets

- Einmal geladene Views werden im Browser manipuliert
- Client-Controller fragt Server nach Datenänderungen, Berechnungen und Seitenwechsel
- Server-Controller lädt und liefert Model und View und führt Geschäftslogik aus

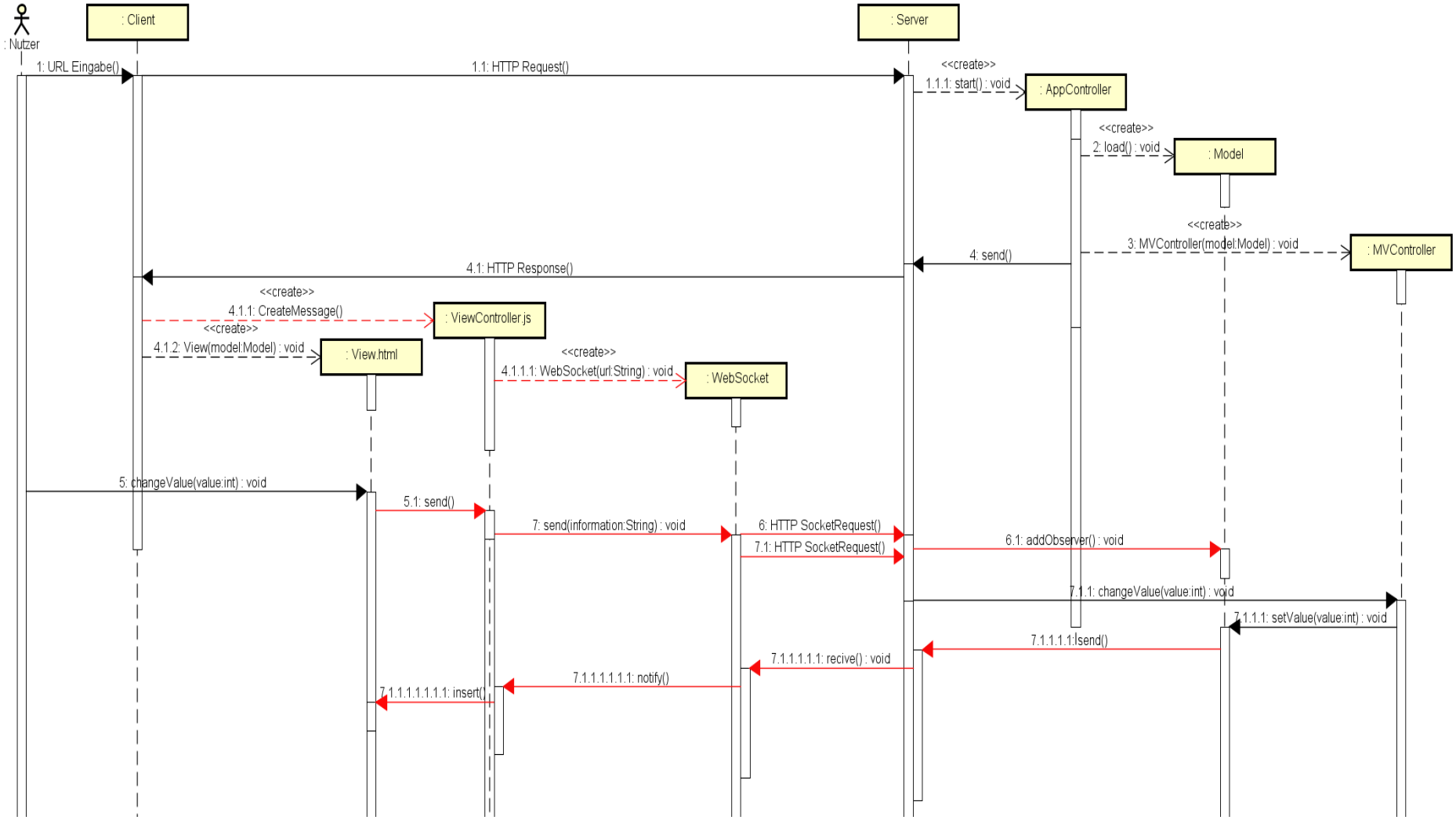
Vorteil:

- Weniger Datentransfehr zwischen Client und Server
- Observer-Pattern kann (mit WebSockets oder Push) umgesetzt werden

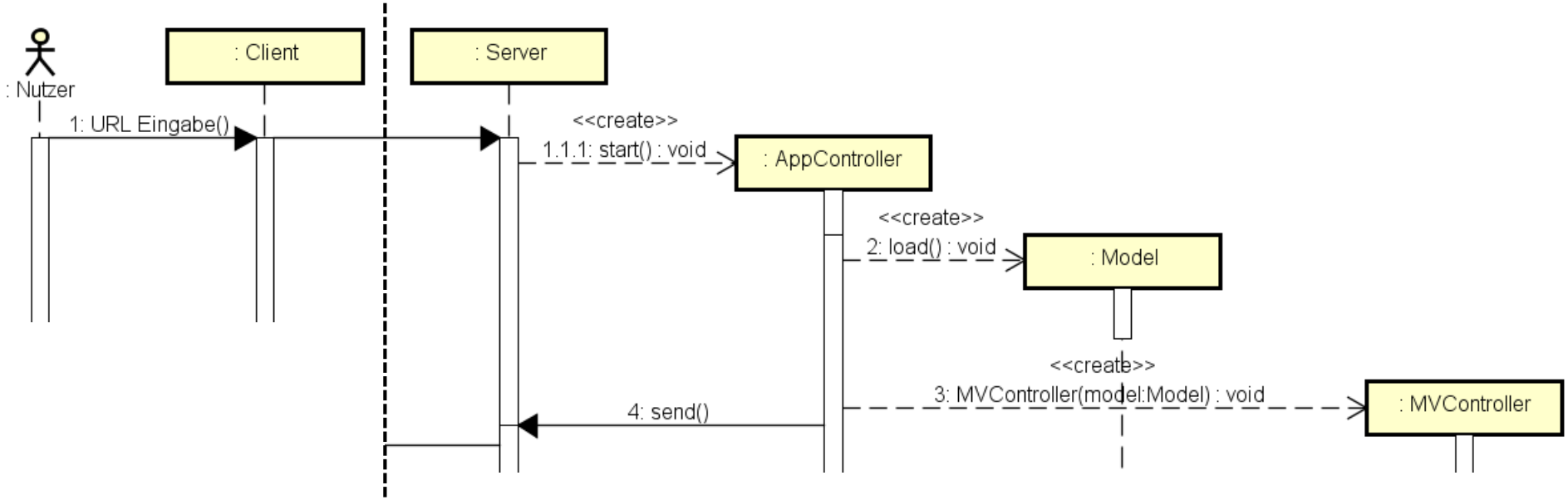
MV* Muster I - Model View Controller



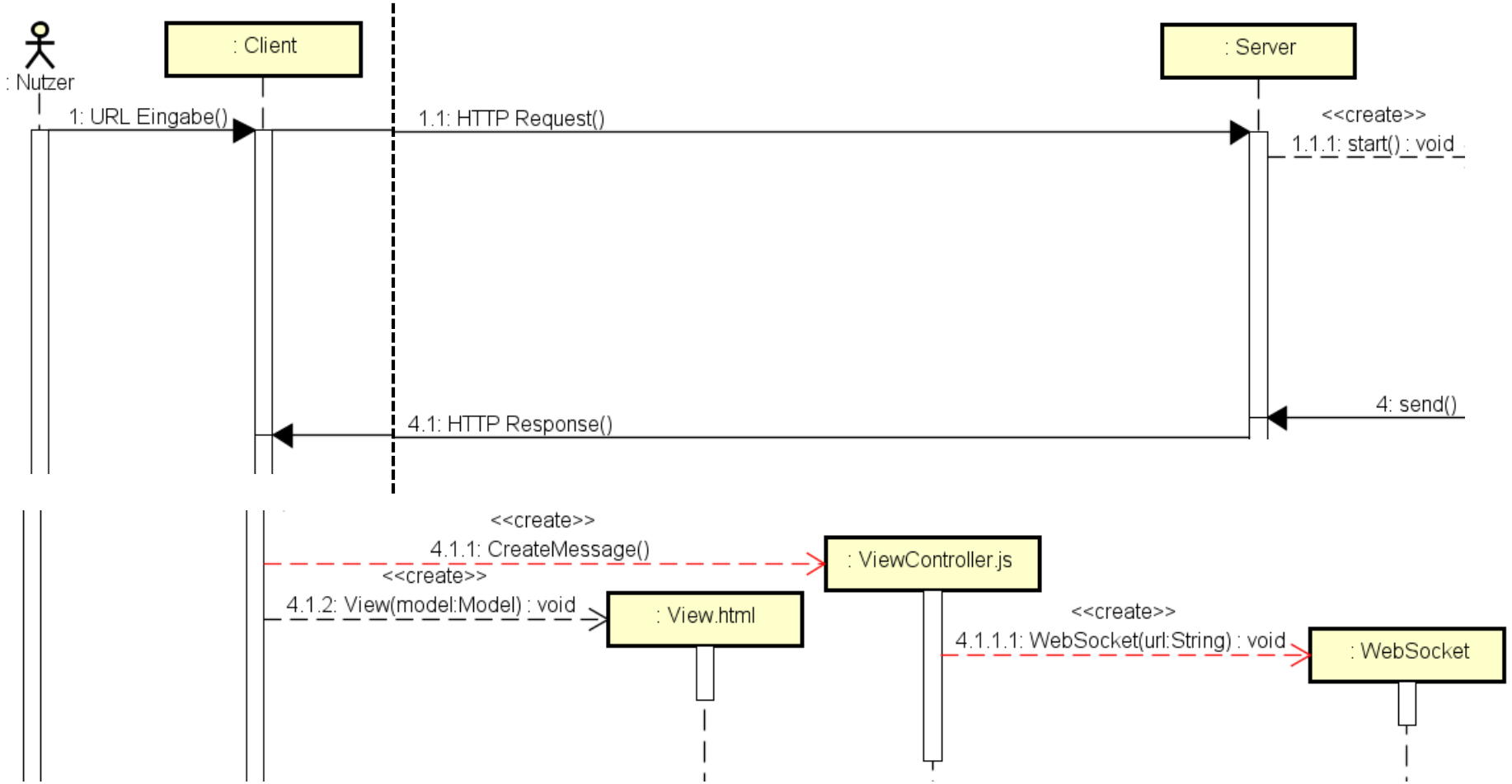
MV* Muster I - Model View Controller



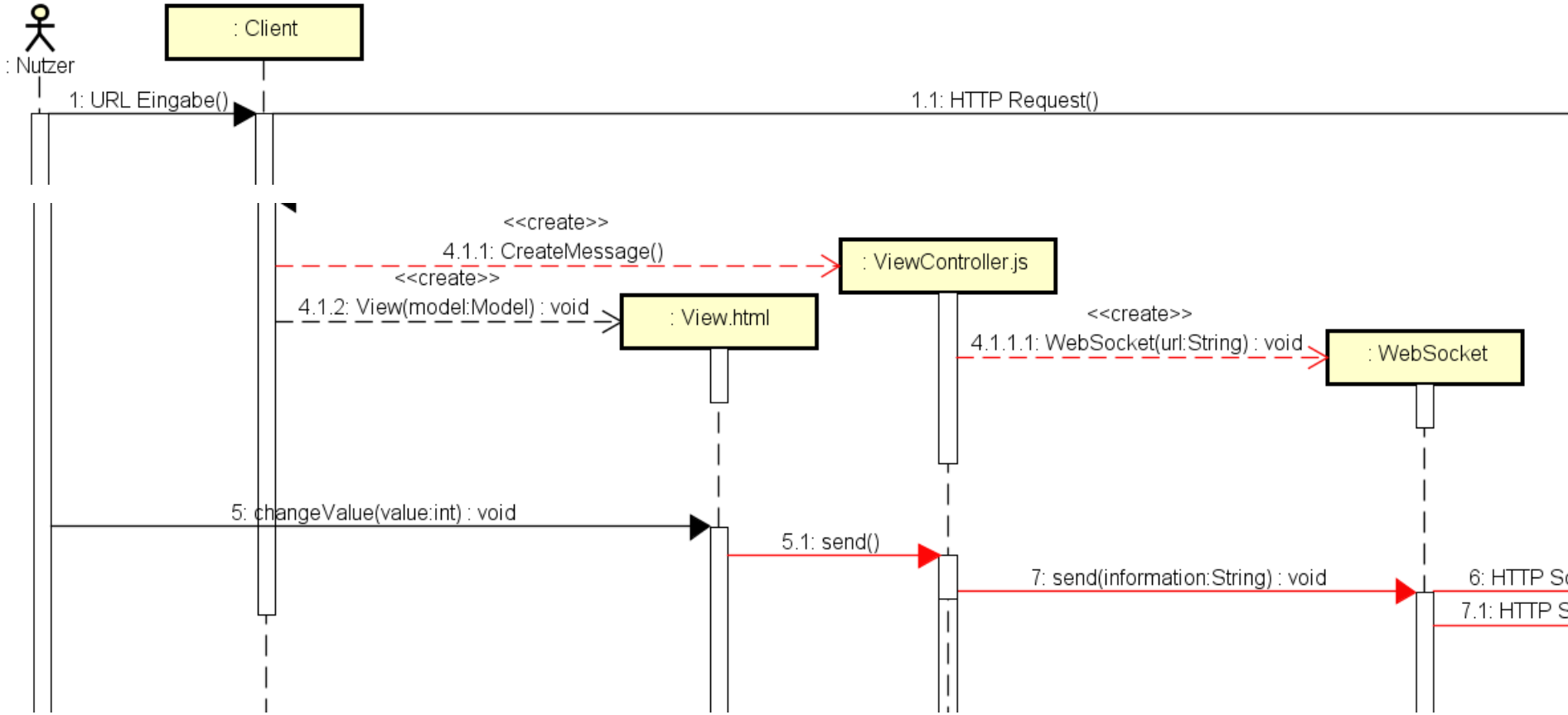
MV* Muster I - Model View Controller



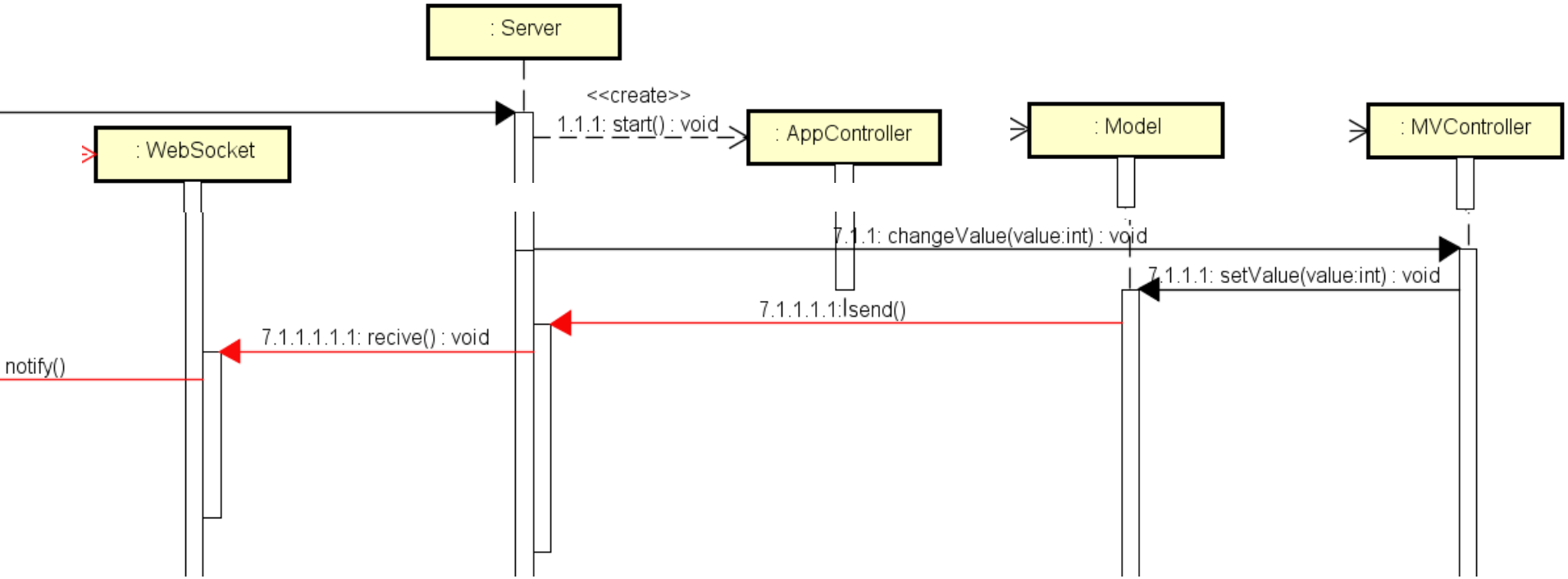
MV* Muster I - Model View Controller



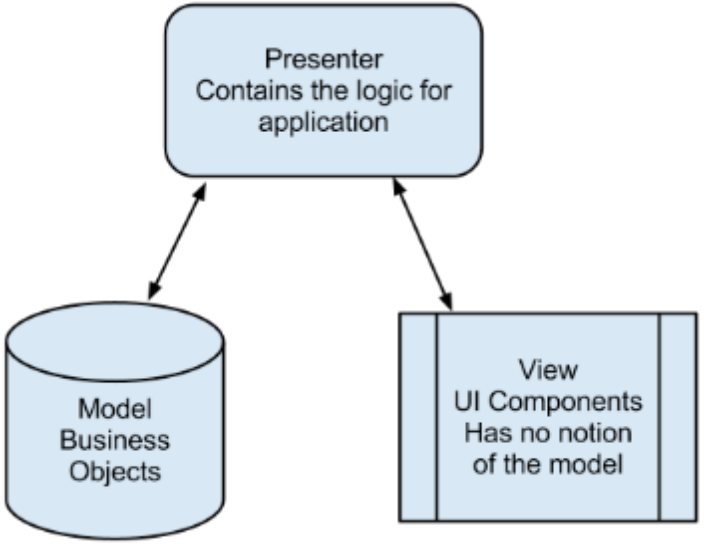
MV* Muster I - Model View Controller



MV* Muster I - Model View Controller



MV* Muster II - Model View Presenter



Model

- enthält die Logik der Ansicht , ggf auch Geschäftslogik
- enthält komplette Funktionalität zur Generierung der Ansicht
- wird nur vom Presenter gesteuert, kennt Presenter und Ansicht nicht

View

- Darstellung der Ansicht und Handling der Ein- und Ausgaben
- wird komplett vom Presenter gesteuert
- hat keinen Zugriff auf Funktionalität den Models und des Presenters

Presenter

- Bindeglied zwischen Model und View
- steuert die logischen Abläufe zwischen Model Model und View

MV* Muster II - Model View Presenter

Ziel: Trennung von Model und View

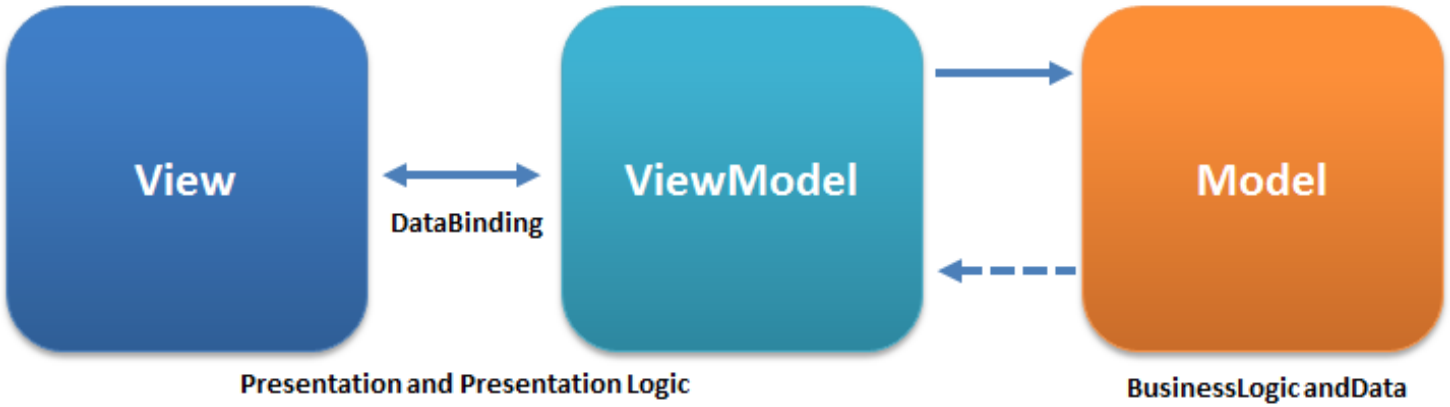
Eigenschaften

- Model und View definieren Schnittstellen
- Der Presenter verbindet diese Schnittstellen

Vorteile

- Stärkere Trennung von Model und View
- Vollständige Austauschbarkeit von Model und View
- Bessere Testbarkeit indem der View durch Tests ausgetauscht wird

MV* Muster III - Model View ViewModel



Model

- enthält die Daten und die Geschäftslogik
- benachrichtigt über Datenänderungen
- validiert die Benutzereingaben

View

- ***GUI-Elemente binden Eigenschaften des ViewModels***
- ist durch Datenbindung ***austauschbar und „schmal“ programmiert***

ViewModel

- enthält UI-Logik (Model der View)
- Bindeglied zwischen View und Model
- ruft Dienste und Methoden vom Model auf
- stellt der View öffentliche Eigenschaften und Befehle zur Verfügung mit Bindung an Steuerelemente (Eventhandler, oder Inhalte ausgeben)
- kennt die View nicht , ***austauschbar***

MV* Muster II - Model View ViewModel

Ziel: Trennung von Darstellung und Logik der Benutzerschnittstelle

Eigenschaften

- Erfordert einen Datenbindungsmechanismus
- Bidirektionale Anbindung von View und ViewModel
- Erfordert keine Controller-Instanzen
- Verwendet in modernen UI-Plattformen JavaFX, Silverlight, HTML5 und Windows Presentation Foundation

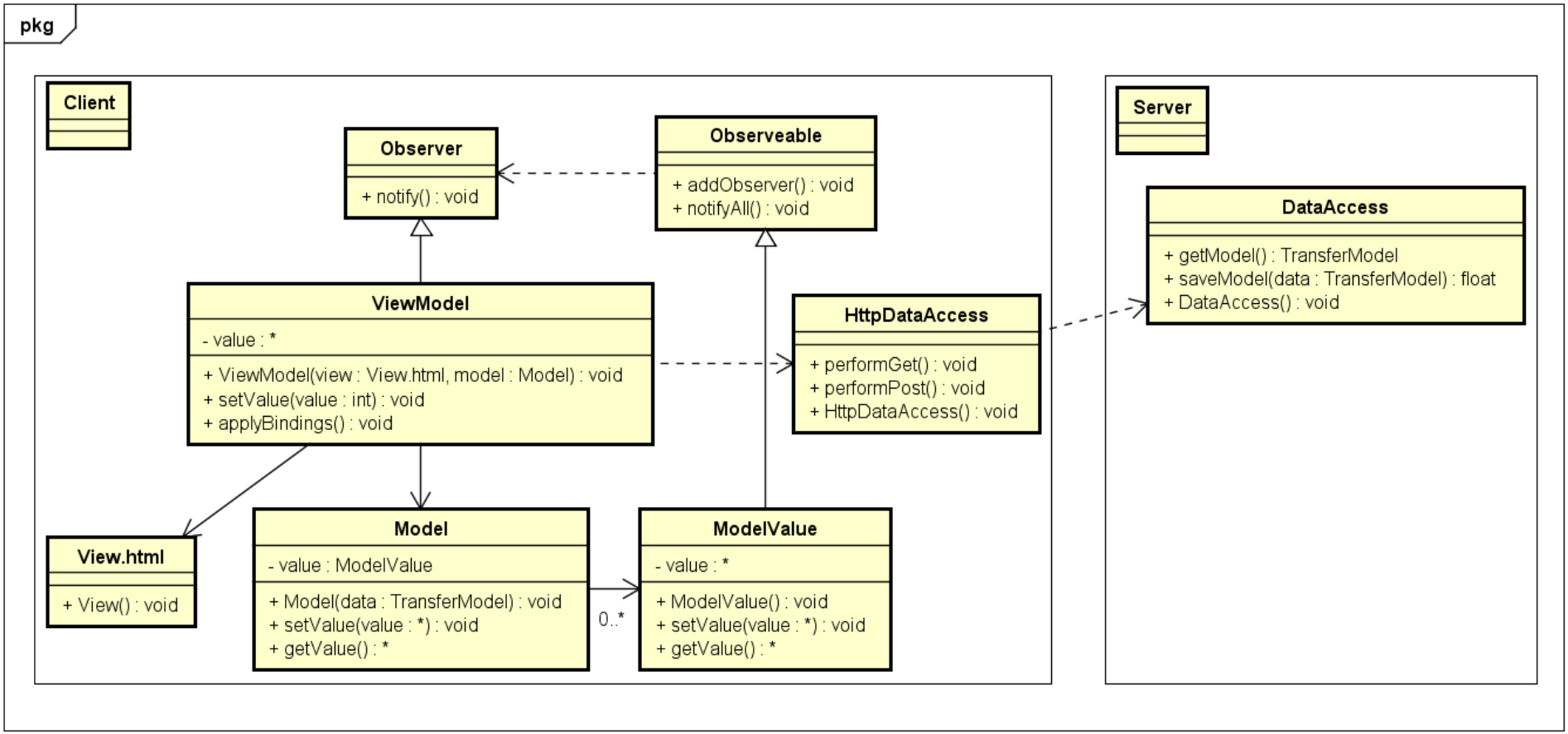
Vorteile

- Bessere Testbarkeit da der View keine Funktionalität enthält
- Weniger Implementierungsaufwand durch Verzicht auf Controll
- Stärkere Trennung von UI und Funktionen

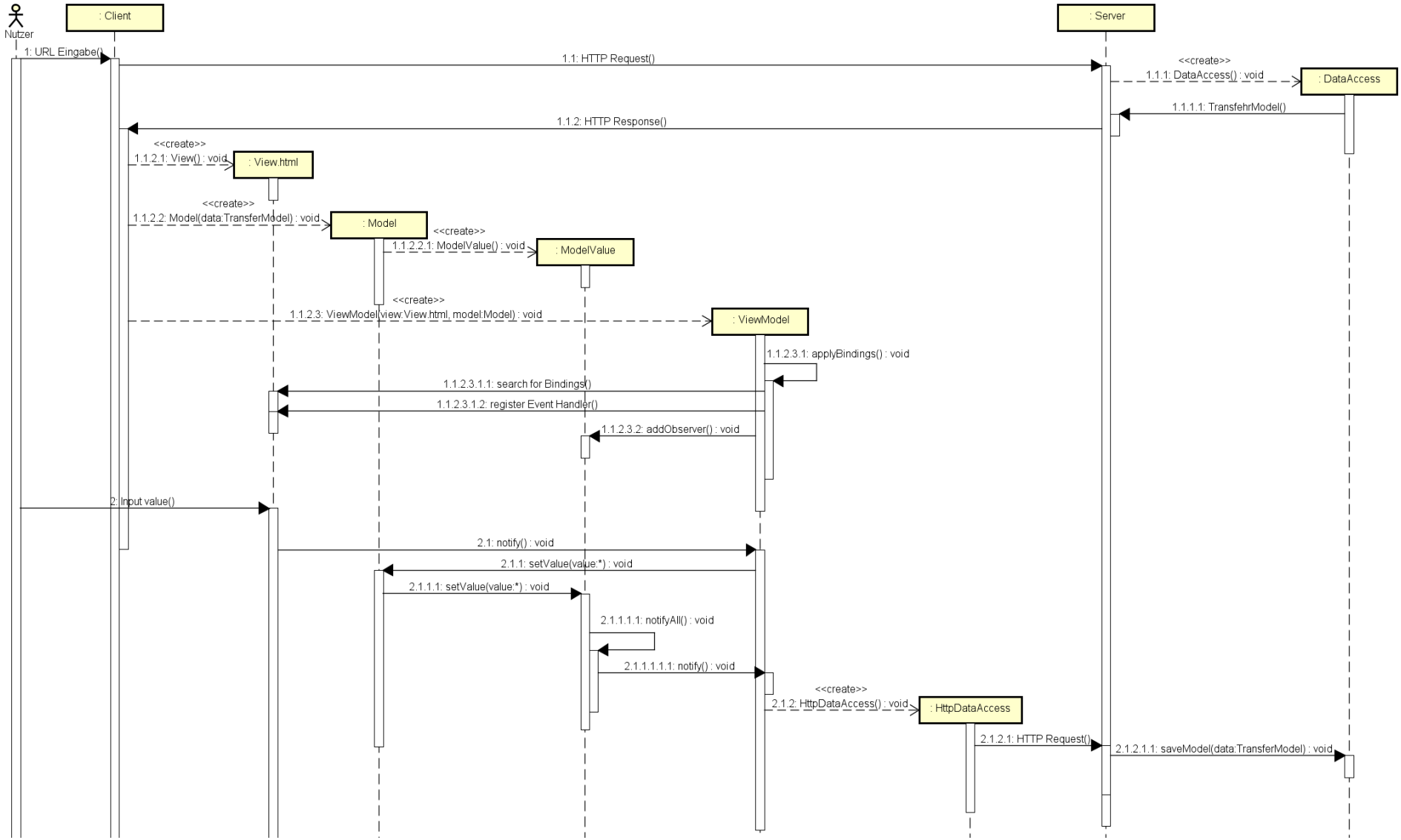
Nachteile

- Höherer Rechenaufwand durch die bidirektionale Anbindung
- Datenbindungsmechanismus ist zwingend erforderlich

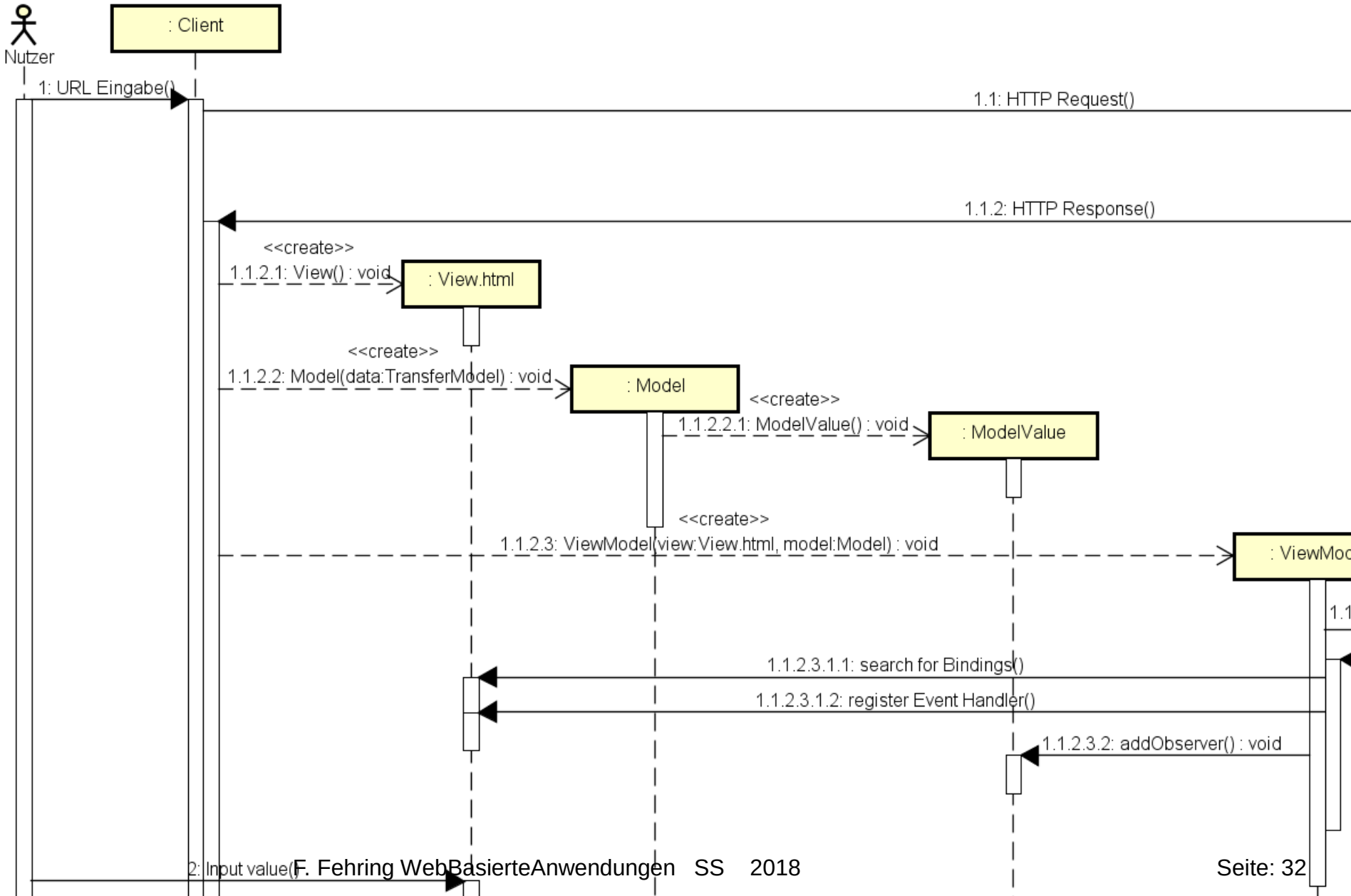
MV* Muster II - Model View ViewModel



MV* Muster II - Model View ViewModel



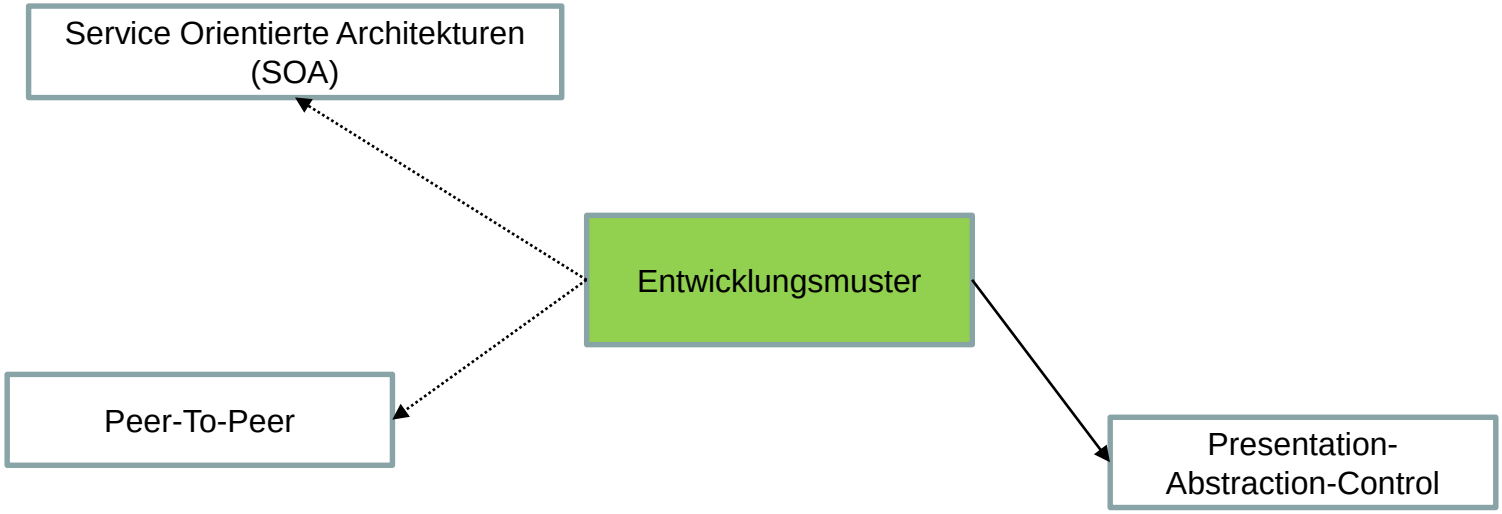
MV* Muster II - Model View ViewModel



Entwicklungsmuster

1. Kontext und Motivation
2. Kategorien
3. Model 1 und Model 2 Muster
4. MV*-Muster
5. **Darüber hinaus**
6. Projekt

Darüber hinaus



Links:

Apache HTTP-Server - http://httpd.apache.org/
PHP - http://php.net/manual/de/intro-whatis.php
Streaming - https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Live_streaming_w

Webanwendungen und Sicherheit

Definition:

Sicherheitslücken in Browserimplementierungen können durch JavaScript-Programme ausgenutzt werden z.B.:

- unbemerktes Versenden von Emails
- Auslesen des Browserverlaufs
- Live-Verfolgungen von Internetsitzungen
- Erraten von EBAY-Passwörtern

- Anwender deaktivieren daher manchmal das „Ausführen von JavaScript-Code“ im Browser

- JavaScript-Anwendungen laufen im Browser: Sandbox (abgeriegelte Umgebung ohne Zugriff auf Dateien, Benutzerdaten, BS,..)

Serverseitige Anwendungen

1. Kontext und Motivation
2. Webserver Interfaces
3. Servlets
4. JSP
5. Darüber hinaus
6. **Projekt**

Projekt

Übungsaufgabe:

Analyse der bisher erstellen Webapplikation auf die Verwendung und Anwendbarkeit Von Entwicklungsmustern.

Literatur: Internet und Netzwerke

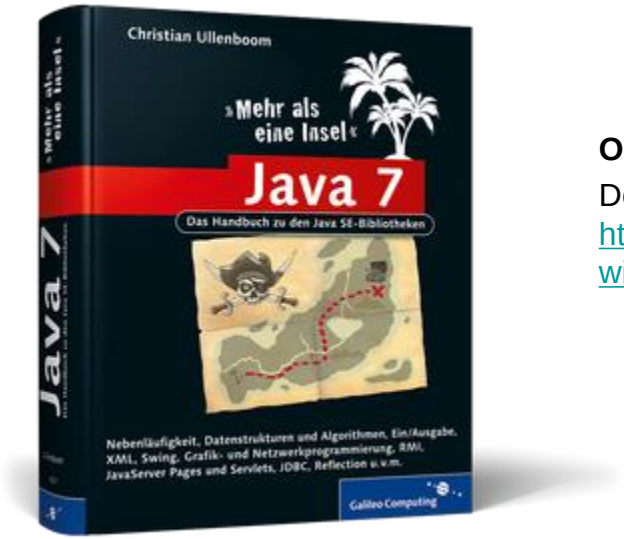
Melzer, Ingo et al. „Service-orientierte Architekturen mit Web Services“ Konzepte – Standards – Praxis 4. Auflage 2010, 381 Seiten, ISBN 978-3-8274-2549-2, Spektrum Akademischer Verlag über Springer Link

Christian Ullenboom: „Java 7 – Mehr als eine Insel

Das Handbuch zu den Java SE-Bibliotheken“

ISBN 978-3-8362-1507-7,

Rheinwerk Verlag 2012



Online-Quellen:

Dokumentation zu Jquery: https://learn.jquery.com/ajax/working-with-jsonp/

Kappel, Gerti & Pröll, Birgit & Reich, Siegfried & Retschitzegger, Werner. (2003). Web Engineering - Die Disziplin zur systematischen Entwicklung von Web-Anwendungen. .