MVVM

Prof. Dr.-Ing. Andreas Heil

v1.1.1

Lernziele

- **Unterschied** zwischen Entwurfsmuster und Architekturmuster **verstehen**
- **Grundlagen** des MVVM-Musters **kennen und anwenden lernen**
- Ein **MVVM-Framework** am Beispiel von Knockout.js **kennen lernen**

Typische Web-Anwendungen

Die meisten Anwendungen sind nach einer Schichtenarchitektur aufgebaut:

- Die Benutzungsschnittstelle (View)
- Daten, die angezeigt und manipuliert werden (Data)
- Die Anwendungslogik, die das Verhalten der Anwendung ausmacht (Logic)

Entwurfsmuster versuchen

- die Schichten zu entkoppeln
- die Anwendung möglichst flexibel zu gestalten

Kurze Wiederholung

Welche Entwurfsmuster sind Ihnen bekannt?

- ...

Welche Probleme lösen Entwurfsmuster?

- ...

Wo finden Sie Informationen über Entwurfsmuster?

- ...

Exkurs: Architekturmuster

- MVC und MVP (s.u.) werden teilweise als Entwurfsmuster angesehen
- MVVM ist kein Entwurfsmuster sonder ein *Architekturmuster* (gleich mehr dazu)
- Architekturmuster beschreiben die **Struktur bzw. Organisation einer Anwendung** und die **Interaktion der einzelnen Komponenten**
- Entwurfsmuster hingegen beschreiben ein **Teilproblem einer Software**

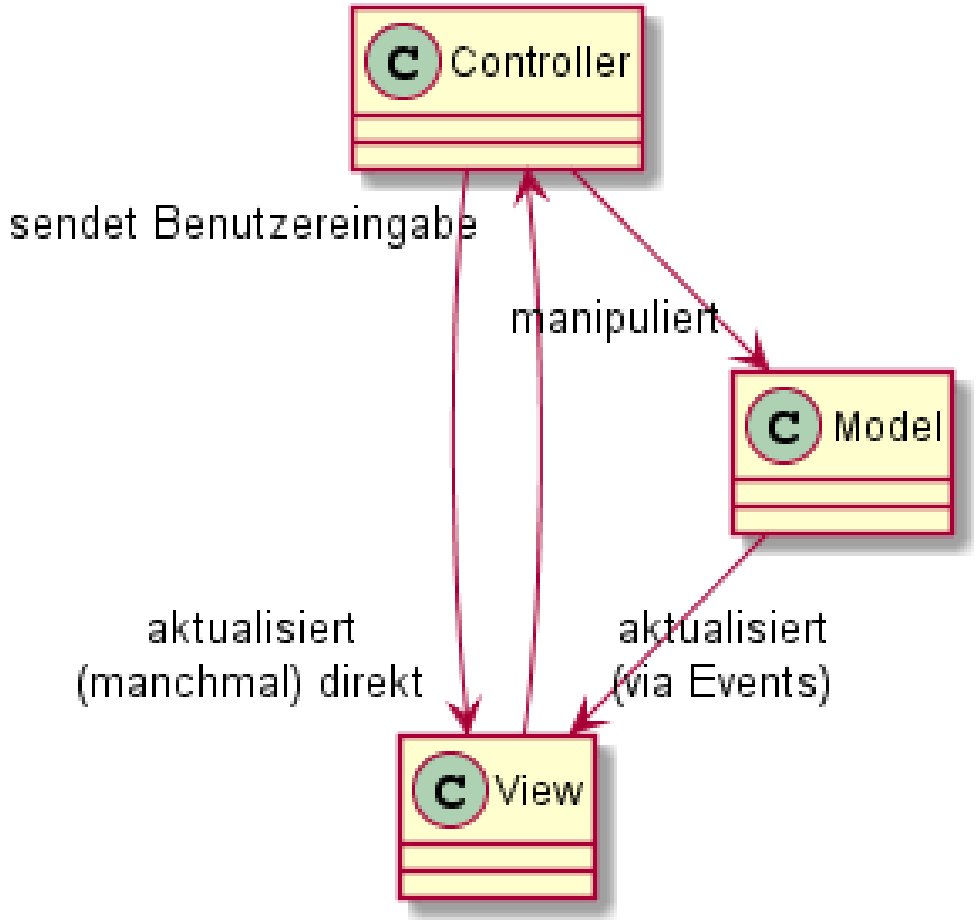
Beispiel: Das Entwurfsmuster, das wir für die Aktualisierung anderer Objekte (z.B. einem View) verwenden, ist das Observer Pattern (dt. Beobachtermuster).

Model-View-Controller (MVC)

- Entstand bereits in den 1980ern in den Anwendungen von Xerox (basierend auf Smalltalk)
- Ein sog. Controller verbindet ein View und das darunterliegende Model
- Der View nutzt das Model um die Ausgabe zu erzeugen
- Das Model enthält die Informationen
- Ein Model kann Ereignisse (engl. Events) versenden, wenn sie Eigenschaften des Models ändern
- Die Events können sowohl vom Controller aber auch vom View genutzt werden

Model-View-Controller

- Unterschiedliche Aspekte de Anwendung werden getrennt (Separation of Concerns)
- Implementierungen weichen voneinander ab



Exkurs: Separation of Concerns - Vorteile

- Stammt vermutlich von E.W. Dijkstra
- Wird für Entkopplung benötigt
- Es kann mehrere Darstellungen geben (Desktop, Web, Mobil)
- Unterschiedliche Entwickler könnten sich um unterschiedliche Teile kümmern

Dijkstra, E.W., On the Role of Scientific Thought, Selected writings on Computing: A Personal Perspective., Springer, ISBN 0-387-90652-5.

Exkurs: Weitere Entwurfsprinzipien

SOLID

- Single Responsibility Principle
  - Eine Klasse sollte nur *eine* Aufgabe besitzen
- Open/Closed Principle
  - Klassen sollen *offen für Erweiterungen*, *geschlossen für Veränderungen* sein
- Liskov Substitution Principle
  - Eine *Elternklasse* sollte durch eine Kindklasse *ersetzbar* sein, ohne dass das System dadurch ein Fehler verursacht

Exkurs: Weitere Entwurfsprinzipien

SOLID (Forts.)

- Interface Segregation Principle
  - Aufrufer sollte nicht von Methoden abhängen, die nicht genutzt werden, d.h. zu große Schnittstellen/Klassen trennen
- Dependency Inversion Principle
  - Übergeordnete Module sollten nicht von Funktionen untergeordneter Modulen abhängen
  - Besser: Alle Klassen (über-/untergeordnet) erben das gleiche Interface, von denen Sie gemeinsam abhängen

Model

- Enthält die Daten
- Regelt den Zugriff und wann Änderungen stattfinden
- Basiert meist auf Objekten der realen Welt

View

- Darstellung des Inhaltes eines Models
- Zugriff auf die Daten (nur) durch das Model
- Darstellung des Models obliegt vollständig dem View

Controller

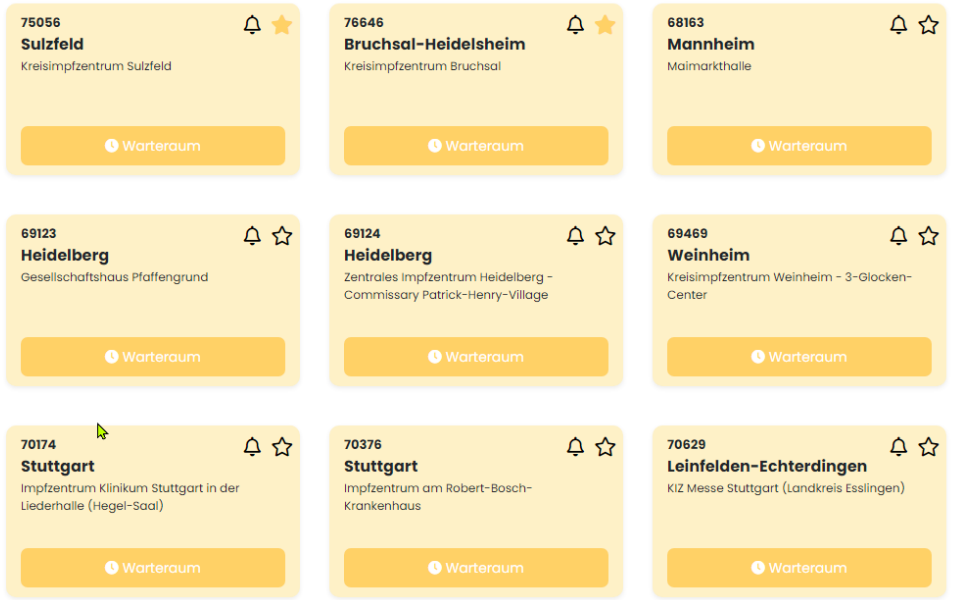
- Übersetzt Interaktionen mit dem View in entsprechende Aktionen
- Ausführung der Aktionen durch das Model
- Desktop-Anwendung: Maus-Klick; Web-Anwendung: HTTP-Request
- Aktionen können sowohl Geschäftsprozesse auslösen aber zu Statusänderungen im Model führen (vgl. HATEOS)
- Abhängig von Benutzerinterkation und dem Ergebnis der Aktionen des Models stellt der Controller einen passenden View dar

Beispiel: Inversion of Control (IoC)

- Entwurfs-Prinzip auch bekannt als Hollywood-Prinzip - "Don't call us, we call you"
- Sowohl auf Code-Ebene
  - Z.B. realisierbar über Observer Pattern
  - Übergabe von
- Als auch auf Anwendungsebene
  - Z.B. über Callbacks (z.B: in SOAP, REST Callbacks[^4], Messaging)

Praxisbeispiel

- Web-Anwendung zur COVID-19 Impfterminvergabe 2021 in BaWü
- Keine Wartelisten sondern aktive Abfrage nach Terminen
- Kontrolle obliegt demnach beim Anwender (F5)
- Führte zur regelmäßigen Überlastung der Server
- Ideen zur Realisierung nach IoC?



Quelle: https://116117.app/

Web Anwendungen

Wie ist MVC in Web Anwendungen umgesetzt?

- View und Controller sind hier auf Client und Server verteilt
- Serverseitig wird ein sog. *Router* benötigt, um Anfragen auf den entsprechenden Controller weiterzuleiten
- Im Controller wird dann die entsprechende *Action* ausgeführt
- Abhängig vom Request wird dann das Model aktualisiert (vgl. HATEOS)
- Das Ergebnis wird in Form eines Views (in einer Web Anwendung i.d.R. HTML) angezeigt

Anwendung

- MVC ist (war) in Web Frameworks weit verbreitet
- MVC Frameworks:
  - ASP.NET MVC (.NET)
  - Rails (Ruby)
  - Spring (Java)
  - AngularJS (JS)
  - CakePHP (PHP)

Exkurs: Spring != Spring Boot

- Spring != Spring Boot!
- Spring Boot setzt auf Spring auf
- Im "Sprachgebrauch" oft und gerne nicht getrennt oder vermischt

Spring

> ...ein umfassendes Programmier- und Konfigurationsmodell für moderne Java-basierte Unternehmensanwendungen – auf allen möglichen Deployment-Plattformen. […]

- Sammlung von Tools für die Anwendungsentwicklung^2
  - Dependency Injection
  - Events
  - Ressourcen-Verwaltung
  - I18n
  - Datenzugriff
  - Transaction-Handling
  - u.v.m.

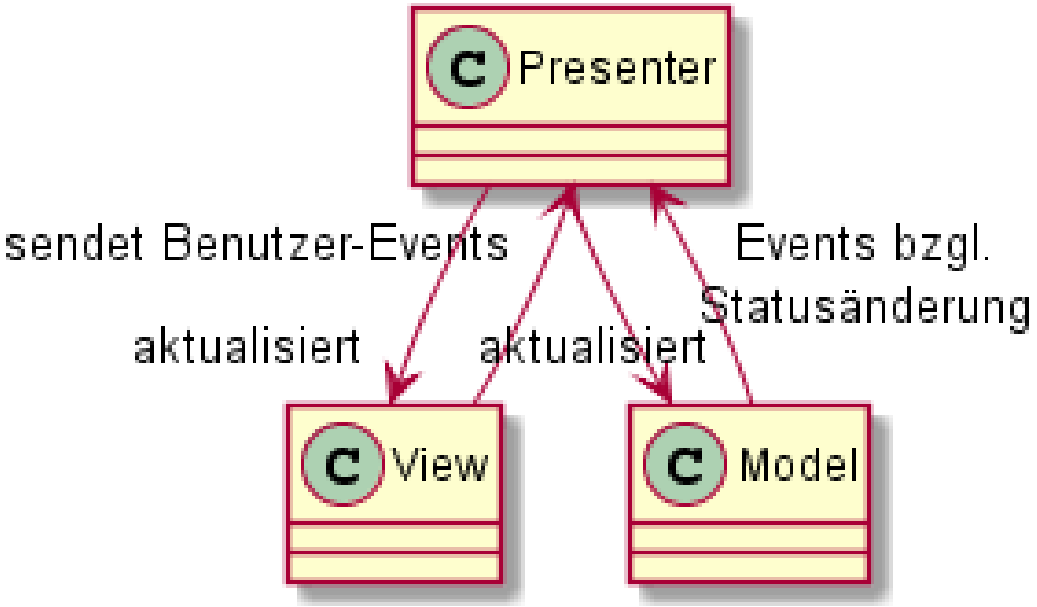
Spring Boot

- Vorkonfiguration der Elemente aus Spring (u.a.)
  - Konfiguration der Spring-Beans über Spring Boot Properties^3
  - Beispiel: Spring Boot Web enthält "automagisch" einen Tomcat-Container auf Port 8080 ohne Konfiguration
  - ...
- Schnelle Erfolge bei Nutzung von Standardkonfigurationen
- Bei Abweichungen von Standards ist tiefergreifendes Wissen erforderlich
- Tipp: Möglichst bei den Standards bleiben

Model-View-Presenter (1)

- Im Gegensatz zu MVC liegt die Präsentationslogik beim Presenter
- Das Model ist ein Interface, in dem die Daten definiert werden, die angezeigt werden
- Ein View ist ein passives Interface, das die Daten (d.h. das Model) anzeigt und Commandos (d.h. Events) an den Presenter routed, dass dieser etwas mit den Daten »macht«
- Der Presenter agiert sowohl für das Model aus auch den View
- Der Presenter bereitet die Daten auf, um Sie im View anzuzeigen
- Ein Presenter pro View

Model-View-Presenter (2)



Einige Anmerkungen

- MVP unterstützt »echte« Zweiwegekommunikation mit dem View
- Jeder View implementiert irgend eine Art von View-Schnittstelle
- Im View wird eine Instanz des Presenters referenziert
- Events werden vom View an den Presenter weitergeleitet
- Der View gibt *niemals* UI-bezogenen Code (z.B. Controls) an den Presenter weiter

Model

- Kommuniziert mit Datenbankschicht
- Feuert Events wenn Daten generiert/erzeugt werden

View

- Rendert die Daten
- Empfängt die Events und repräsentiert die Daten
- Grundlegende Validierung (z.B. gültige E-Mail, PLZ etc.)

Presenter

- Entkoppelt einen konkreten View vom Model
- Unterstützt den View bei komplexen Abläufen
- Kommunikation mit dem Model
- Im Gegensatz zum View finden hier komplizierte Validierungen statt (z.B. Einbezug weiterer Daten)
- Frägt das Model ab (sendet Queries an das Model)
- Empfängt Daten vom Model, bereitet die Daten auf bzw. formatiert die Daten und sendet diese an die View
- MVP nutzt die gleichen Verfahren bzgl. Events wie MVC

Einige Hinweise

- MVP wird überwiegend in der Client-Entwicklung genutzt
  - Google Web Toolkit (Java)
- In der Praxis ist der Unterschied zu MVC verschwindend gering

Zwei Variante

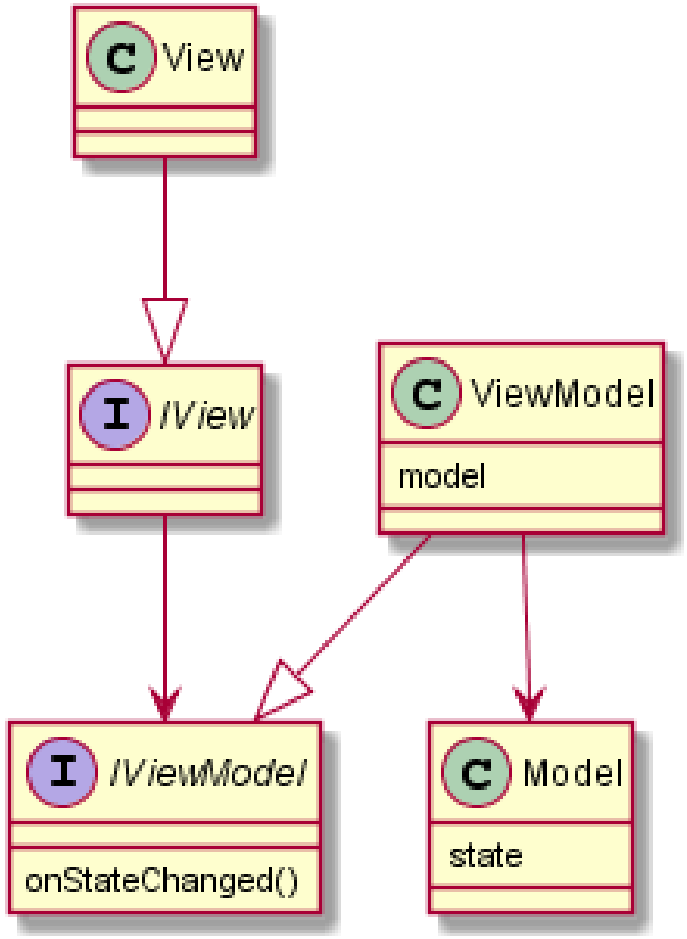
Es existieren zwei Varianten für MVP:

- Passiver View
  - Interaktionen werden ausschließlich vom Presenter bearbeitet
  - View wird ausschließlich vom Presenter aktualisiert
- Supervising Controller
  - View interagiert mit dem Model (via einfachem Binding)
  - View wird durch den Presenter via Data Binding aktualisiert

Model-View-ViewModel (MVVM)

- MVVM vereinigt die Vorteile von MVC (Separation of Concerns) und MVP (Data-Binding)
- Das *Model* ist identisch zu dem aus MVC
- Der *View* ist reine Repräsentation der Daten (analog zu MVC)
- Das *ViewModel* ist eine Abstraktion des Views, das speziell auf Data Binding ausgerichtet ist
- Die Rolle des Controllers aus MVC wird durch einen sog. *Binder* übernommen
- Aktualisierung der UI und des ViewModels (two-way)
- Binder sind Bestandteil des Frameworks und für den Entwickler i.d.R. transparent

MVVM


View
IView
ViewModel
model
IViewModel
onStateChanged()
Model
state

Einige Anmerkungen

- Erstmals im Microsoft WPF Framework
- Zweiwegekommunikation via Binding in XAML in WPF
- ViewModel repräsentiert den View in einer »darstellungsneutralen Form«
- Direktes Binding zwischen View und ViewModel
- Ein ViewModel pro View
- Wenn auch nicht gebräuchlich findet man manchmal Bezeichnung Model-View-Binder
- In WPF ist XAML der Binder

ViewModel

- Das ViewModel ist das »Model« des Views
- Vermittler (engl. mediator) zwischen dem View und dem Model
- Übernimmt die Funktion des Controllers in MVC
- Konvertiert die Daten aus dem Model für den View und umgekehrt
- Leitet Commands (Aktionen) an das Model weiter

Praxistipps

- Durch Implementierung von abstrakter Basisklassen lässt sich sehr viel Redundanz in den Klassen einsparen
- Speziell in WPF> Kommandos erhalten ebenfalls in Interface: `ICommand`
- Commands enthalten eine `execute` und `canExecute` Methode
- Ob ein Kommando ausgeführt werden kann (UI-Element aktiv) hängt vom Status der `canExecute` Methode ab

Beispiel Framework: Knockout.js

- OSS (MIT Lizenz)
- JavaScript, daher mit allen Frameworks einsetzbar
- Keine weiteren Abhängigkeiten (z.B: Bootstrap o.ä.)

Web: https://knockoutjs.com/

Tutorial: http://learn.knockoutjs.com/

Acknowledgments

Die Folien basieren auf einer Vorlesung von Florian Rappl^1.

Dank an Clemens Vasters für den konstruktiven Beitrag zu IoC

Referenzen