

Observer Pattern

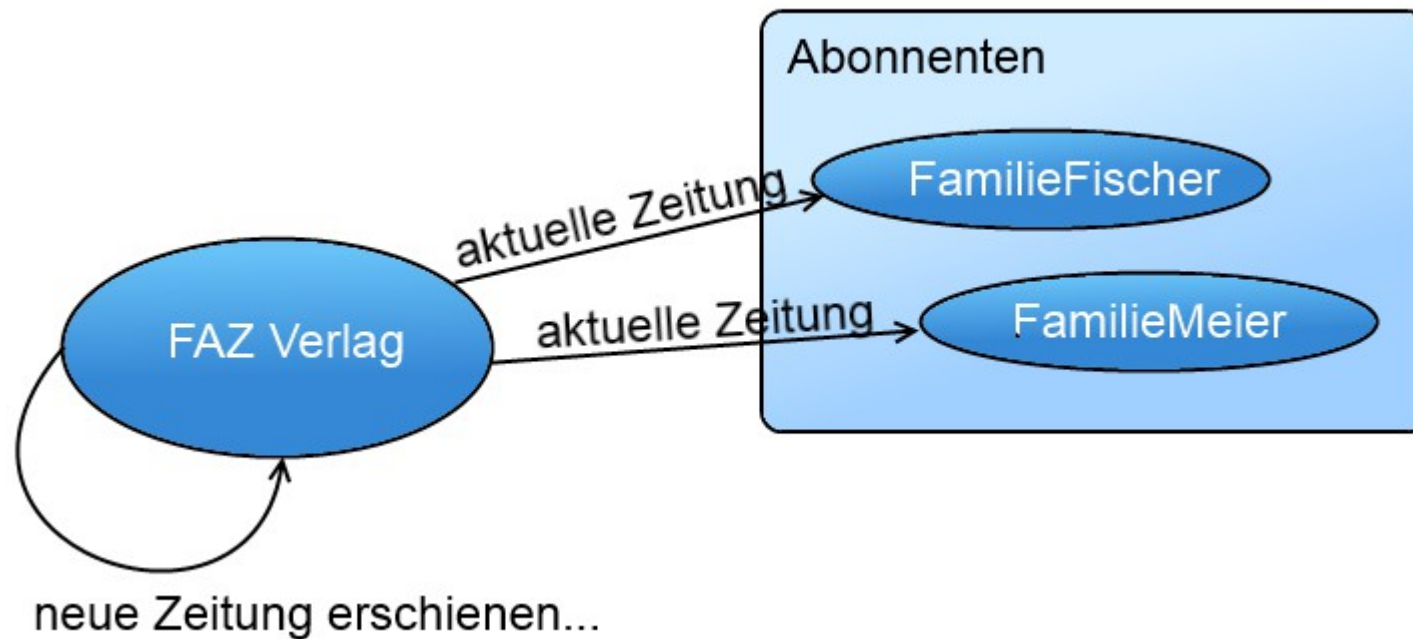


Observer Pattern

Agenda

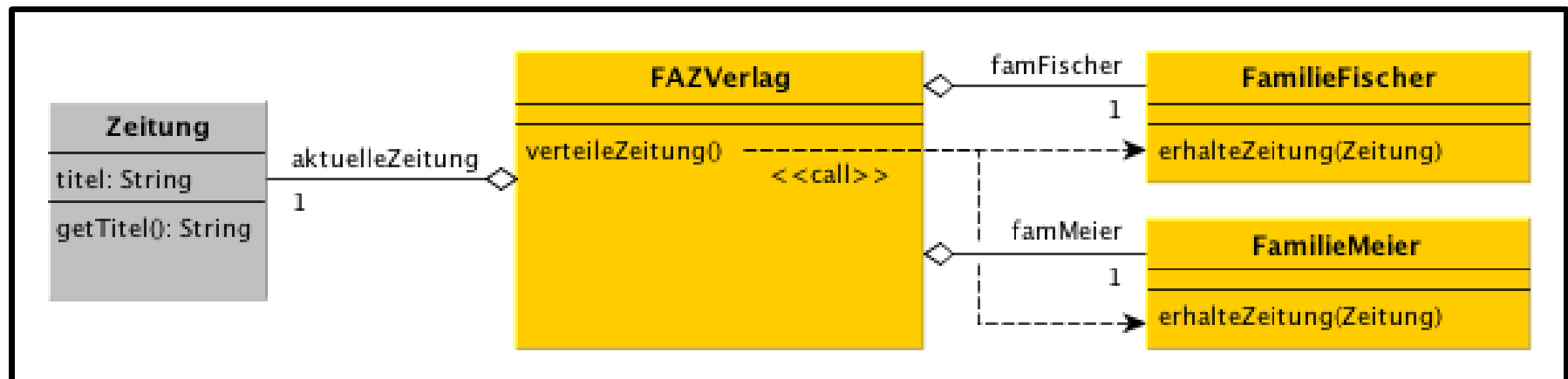
- Problem
- Lösung: Observer Pattern
- Variationen
- Übung/Beispiel
- Vor- und Nachteile

Observer Pattern: Problem



Observer Pattern: Problem

Klassendiagramm



Observer Pattern: Problem

Simpler Java-Code

```
public class FAZVerlag {  
  
    private Zeitung aktuelleZeitung;  
    private FamilieFischer famFischer;  
    private FamilieMeier famMeier;  
  
    //Sobald eine neue Ausgabe existiert  
    public void verteileZeitung() {  
        famFischer.erhalteZeitung(aktuelleZeitung);  
        famMeier.erhalteZeitung(aktuelleZeitung);  
    }  
}
```

Nachteile?

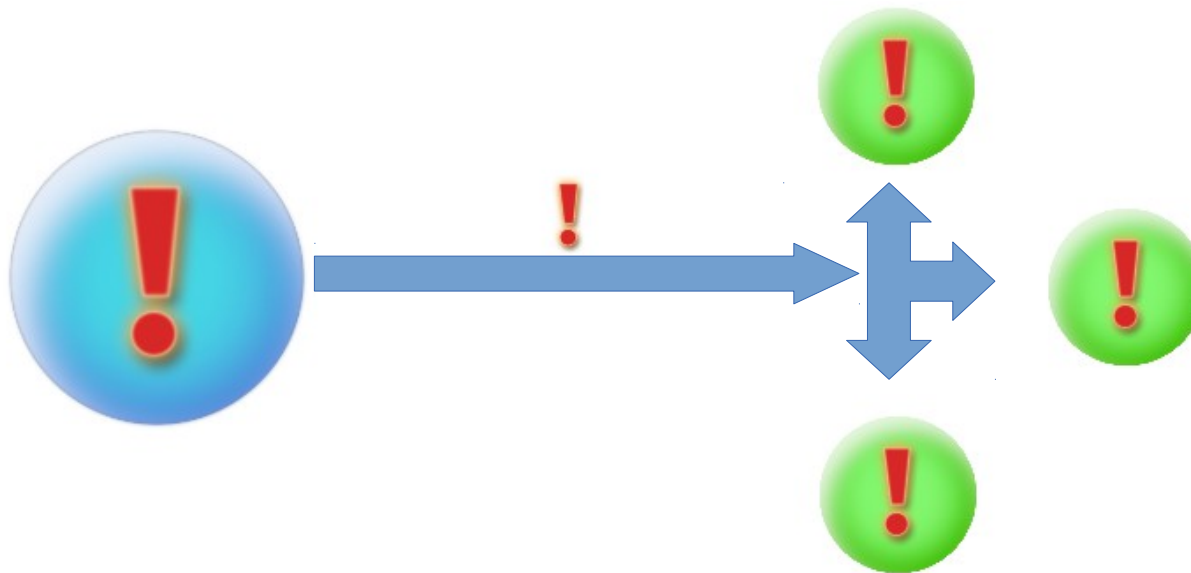
Observer Pattern: Problem

Nachteile:

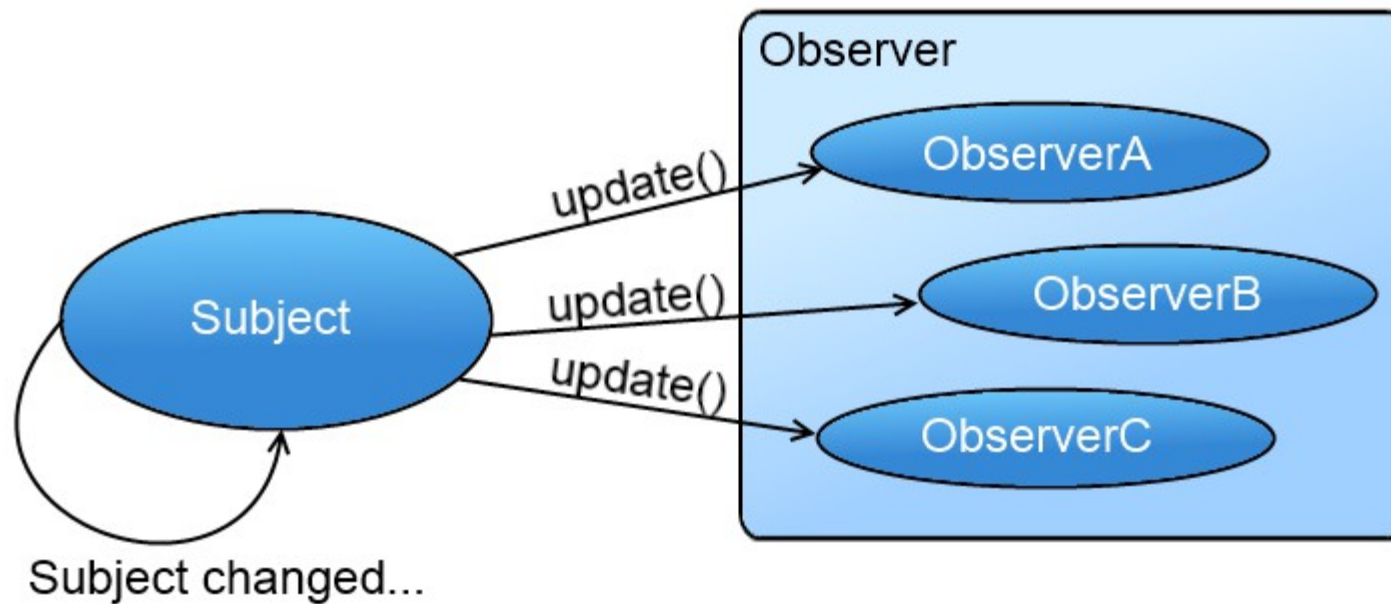
- 1) Enge Verbindung zwischen „FAZVerlag“ und „Abonnenten“
- 2) Erweiterbarkeit stark eingeschränkt!
- 3) Abonnement bestellen oder abbestellen während der Laufzeit nicht möglich!

Observer Pattern: Lösung

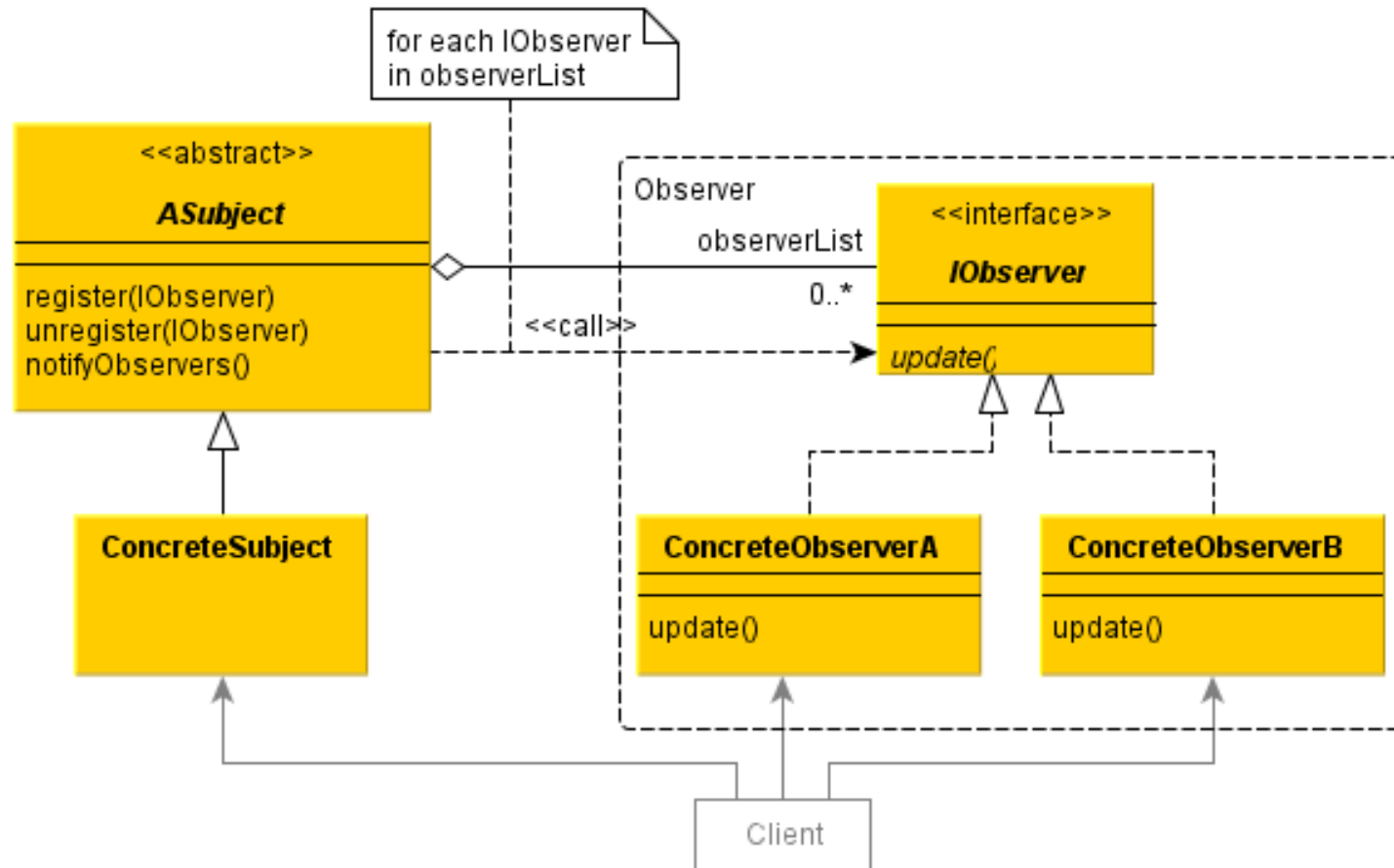
„Definiere eine 1-zu-n-Abhängigkeit zwischen Objekten, so dass die **Änderung des Zustands** eines Objekts dazu führt, dass alle abhängigen **Objekte benachrichtigt** und automatisch **aktualisiert** werden.“ ([GoF], Seite 287)



Observer Pattern: Lösung

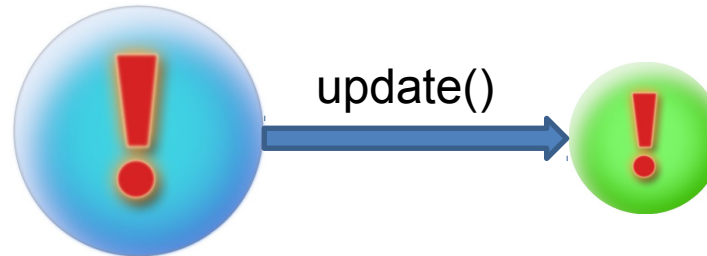


Observer Pattern: Lösung



Observer Pattern: Variationen

Push-Modell

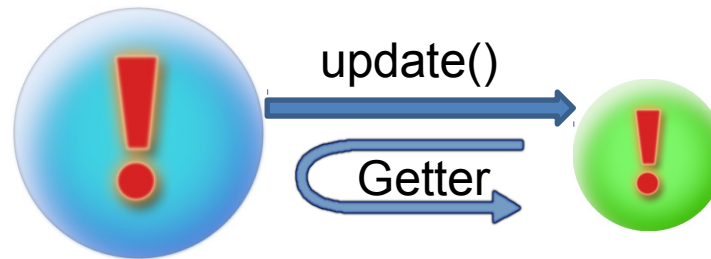


```
public interface IObserver1{  
    public void update(int pLength, int pWidth, boolean pVisible,  
        String pName);  
}
```

- ✓ Observer benötigt keine Informationen über Subjekt -> Starke Entkopplung
- ✗ Nicht jeder Observer benötigt alle/die selben Parameter!
- ✗ Bei Erweiterung müssen alle Observer angepasst werden

Observer Pattern: Variationen

Pull-Modell



```
public interface IObserver2 {  
    public void update(ConcreteSubject pConcreteSubject);  
}
```

- ✓ Jeder Observer holt sich per Getter nur die benötigten Informationen.
- ✓ Bei mehreren Subjekten: Eindeutig von welchem Subjekt!
- ✗ Kann ineffizient werden, da Observer herausfinden muss was sich konkret verändert hat.

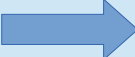
Observer Pattern: Variationen

Pull oder Push?

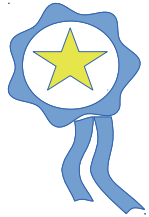
Merke:

Weiß das Subjekt...

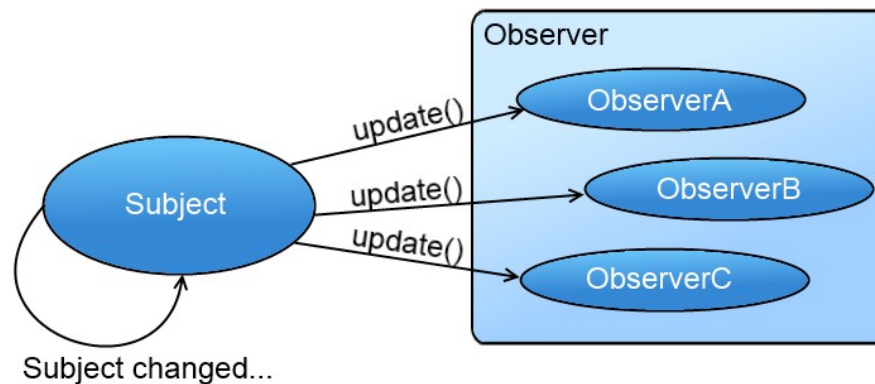
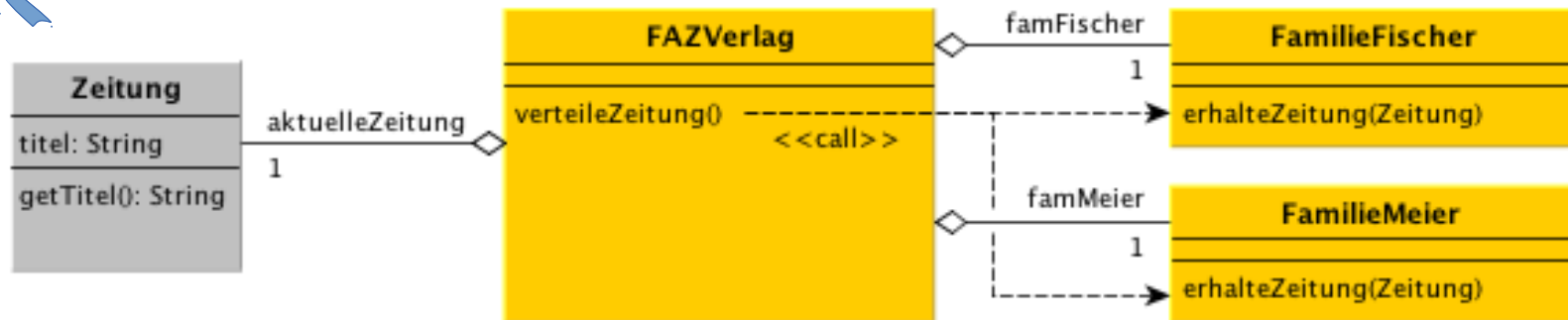
...von den Anforderungen der Observer  Push-Modell

.. nichts über die Observer  Pull-Modell

Observer Pattern: Übung/Beispiel

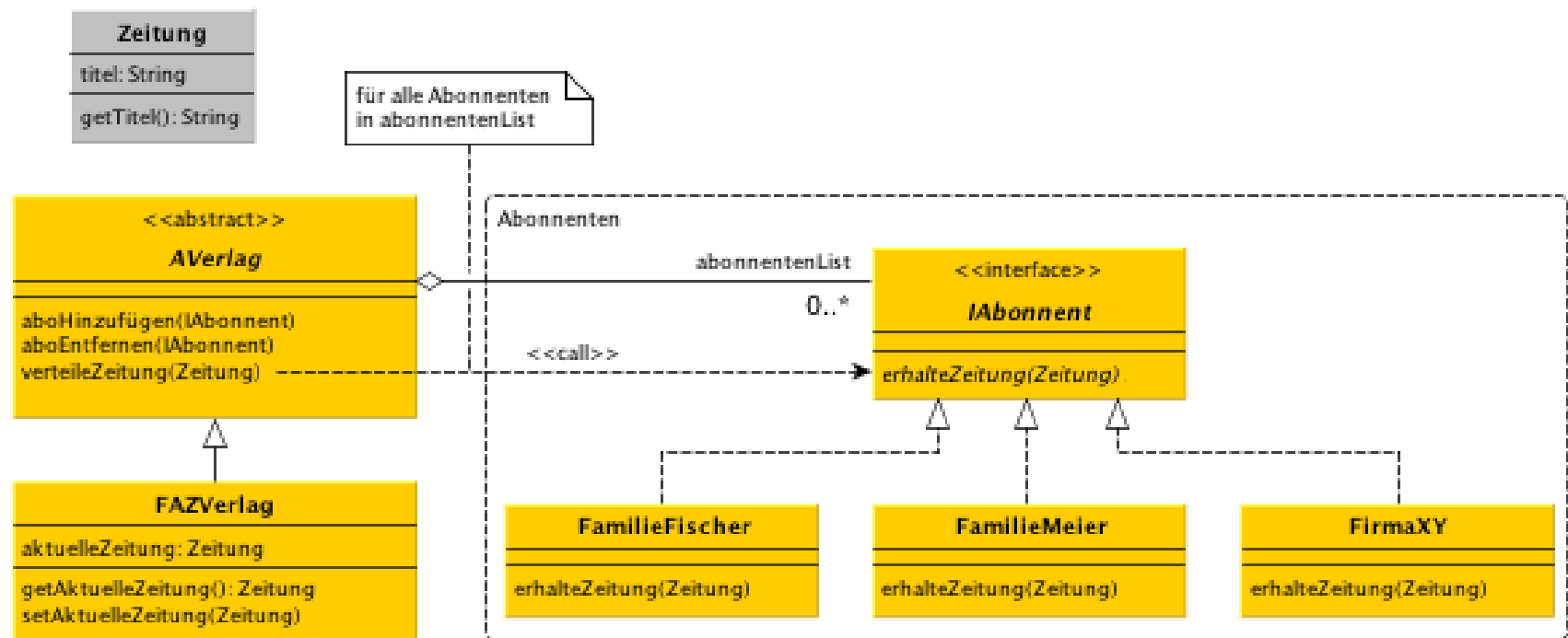


Übung: Klassendiagramm mit Observer Pattern!



Observer Pattern: Übung/Beispiel

Übung: Lösung



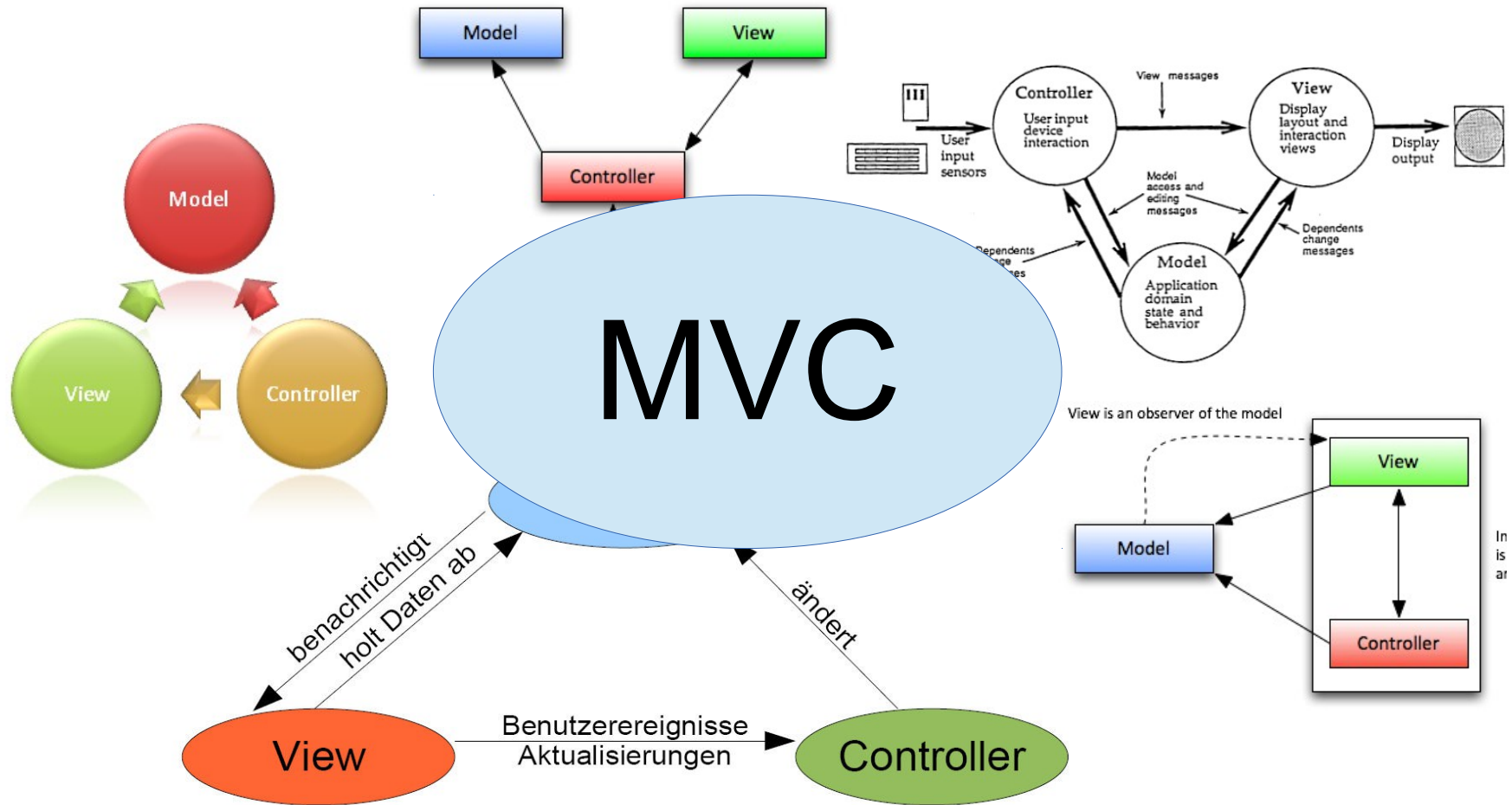
Observer Pattern: Vor und Nachteile

Vorteile

- ✓ Zustandskonsistenz
- ✓ Modularität
- ✓ Wiederverwendbarkeit

Nachteile

- ✗ Aktualisierungszyklen
- ✗ Abmeldung vom Observer



Model-View-Controller

Agenda

- Probleme
- Bestandteile von MVC
- Schlüsselaspekte
- Unterschiedliche Definitionen
- Weiterentwicklungen

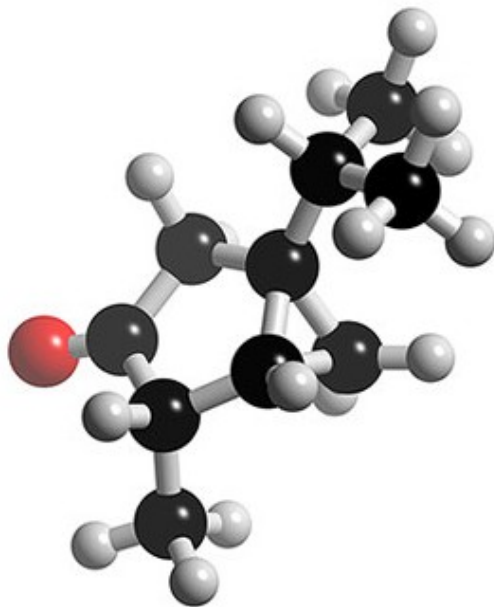
Model-View-Controller: Probleme

Mögliche Probleme der Software-Entwicklung

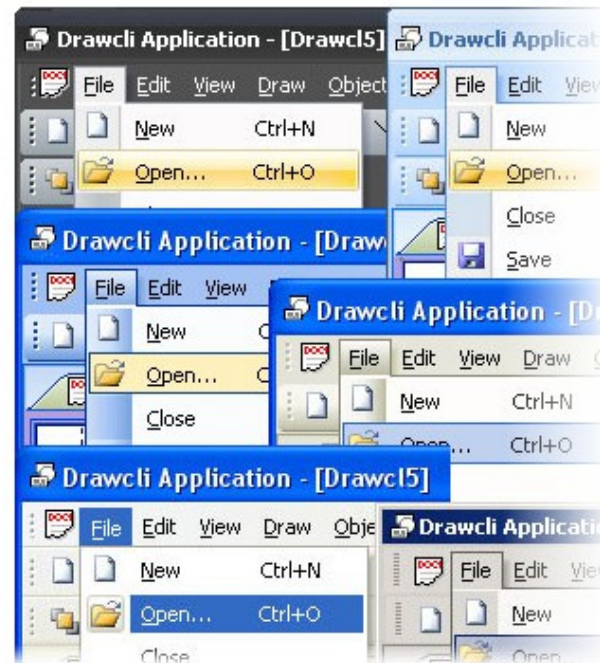
- Mehrere verschiedene Ansichten bei gleichen Daten
- Änderung der Ansicht (z.B. von 2D auf 3D, Punktdiagramm, Liniendiagramm, Kreisdiagramm) bei gleichen Daten.

Model-View-Controller: Bestandteile

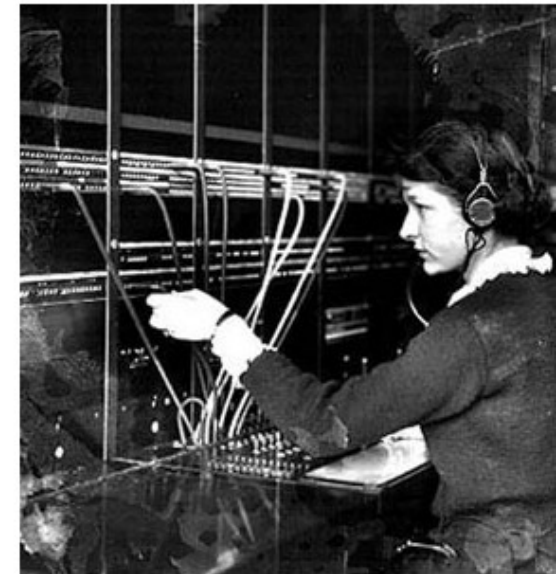
Model:



View:



Controller:



Model-View-Controller: Bestandteile

Model

- Implementiert zentrale Struktur
- Enthält die Geschäftslogik
- Schnittstelle für Datenzugriff
- Kann auch nur Proxy auf Daten sein

Anzahl der Studierenden in
Deutschland im Wintersemester
2012/2013 nach Hochschulart

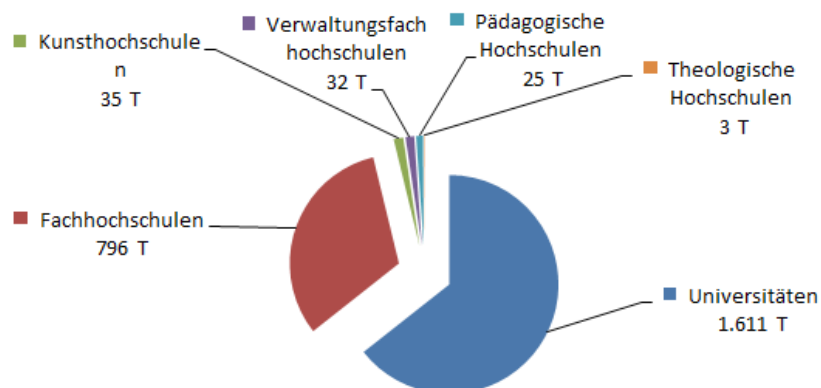
Anzahl Studenten	Hochschulart
1.610.833	Universitäten
796.083	Fachhochschulen
35.144	Kunsthochschulen
32.177	Verwaltungsfachhochschulen
25.188	Pädagogische Hochschulen
2.565	Theologische Hochschulen

Model-View-Controller: Bestandteile

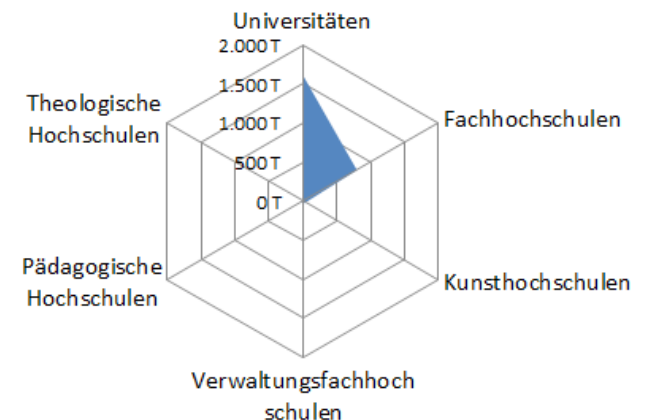
View

- Repräsentiert die Anzeige des Models in dem User Interface

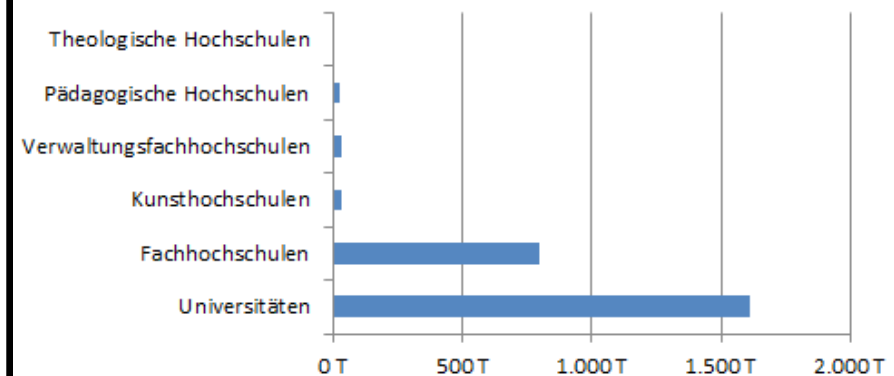
Anzahl Studenten in Deutschland WS 2012/2013 nach Hochschulart



Anzahl Studenten in Deutschland WS 2012/2013 nach Hochschulart



Anzahl Studenten in Deutschland WS 2012/2013 nach Hochschulart



Model-View-Controller: Bestandteile

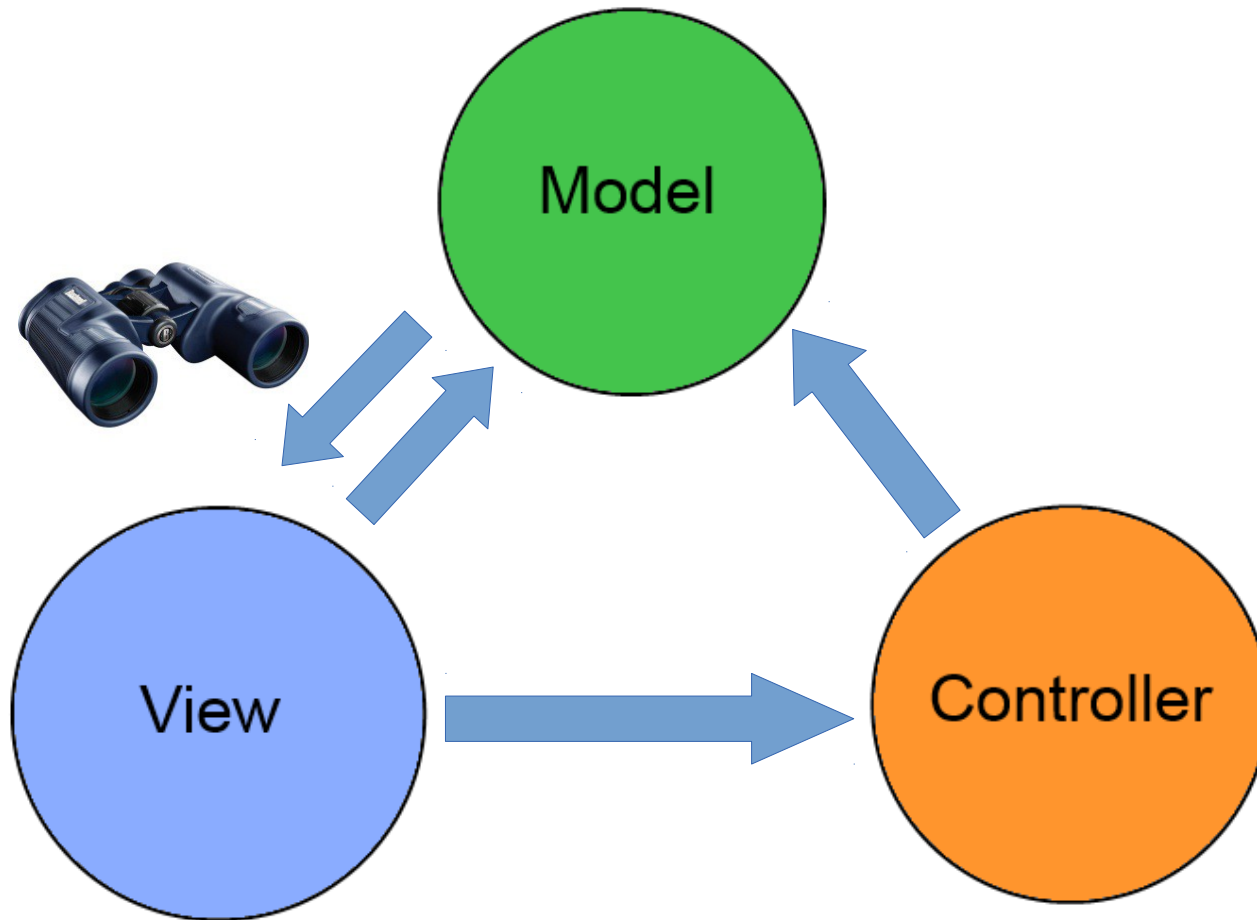
Controller

- Verwaltet Benutzereingaben
- Manipuliert das Model
- Aktualisiert die View



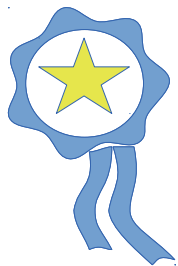
C2.com: "We need SMART Models, THIN Controllers, and DUMB Views"

Eine mögliche Implementierung



Model-View-Controller: Schlüsselaspekte

- Die View/Ansicht von dem Modell trennen.
- Ermöglicht es mehrere verschiedene User Interfaces zu implementieren und die Module besser zu testen.
- Den Controller von der Präsentation trennen.
- Besonders nützlich mit Web Interfaces. (bei GUI Frameworks eher weniger)

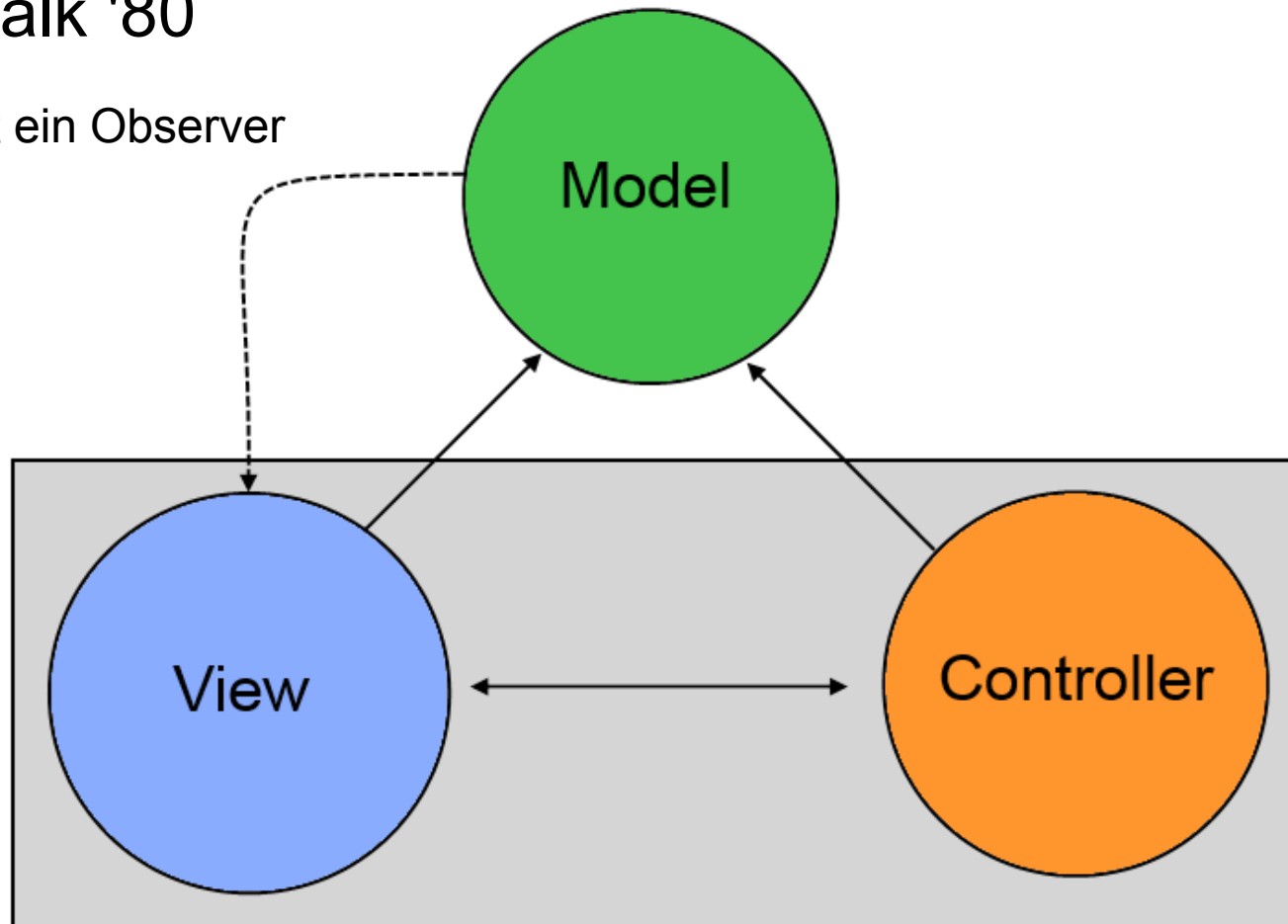


Übung!

Model-View-Controller: Unterschiedliche Definitionen

Smalltalk '80

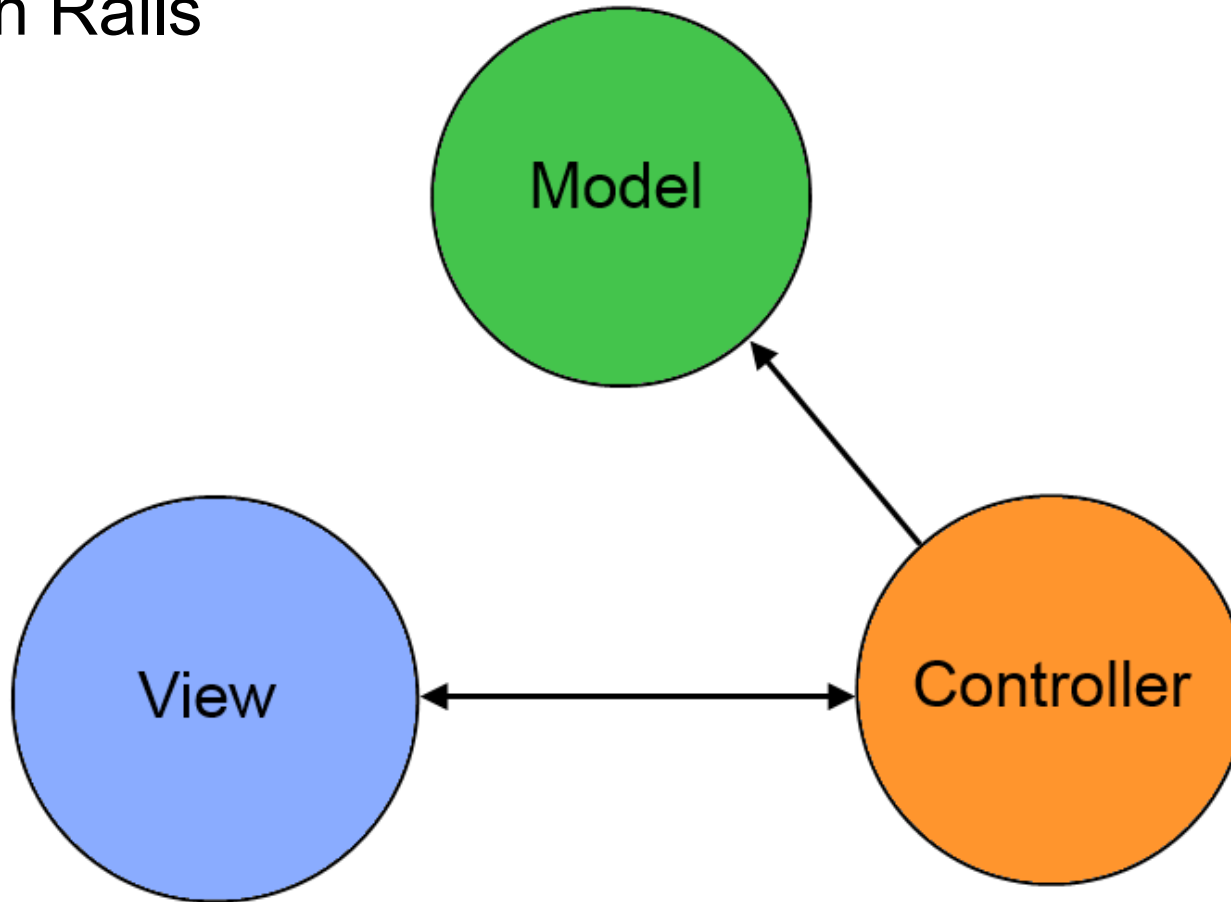
View ist ein Observer



In den meisten Fällen in Smalltalk nicht getrennt!

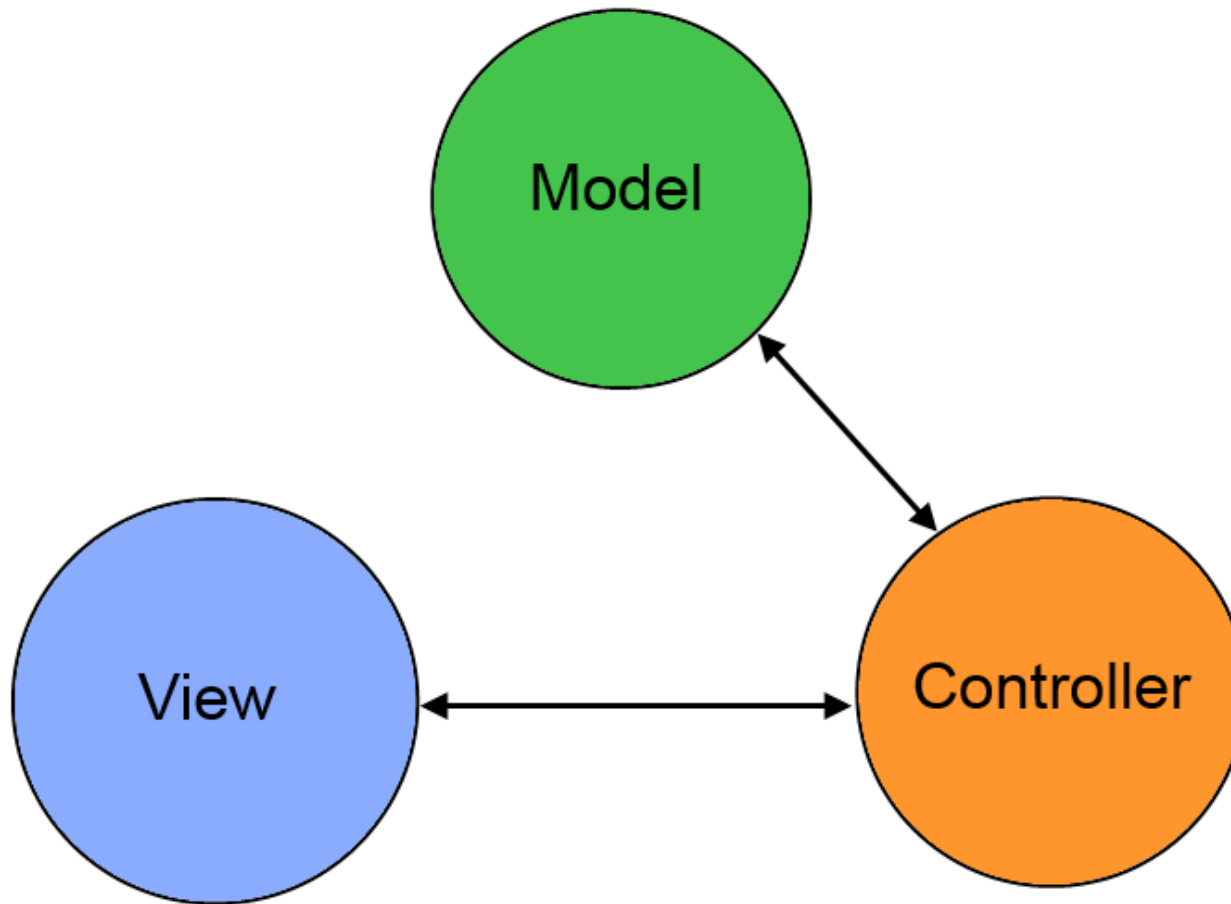
Model-View-Controller: Unterschiedliche Definitionen

Ruby on Rails



Model-View-Controller: Unterschiedliche Definitionen

Cocoa



Quellenangaben

- <http://amix.dk/blog/post/19615>
- <http://www.philippbauer.de/study/se/design-pattern/observer.php>
- <http://c2.com/cgi/wiki?ModelViewController>
- <http://needcoffee.cachefly.net/needcoffee/uploads/2012/02/September.jpg>

Model-View-Controller: Weiterentwicklungen

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