

# 46. Architektur interaktiver Systeme

1

Prof. Dr. rer. nat. Uwe Alsmann  
Institut für Software- und  
Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
Version 13-1.0, 13.07.13

- 1) Benutzungsoberflächen und Anwendungslogik
- 2) Kopplung von synchronen und formularbasierten Benutzungsoberflächen und Anwendungslogik
- 3) Kopplung von reaktiven, graphischen Benutzungsoberflächen und Anwendungslogik
- 4) Controller als Steuerungsmaschinen
- 5) Implementierung der Konnektoren



Softwaretechnologie, © Prof. Uwe Alsmann  
Technische Universität Dresden, Fakultät Informatik

## Literatur

2

- ▶ **Obligatorisch:**
  - [PassiveView] Martin Fowler. Passive View. <http://www.martinfowler.com/eaDev/PassiveScreen.html>. Strikte Schichtung und passiver View.
- ▶ **Weitere:**
  - F. Buschmann. N. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-orientierte Software-Architektur. Addison-Wesley.
  - Entwurfsmuster und Architekturstile. MVC, Pipes, u.v.m.
  - [Herrmann] M. Veit, S. Herrmann. Model-View-Controller and Object Teams: A Perfect Match of Paradigms. Aspect-Oriented System Development (AOSD) 2003, ACM Press
  - Mike Potel. MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. VP & CTO Taligent, Inc.
    - ♦ <ftp://www6.software.ibm.com/software/developer/library/mvp.pdf>
  - html web frameworks
    - ♦ STRUTS <http://exadel.com/tutorial/struts/5.2/guess/strutsintro.html>
    - ♦ Web Application Component Toolkit [http://www.phpwact.org/pattern/model\\_view\\_controller](http://www.phpwact.org/pattern/model_view_controller)



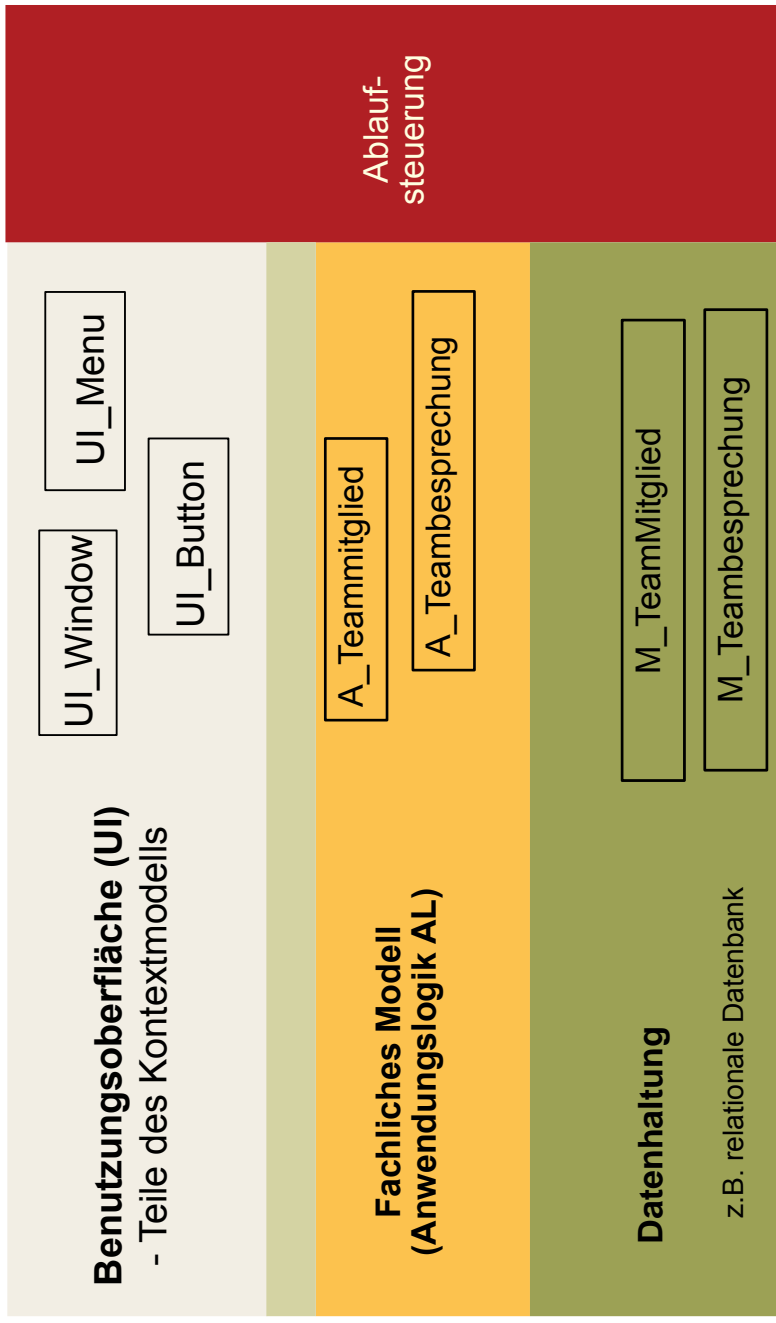
- ▶ Die Architektur interaktiver Anwendungen ist eines der komplexesten Gebiete der Software-Architektur
- ▶ Um sie zu verstehen, brauchen wir *alle Teile* des Kurses:
  - Kollaborationen und Konnektoren
  - Schichten
  - Steuerungs- und Protokollmaschinen
  - Sequenzdiagramme
  - Entwurfsmuster

## 46.1 Benutzungsoberflächen (UI) und Anwendungslogik

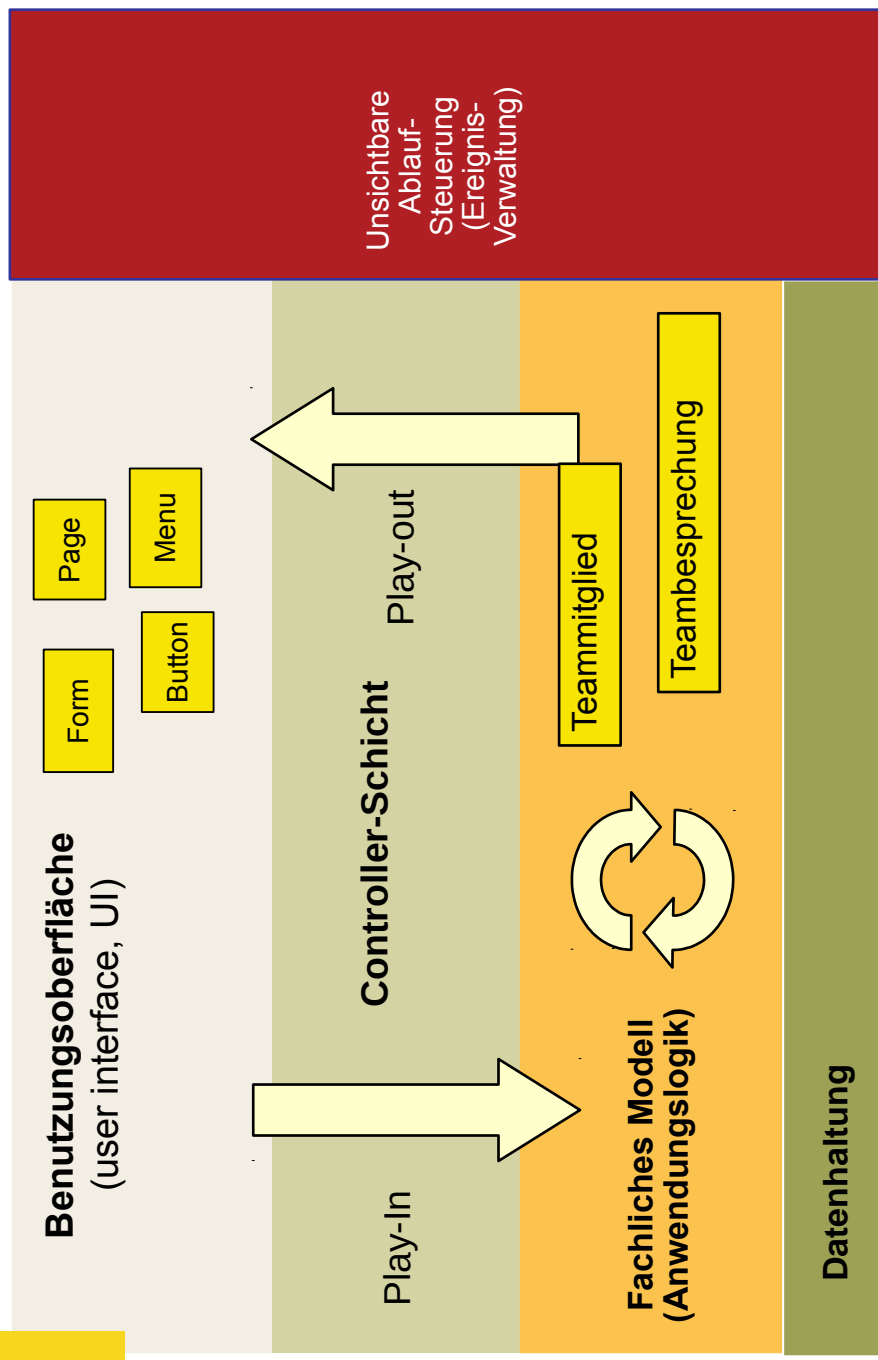
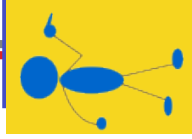
Verschiedene Arten der Kopplung zwischen Benutzer und Software

# 3-Schichtenarchitektur (3-tier architecture)

5



# Controller bildet 4. Schicht zwischen der Benutzungsoberfläche (UI) und der Anwendungslogik



# Arten von Benutzungsschnittstellen (User Interface, UI)

- ▶ **Synchrone UI:** die Anwendungslogik ruft die UI auf und wartet auf Eingaben (Treiber ist die Anwendungslogik)
  - Kommandozeilen-orientiert, textuelle UI (TUI)
  - Maskenorientiert (screen flow) oder formularorientiert (form flow, FUI)  
==> dann kann der Controller entfallen
- ▶ **Asynchrone UI:** die Anwendungslogik reagiert auf die UI (Treiber ist die UI)
  - Graphische UI (GUI)
  - Tangible UI (TUI)

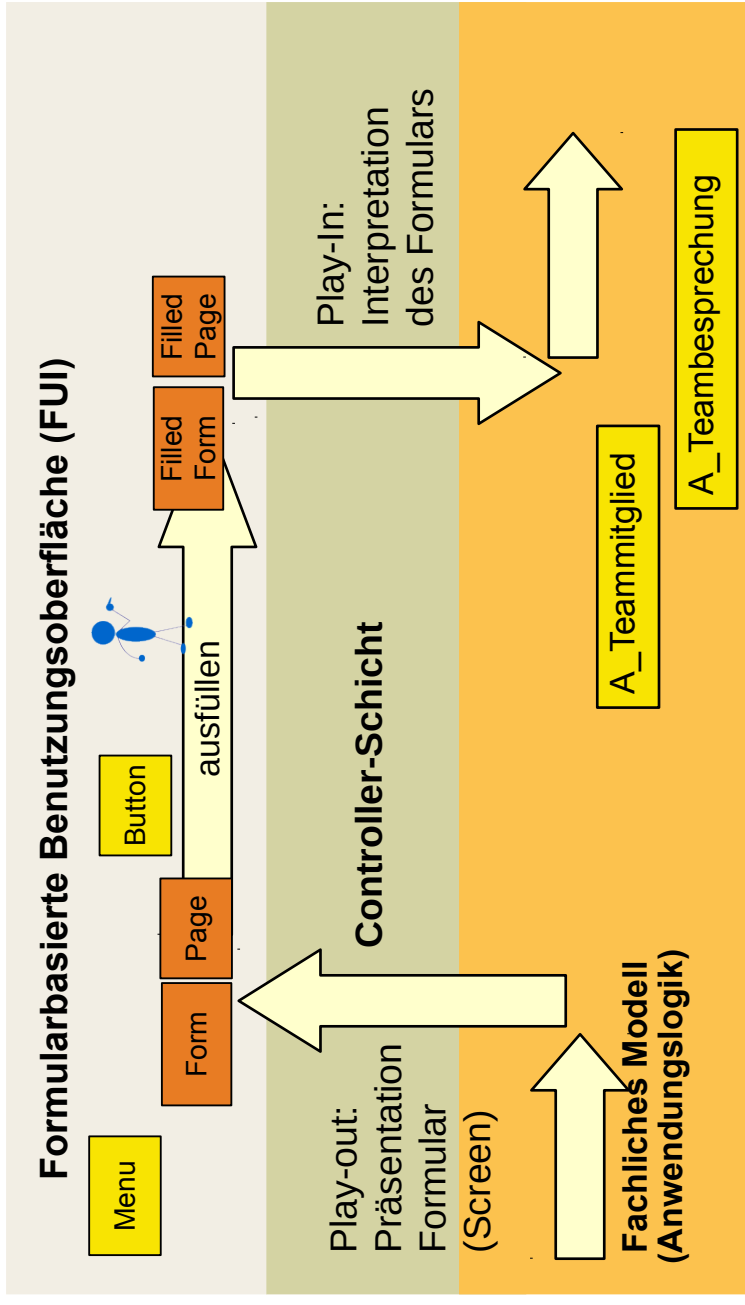
==> dann muss der Controller die parallele Verarbeitung steuern

## 46.2 Kopplung von synchronen Benutzeroberflächen und Anwendungslogik

- Text- und formularbasierte Oberflächen (Form-Based UI, FUI) sind meist synchron mit der Anwendungslogik gekoppelt
- Die Anwendungslogik ruft die Oberfläche auf und wartet auf die Eingaben des Benutzers, z.B. das Ausfüllen von Formularen

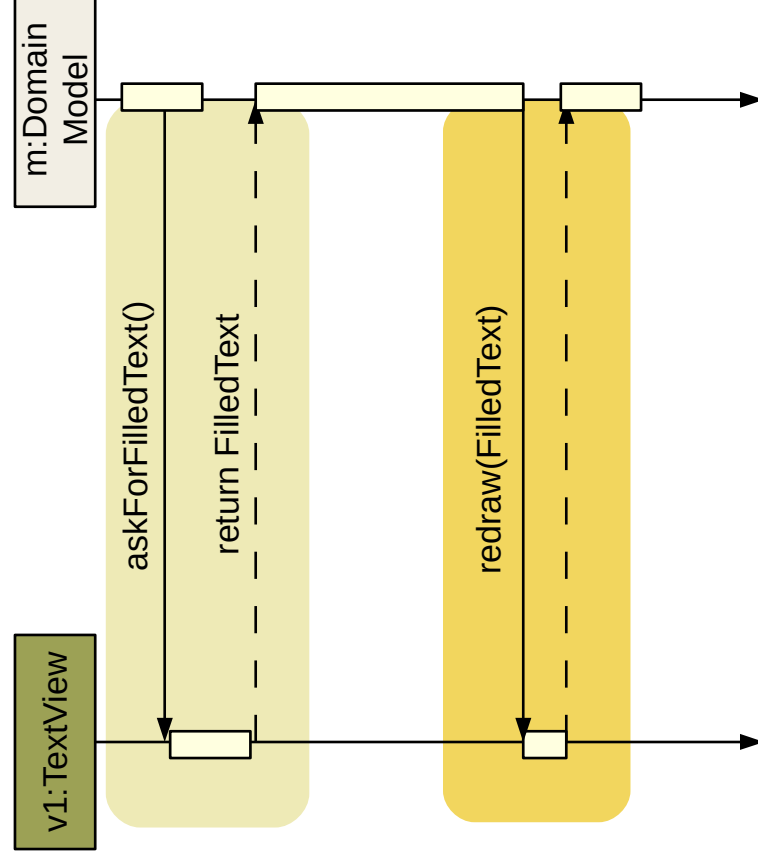
# Synchrone Kopplung zwischen Anwendungslogik, Controllerschicht und FUI

- Die Anwendungslogik ruft das formularbasierte UI mit einem leeren Formular auf und warten auf das Ausfüllen des Benutzers (synchron)



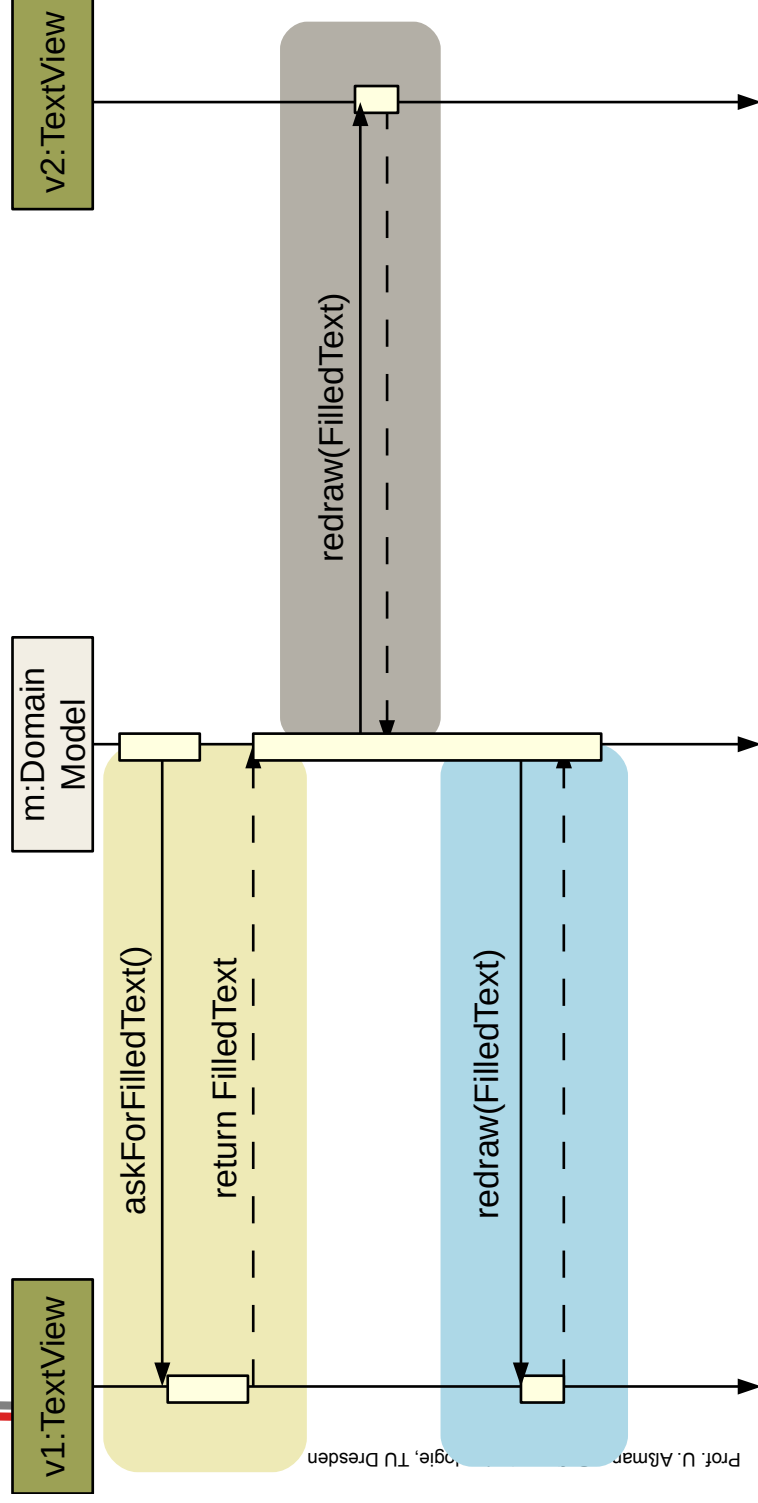
## 46.2.1. Textbasierte UI mit synchronem Update (ein View)

- In Java: Eingabe mit System.in, Ausgabe mit System.out



# Textbasierte UI mit synchronem Update (mehrere Text-Views)

11



Prof. U. Almann, Software Engineering, TU Dresden



## Einfache textuelle Sichten

12

- ▶ Textbasierte UI sind spezielle formularbasierte UI
- ▶ In Java: Aufruf der Objekte `System.in` und `System.out`

```
class PersonModel {
```

```
    ... activities of the model ...
```

```
    System.out.println("Enter a number\n");
    int num = System.in.read();
```

```
    Person p.number = num;
```

```
    foreach (view ; model.getViews()) {
        view.redraw(p);
    }
```

```
    ... further activities of the model ...
```

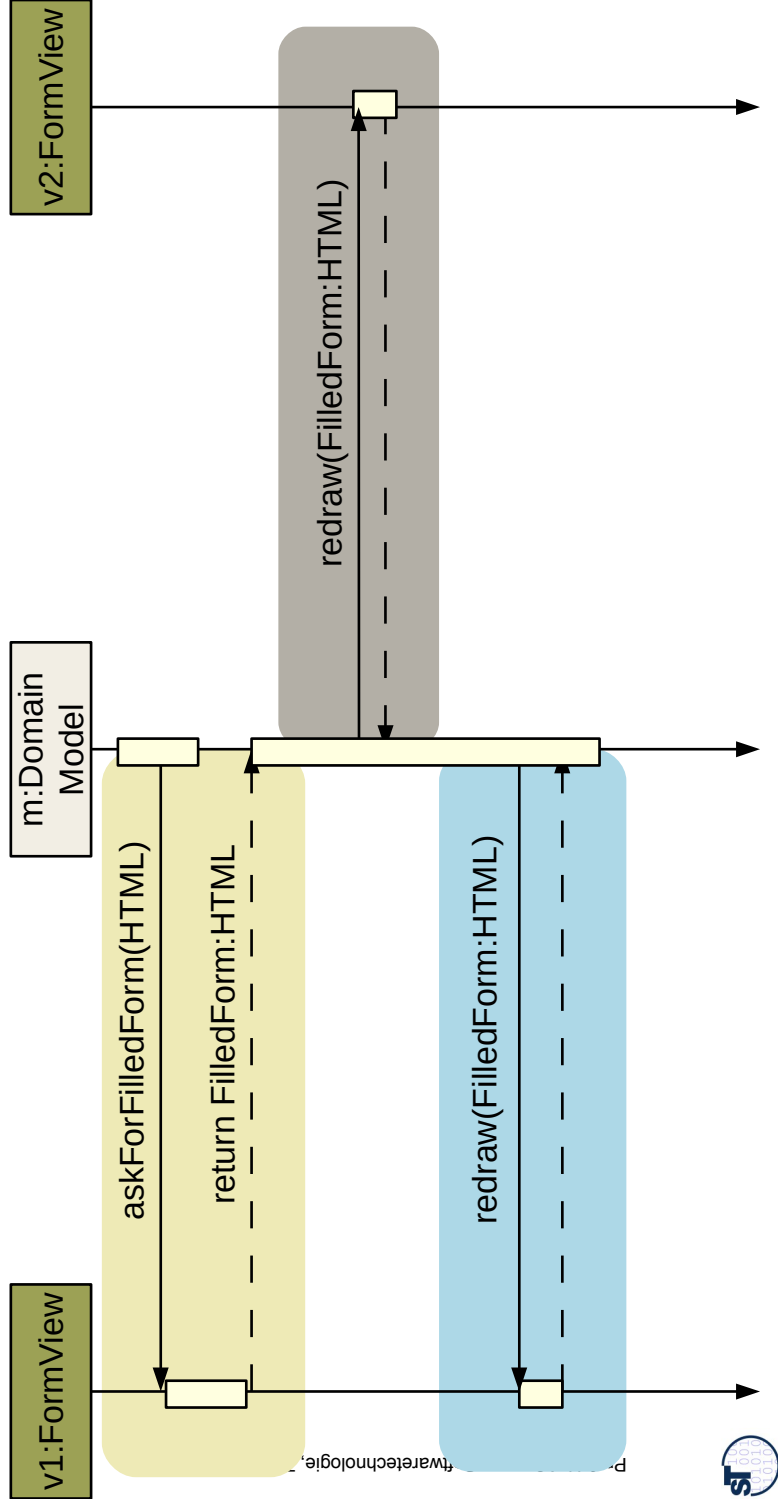
```
}
```

Prof. U. Almann, Software Engineering, TU Dresden



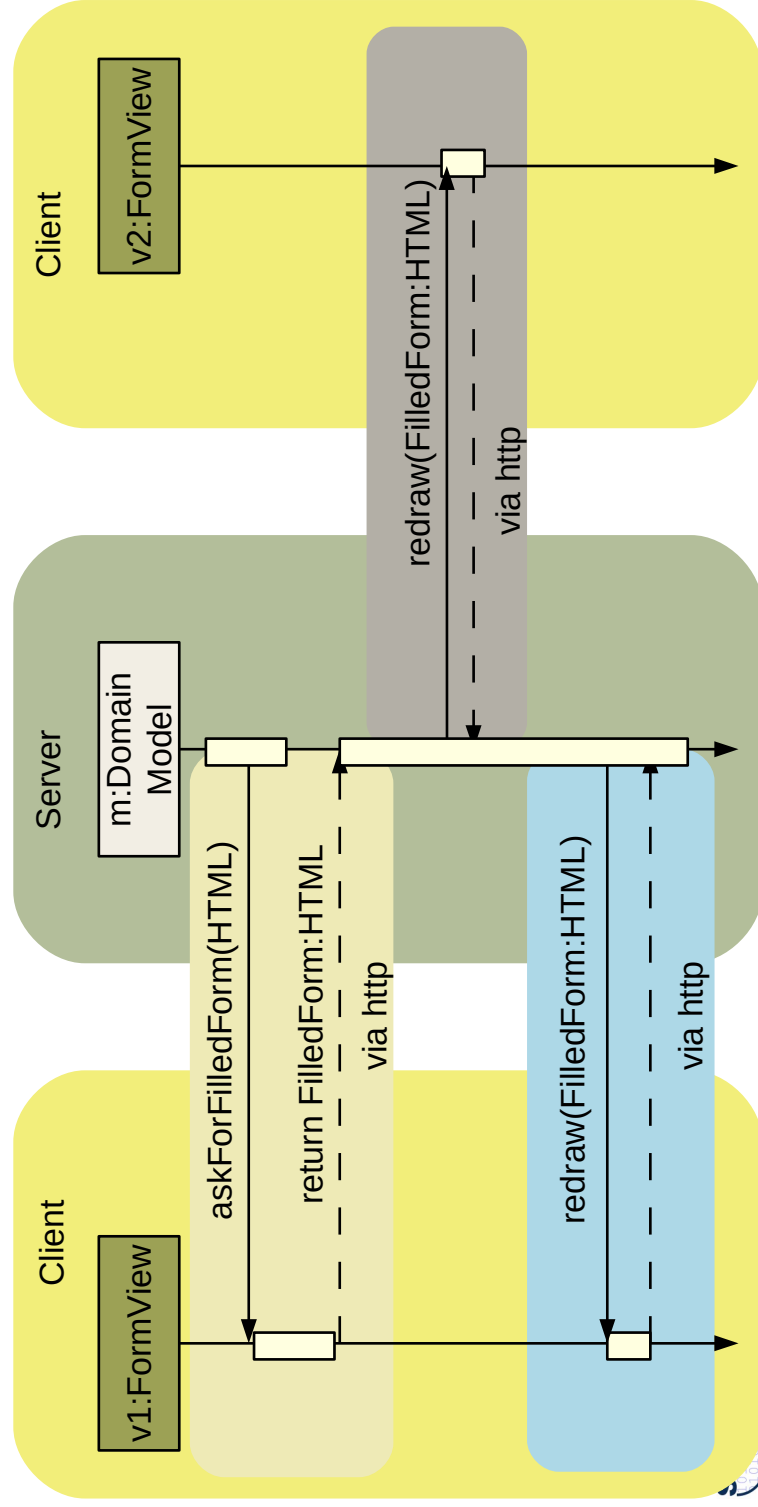
## 46.2.1. Formularbasierte UI mit XML

- ▶ HTML und XML bieten standardisierte Formate für Formulare an, die von Browsern dargestellt, interpretiert, und ausgefüllt werden können

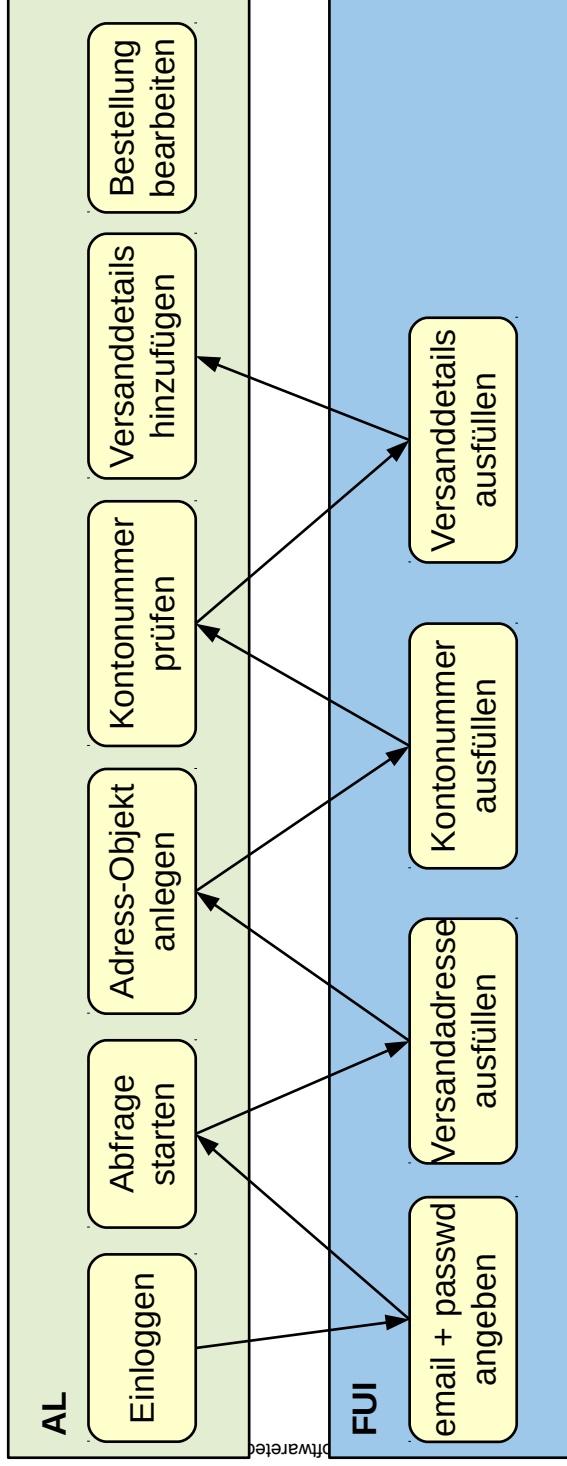


## Formularbasierte UI mit XML übers Web

- ▶ HTML und XML können vom Client zum Server übertragen werden
- ▶ Kanalprotokol http oder https



- ▶ Der Fluss von Daten zwischen AL und FUI wird als **Screen Flow** bezeichnet und kann durch ein Aktivitätsdiagramm mit zwei Swimlanes beschrieben werden
- ▶ Die Initiative liegt in der AL: Der FUI wird jeweils von der AL beauftragt, die Daten einzuholen



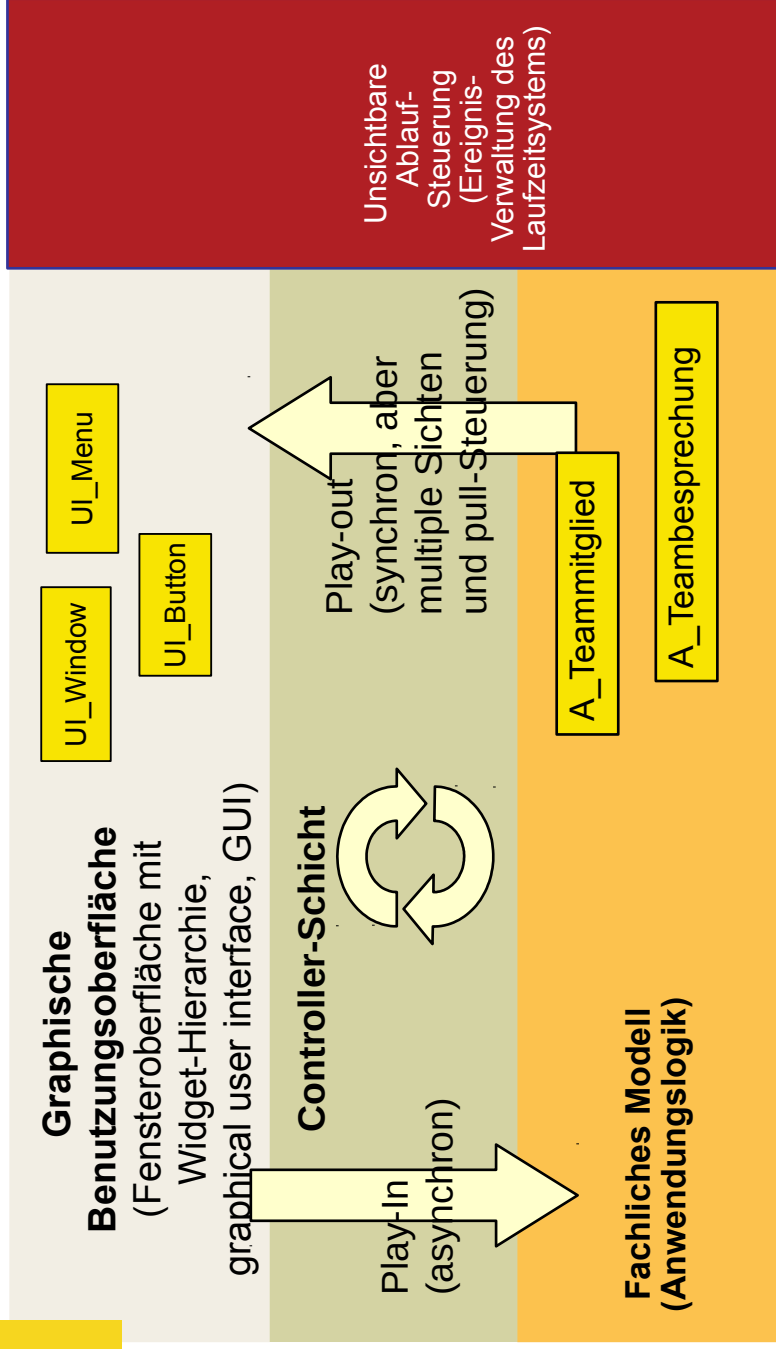
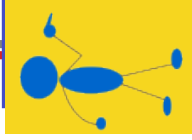
## 46.3 Überblick zu reaktiven graphischen Benutzeroberflächen (GUI)

### Kopplung der GUI und Anwendungslogik durch Controller

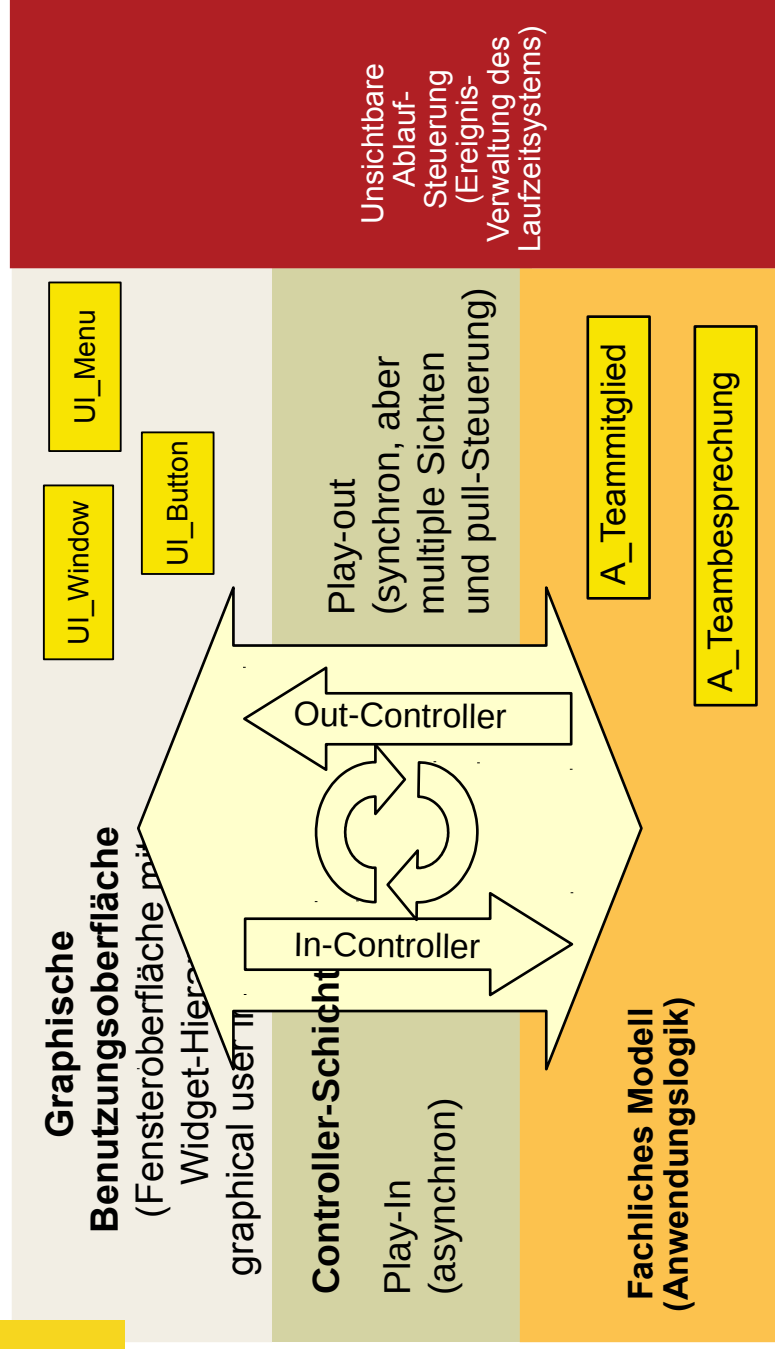
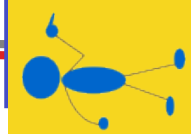
- Bislang war es einfach, aber auch unflexibel
- Jetzt bringt ein *Controller* bzw. eine *Controllerschicht* die Ereignisse, “auslösenden” Fensterelemente (Sicht) und Modell asynchron zusammen
  - Der Controller beherrscht und kapselt die Interaktion, die Initiative geht von ihm aus
  - View und Modell sind gegenüber ihm passiv



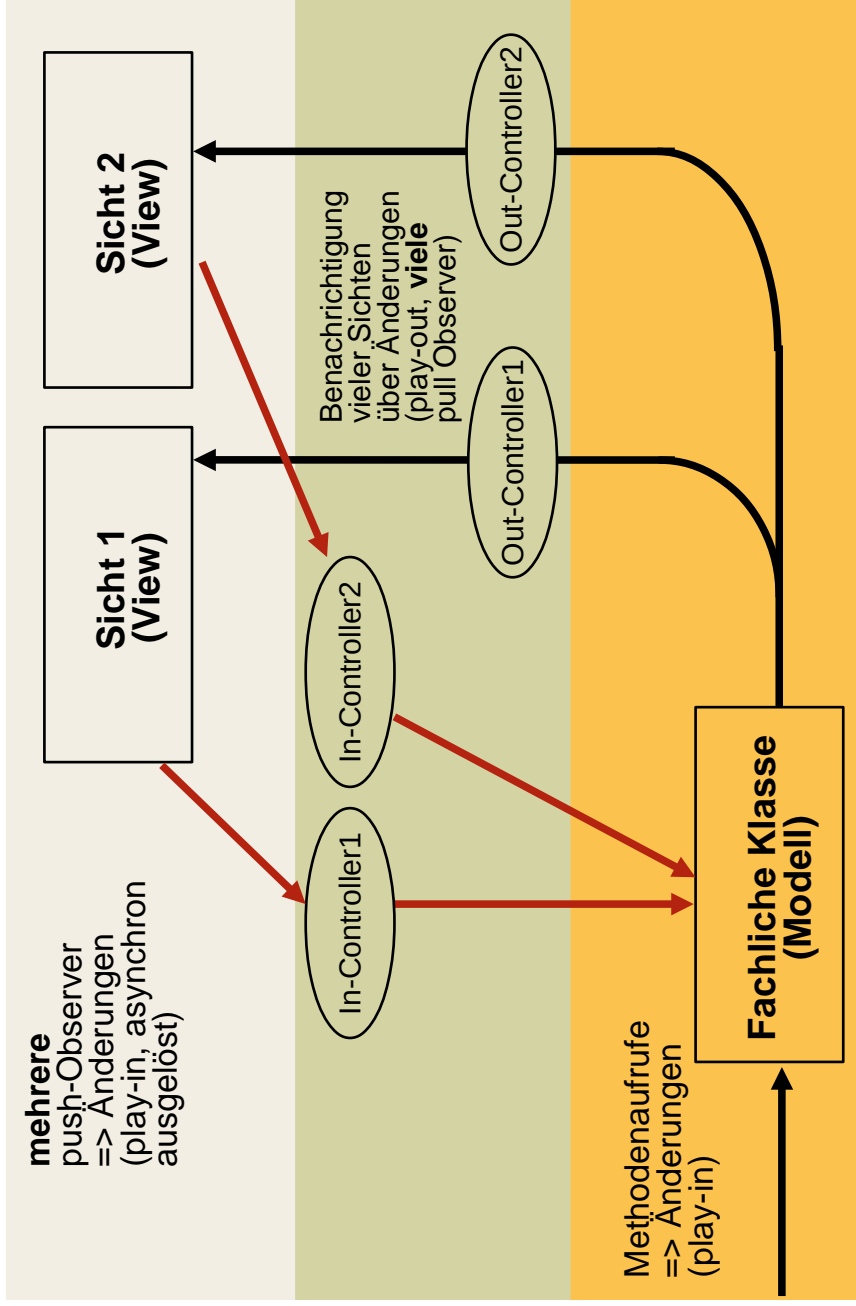
# Schichtenarchitektur der reaktiven Benutzungsoberfläche (GUI)



# Schichtenarchitektur der reaktiven Benutzungsoberfläche (GUI)

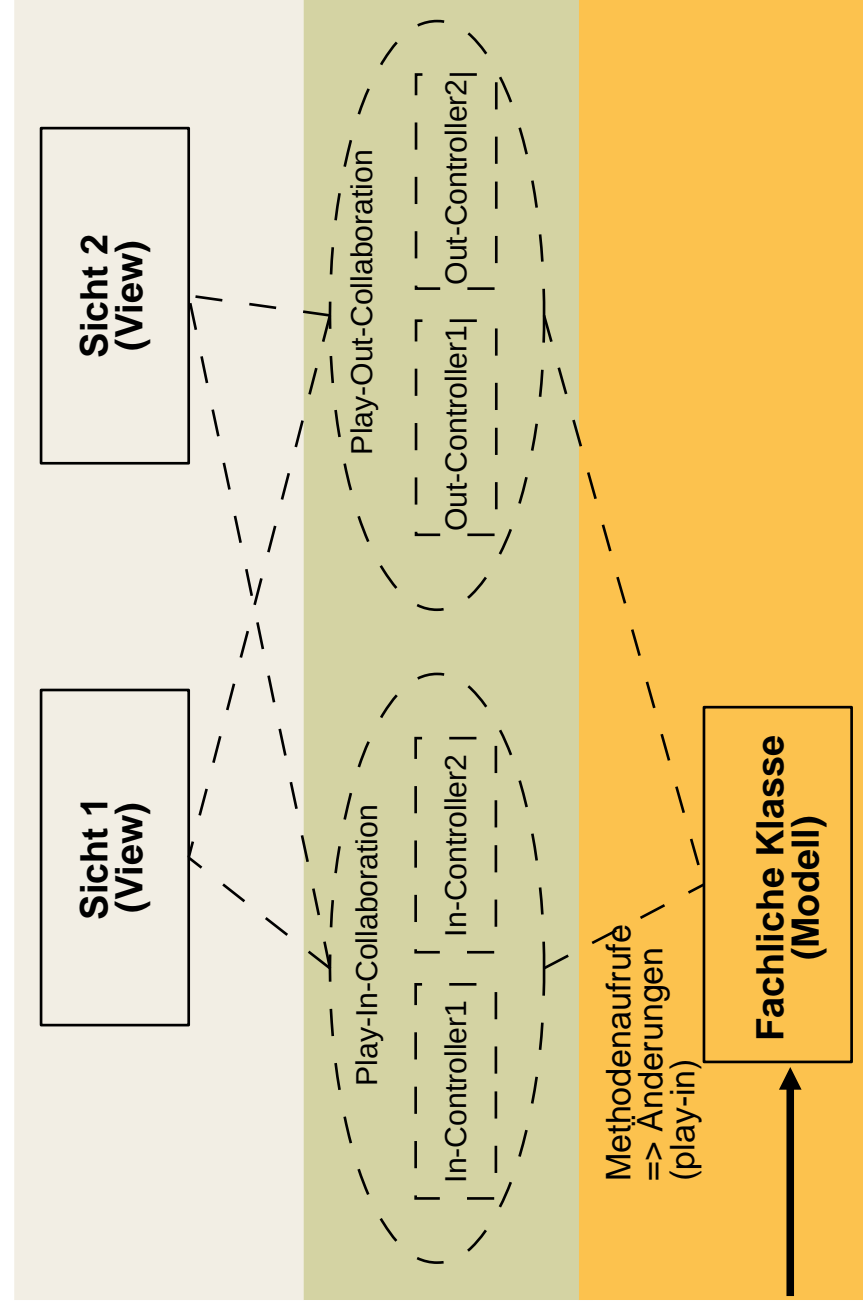


# Modell, Controller und Views in strikter Schichtung



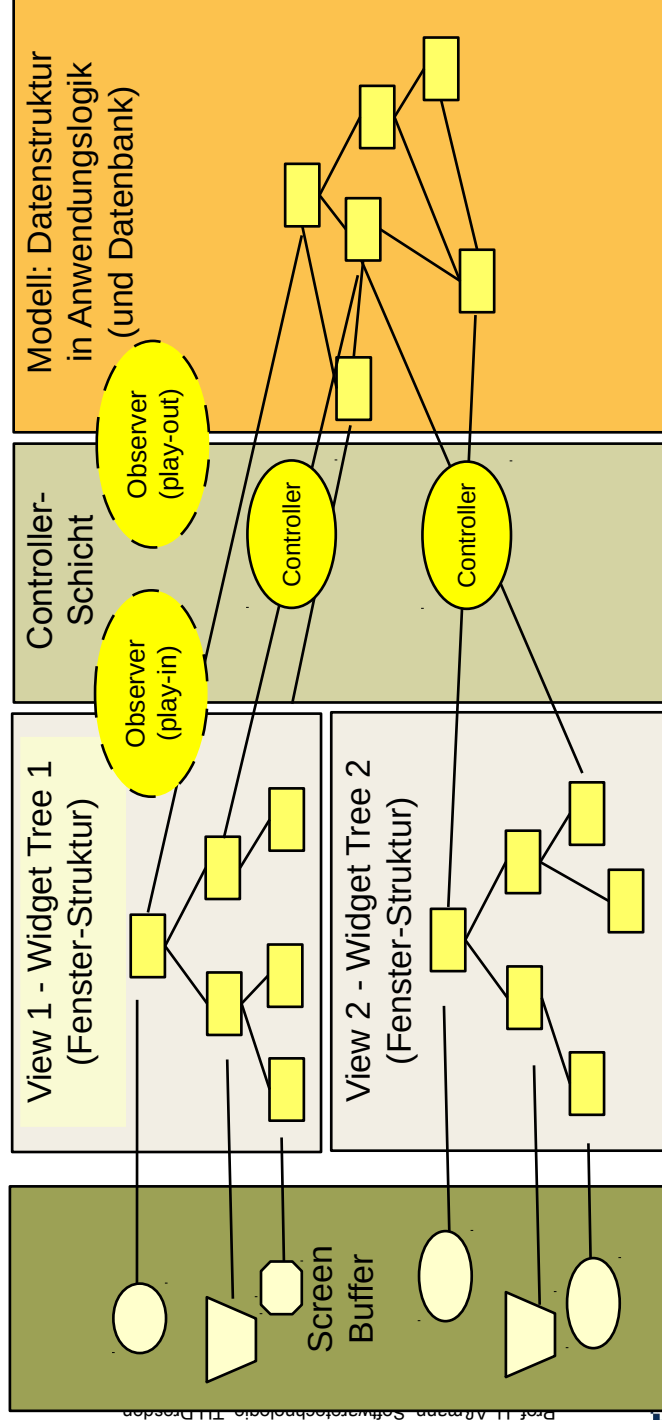
# Controller sind Kollaborationen zwischen Model und View

- ▶ Gibt es ein Hauptobjekt in der Kollaboration, ist der Controller ein Konnektor



# Widgets und Datenstrukturen in asynchronen GUI

- ▶ Fensterstrukturen sind hierarchisch (Einkapselung von Widgets)
- ▶ Datenstruktur in Anwendung wird den Widget-Hierarchien zugeordnet
- ▶ Screen-Buffer zeigt die Widget-Struktur bitweise (`paint()`)
- ▶ Pro View ein Controller



21

# Programme mit asynchronen GUI laufen in 3 Phasen

- ▶ **Interaktive Anwendungen mit GUI laufen in Phasen:**

## 1) Aufbau der Schichten: Aufbau der Datenstrukturen

- Aufbau der Anwendungslogik
- Aufbau der Controllerschicht
- Aufbau der Widget-Schicht (widget hierarchies): Hierarchischer Aufbau der Fensteroberfläche durch Konstruktoraufrufe an Widgets und Einfügen in Fensterhierarchie (widget embodiment)

## 2) Netzaufbau

- Vernetzung der Fensteroberfläche mit der Anwendungslogik** über die *Controller*, um Reaktionen der Anwendungslogik zu ermöglichen
  - Play-Out-Kollaboration:** Anschluß des GUI-Reaktionscodes auf Veränderungen der Modellstruktur (View wird pull-Observer des Controller, indirekt des Modells, Vorbereitung des Play-Out)
  - Play-In-Kollaboration:** Anschluß des Modell-Reaktionscode auf Benutzereingaben (Controller ist push-Observer der Widgets, Vorbereitung des Play-In)

## 3) Reaktionsphase (Reactive, asynchrone Phase)

s. nächste Folie

22

# Zusammenspiel der Widget-Struktur und der Anwendungslogik

23

## 3) Reaktionsphase (Reaktive, asynchrone Phase)

- **Play-In:** bei der die Benutzeraktionen vom System (Ereignisverwaltung) als Ereignisobjekte ins Programm gegeben werden
    - ♦ **Event notification:** Ereignismeldung, dass Benutzer etwas getan hat
    - ♦ **Data transmission:** etwaiger Transfer der Daten
  - **Play-Out:** Bei der in der Anwendungslogik durchgeführten Aktionen die Fensteroberfläche auf den neuesten Stand gebracht wird
    - ♦ **Event notification:** Ereignismeldung, dass Anwendung etwas getan hat
    - ♦ **Data transmission:** Transfer der Daten zum GUI
    - ♦ **Visualization:** Neuzeichnen des GUI
- ▶ Der Steuerfluß eines GUI-Programms wird *nie* explizit spezifiziert, sondern ergibt sich aus den Aktionen des Benutzers oder des Modells
- Die Controllerschicht hat die Kontrolle über das Verhalten
  - *reagiert* auf die Ereignisse im View und im Anwendungsmodell (reaktives System)
  - *steuert* Redraw und Aktionen auf Modell

Prof. U. Almann, Softwaretechnologie, TU Dresden



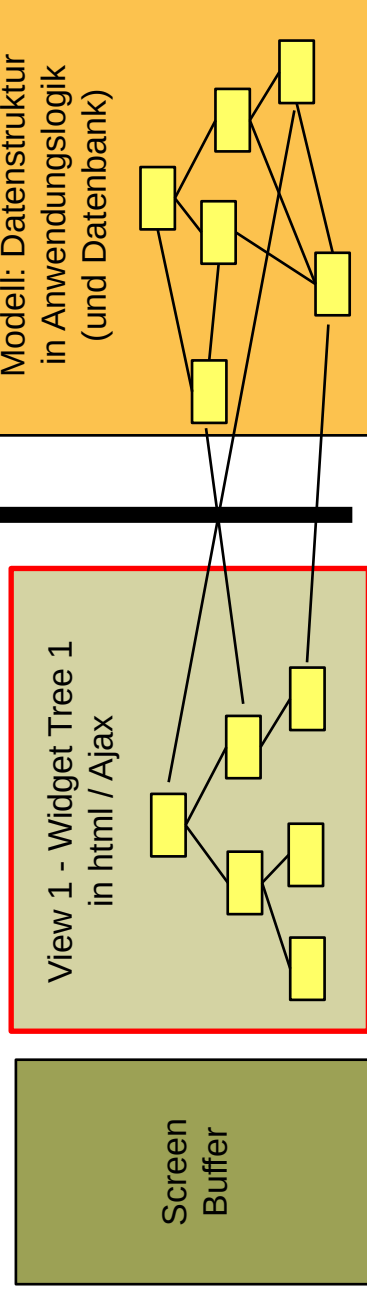
24

## 1) Aufbauphase Schichten: Aufbau der Widget-Struktur und fachl. Modell

Verschiedene Techniken für den Aufbau der Datenstrukturen:

- **Rich Client:** Durch Konstruktoraufrufe und Additionen von Unterwidgets zu Oberwidgets (encapsulation)
  - rein in Java-AWT/Swing, mit expliziter Konstruktion der Widget-Hierarchien
- **App:** App-Frameworks wie Android oder iOS
- **Web:** z.B. Durch einen HTML-Baum, der von einem Brauser interpretiert wird (für Webanwendungen)
  - Durch einen XML-Baum, der von einem XML-Parser eingelesen und als Objekt-Struktur im Speicher abgelegt wird (XUL - Firefox, XAML - Vista)
  - Durch einen HTML-Baum, der bei Veränderungen inkrementell im Brauser nachgeladen wird (Web 2.0, Ajax)

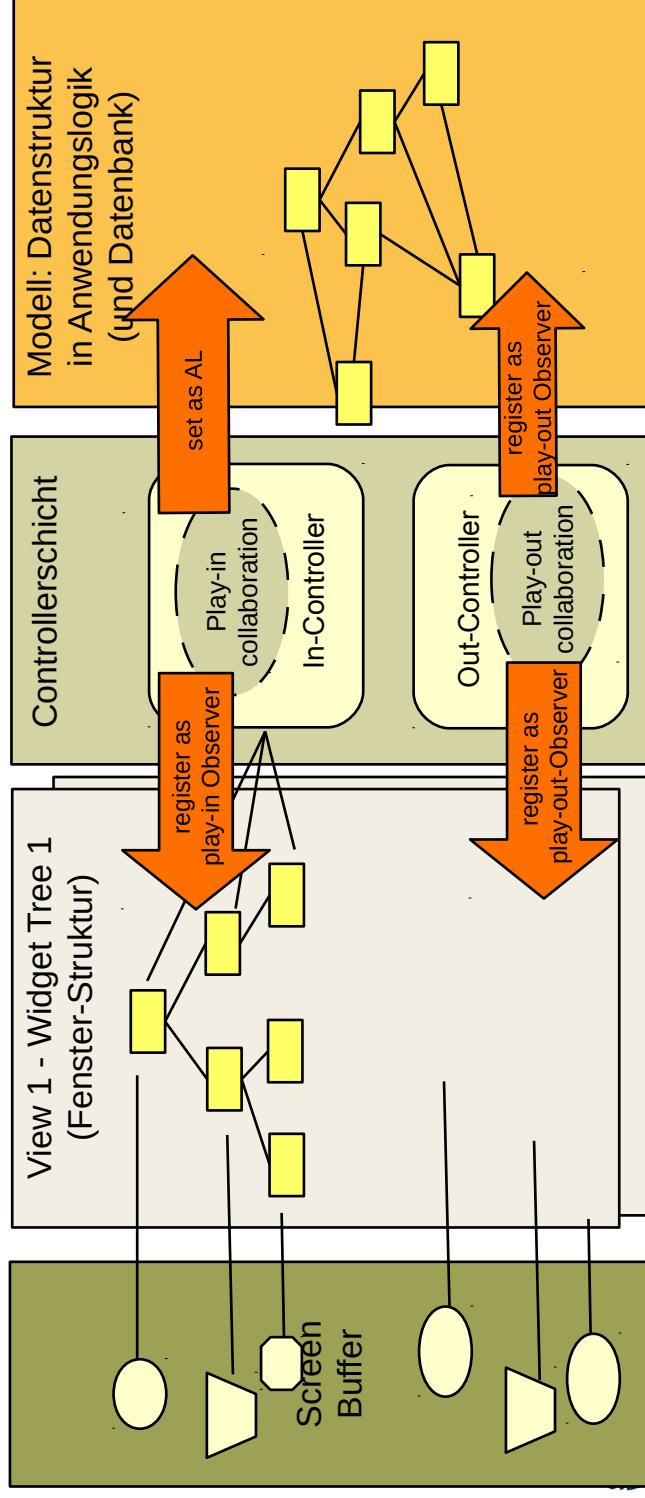
Prof. U. Almann, Softwaretechnologie, TU Dresden



## Phase 2) Netzaufbauphase: Aufbau der Verbindung

25

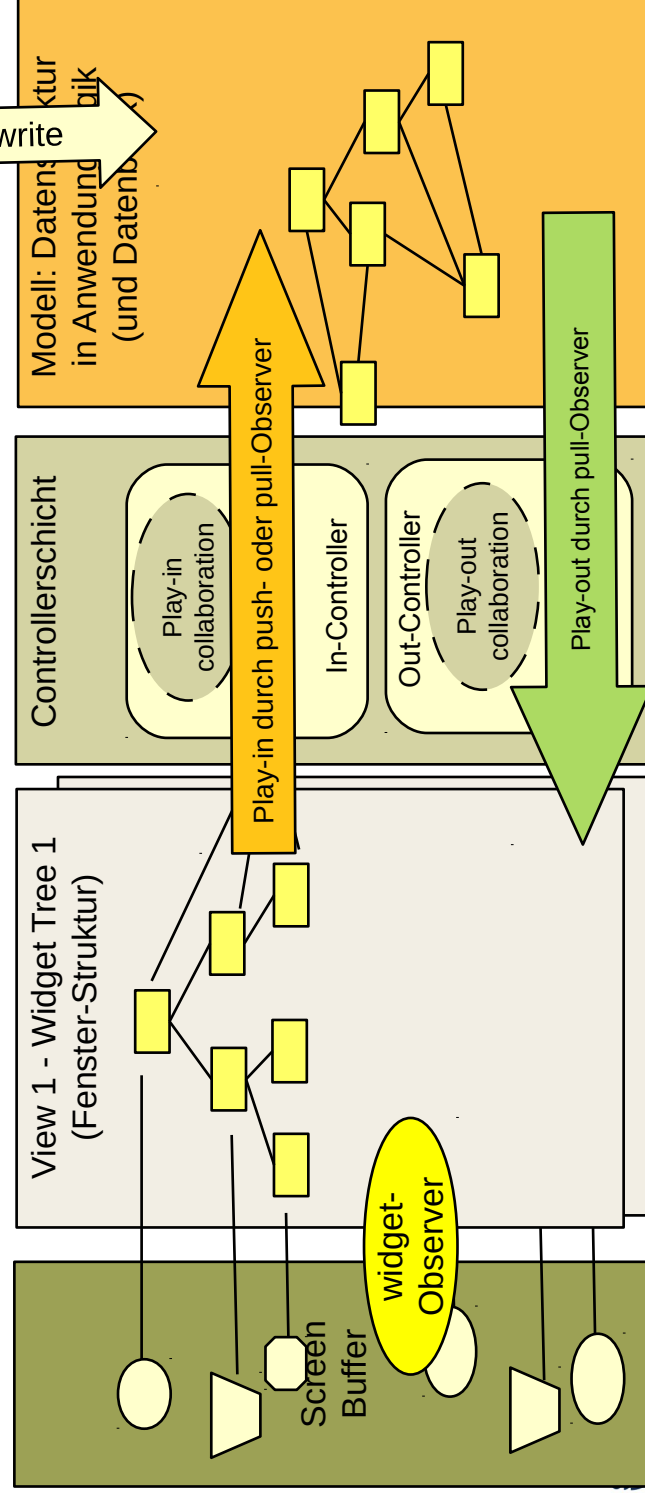
- Die Netzaufbauphase verbindet mit Kollaborationen
  - GUI, Input-Controller und AL für Play-In (In-Connector)
  - AL, Output-Controller und GUI für Play-Out (Out-Connector)



## Phase 3) Überblick MVC Dynamik

26

- Model** ist passiv. Der Controller interpretiert die Eingaben und schreibt das Modell entsprechend
- View** ist weitg. passiv. Controller benachrichtigt View, wenn sich was im Modell geändert hat
- In-Controller** ist ein Observer, der wenig Daten (Events) zu transferieren hat, kann also als push-Observer oder pull-Observer implementiert werden; meist push-Observer
- Out-Controller** muss u. U. große Datenmengen transferieren und wird meist als pull-Observer realisiert



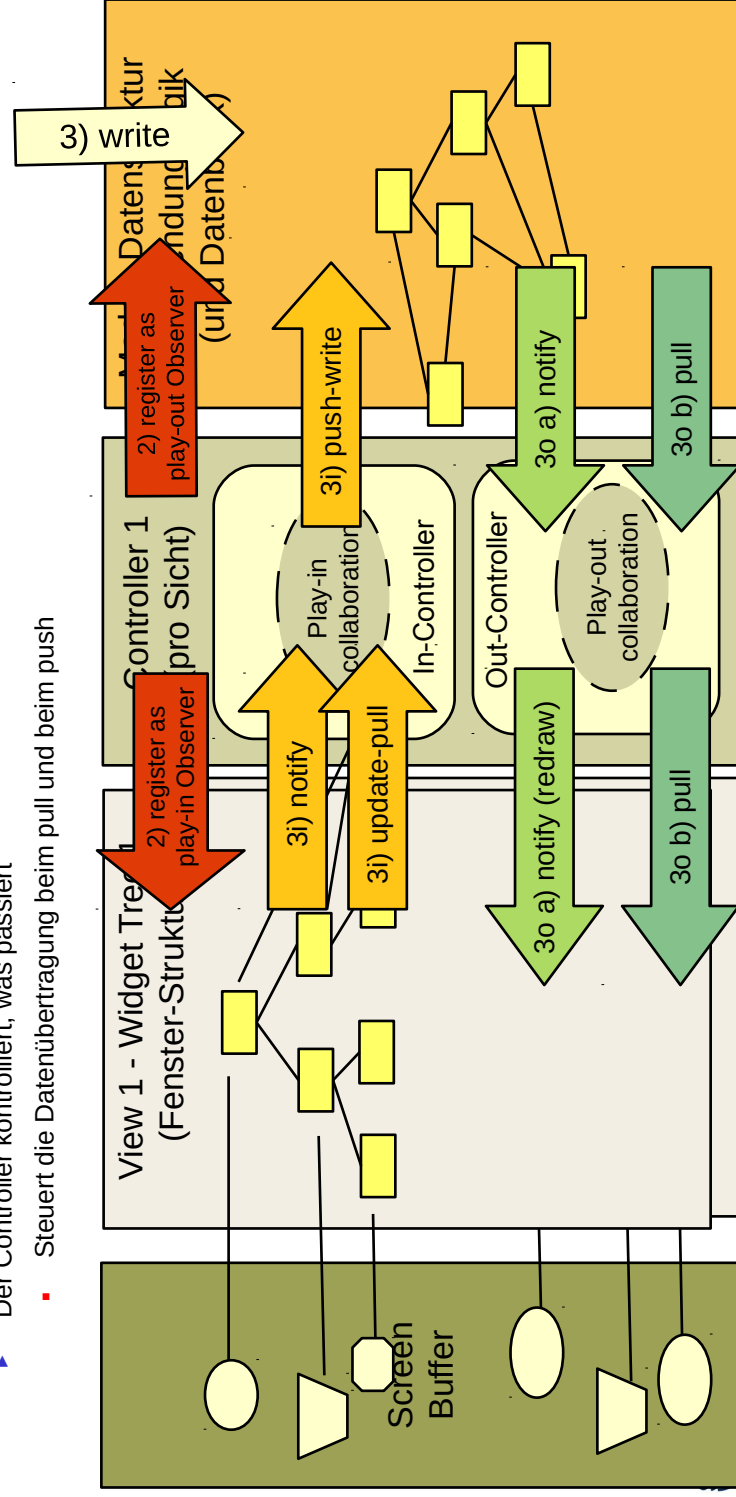
Phase 3 (Dynamik) trennt zwischen Ereignisverarbeitung und Datentransport  
Die Konnektoren setzen push- oder pull-Observer-Muster ein

Phase 3 behandelt Verteilung (Web) mit unterschiedlichen Controller-Architekturen

Frameworks geben die Architektur vor (z.B. Spring, Grails, Ruby on Rails)

## Gesamte MVC-Dynamik (indirektes Play-In und Play-Out)

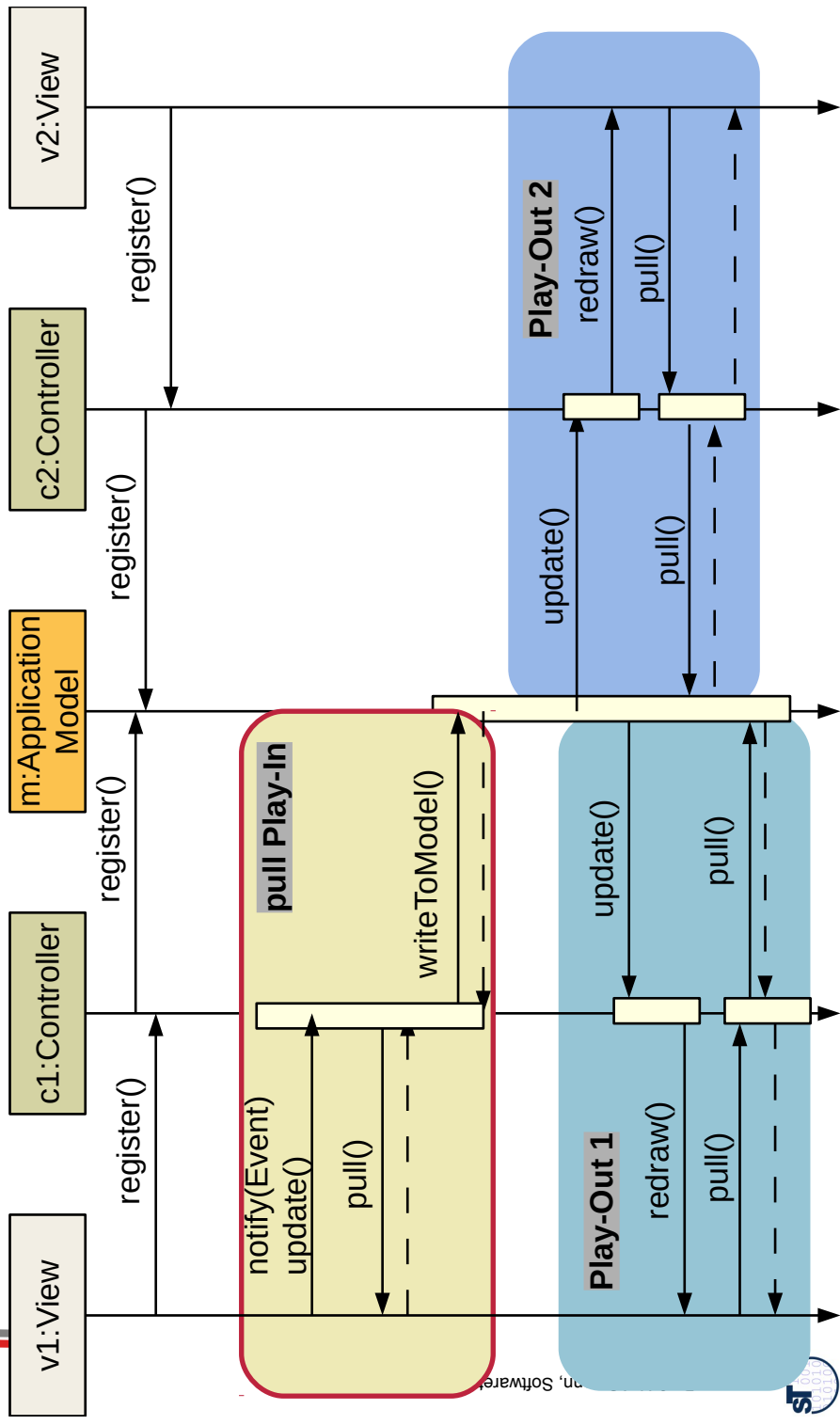
- ▶ Model ist völlig passiv, wird vom Controller geschrieben
- ▶ View is ebenfalls passiv, wird vom Controller aktiviert und gelesen
- ▶ Play-Out Observers indirektem play-out:
  - greift *indirekt* über den Observer auf das Modell zu (update, data-pull)
- ▶ Der Controller kontrolliert, was passiert
  - Steuert die Datenübertragung beim pull und beim push



# Play-In mit passivem View und pull-In-Controller; Passives Play-Out mit indirektem pull-Out-View

[PassiveView]

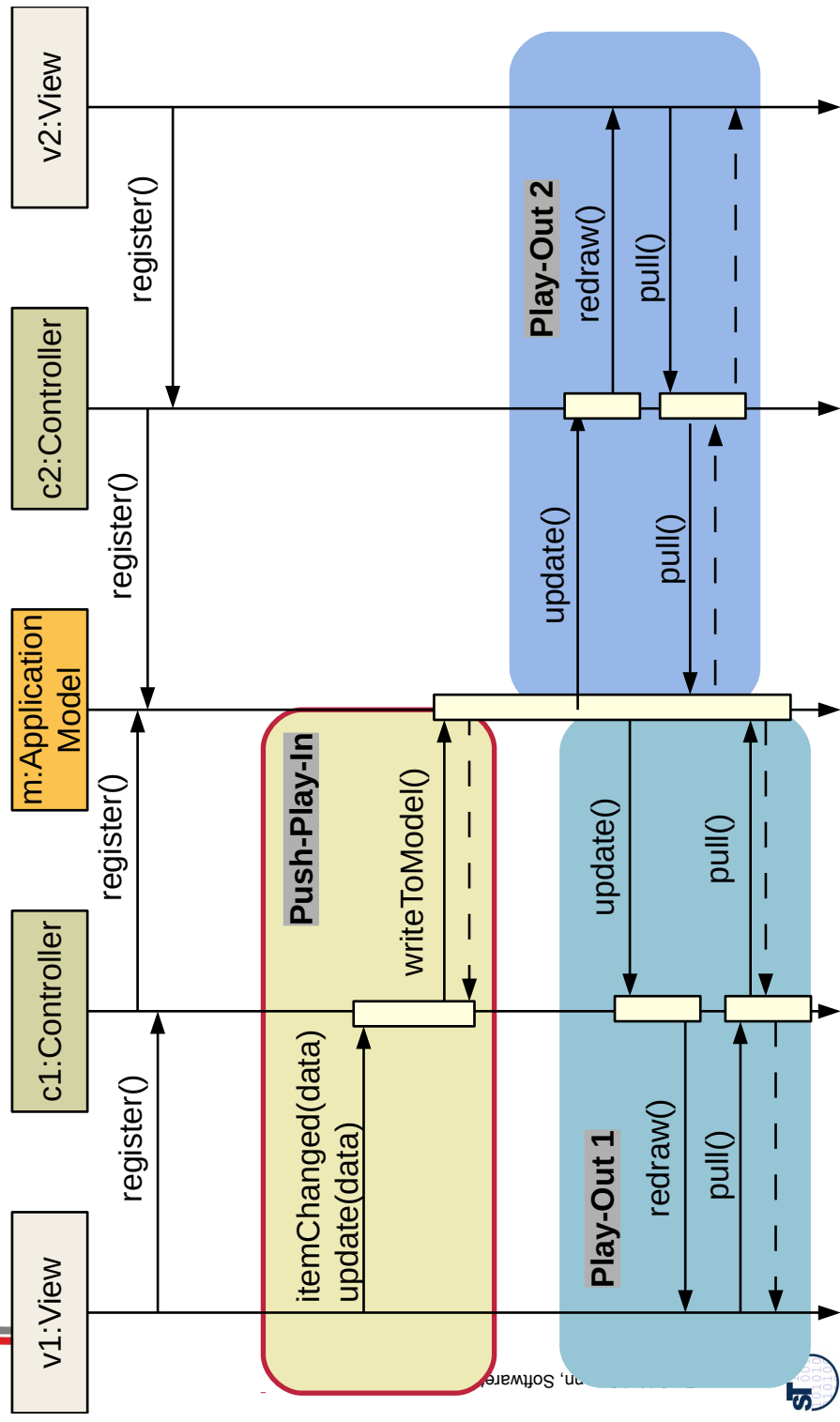
29



## Variante:

# Play-In mit aktivem View und push-In-Controller; Passives Play-Out mit pull-Out-View

30



# 46.4 Controller als Steuerungsmaschinen

31



Software-Technologie, © Prof. Uwe Alsmann  
Technische Universität Dresden, Fakultät Informatik

## Implementierung der Controller

32

Ein Controller ist eine Steuerungsmaschine, die die Ereignisse auf der Fensterhierarchie (UI) in die Aufrufe an die Anwendungslogik *übersetzt* (u.u.)

- ▶ Ereignisse auf der Fensterhierarchie (UI)
  - Button-Pressed, WindowClosed, MenuItemSelected, etc.
- ▶ Aufrufe an die Anwendungslogik:
  - Erzeugen von Kommandoobjekten
  - Schreiben auf Materialien (Domänenobjekte)
  - Aufrufen von Tools und Workflows

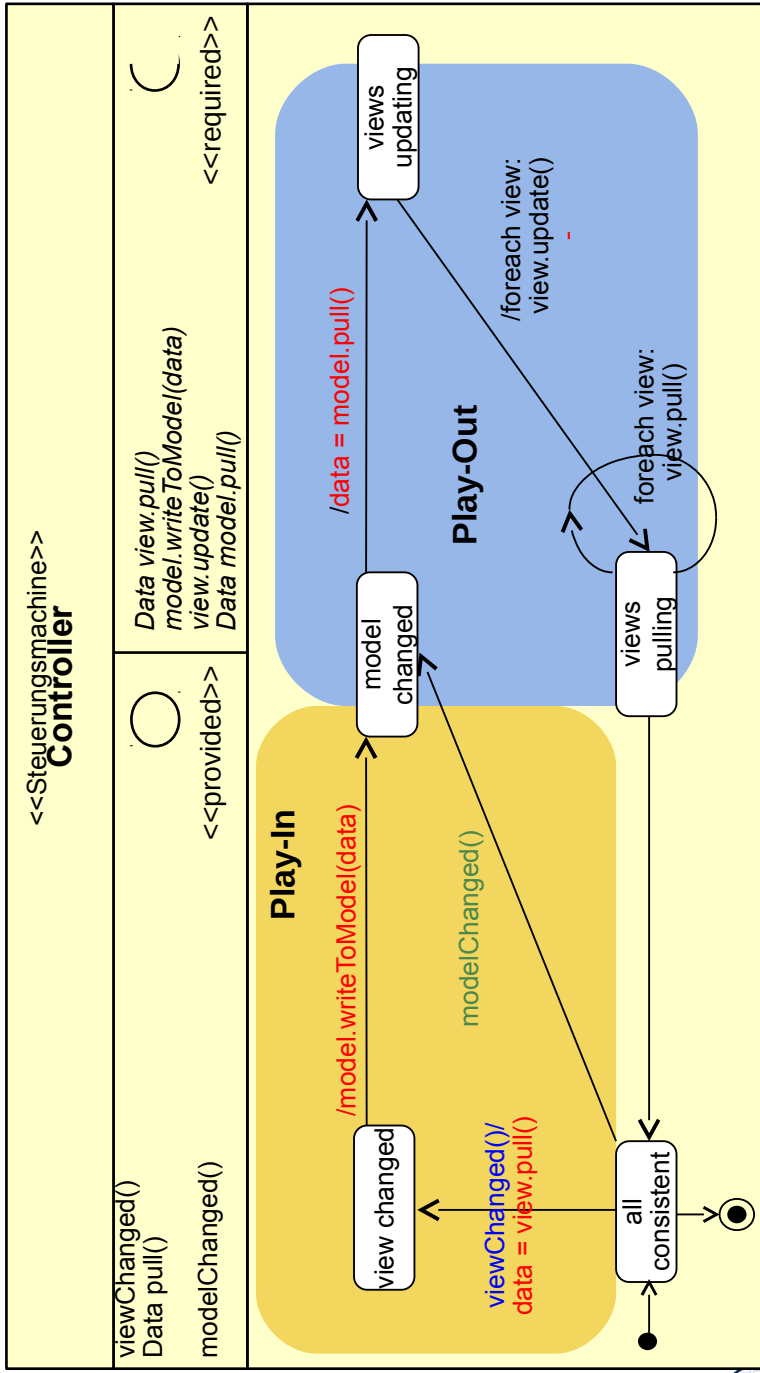
Ein In-Controller übersetzt die Ereignisse des UI in die Ereignisse der AL.  
Ein Out-Controller übersetzt die Ereignisse der AL in die Ereignisse der UI.  
Beide können kombiniert sein.





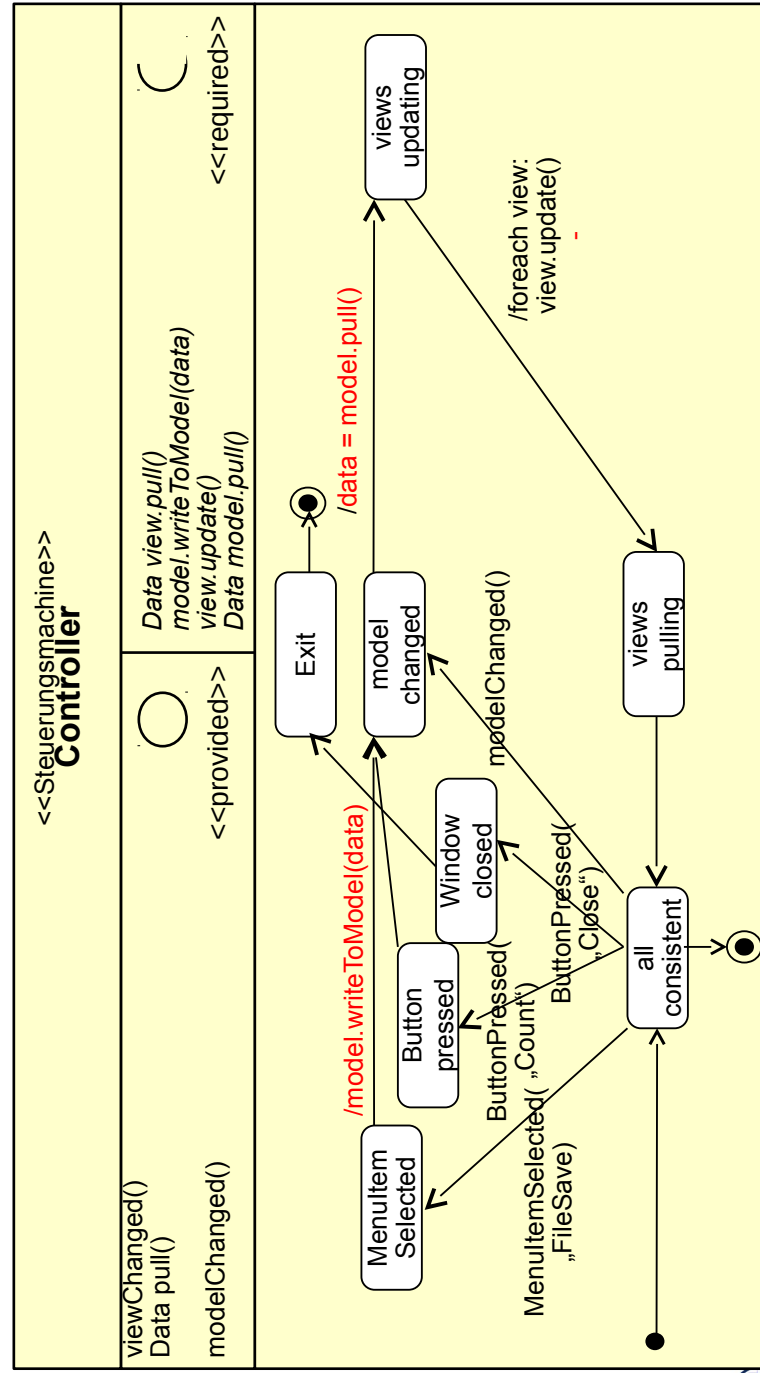
# Prinzipieller Aufbau von Controllern

- ▶ Die Steuerungsmaschine (hier kombiniert) steuert View und Model an ("beherrscht" sie)
- ▶ Getriggert wird sie durch die Ereignisse `viewChanged` und `modelChanged`



# Prinzipieller Aufbau von Controllern

- ▶ Die Steuerungsmaschine kennt viele verschiedene Ereignisse des UI und kann sie in spezifischen Zuständen behandeln



# Implementierung der Controller

- ▶ Die Controlerschicht wird als Kollaborationen realisiert
- ▶ Controller können kombiniert, oder auch als Paare von kommunizierenden Steuerungsmaschinen auftreten:
  - Der **Input-Controller** ist eine Steuerungsmaschine, die die Ereignisse auf der Fensterhierarchie in die Aufrufe an die Anwendungslogik übersetzt
  - Der **Output-Controller** ist eine Steuerungsmaschine, die die Ereignisse in der Anwendungslogik in die Aufrufe an die Fensterhierarchie übersetzt

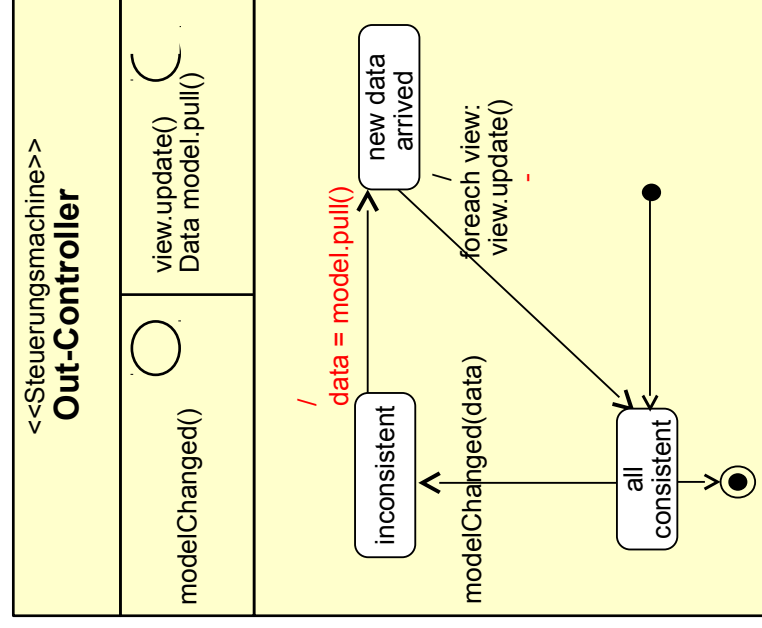
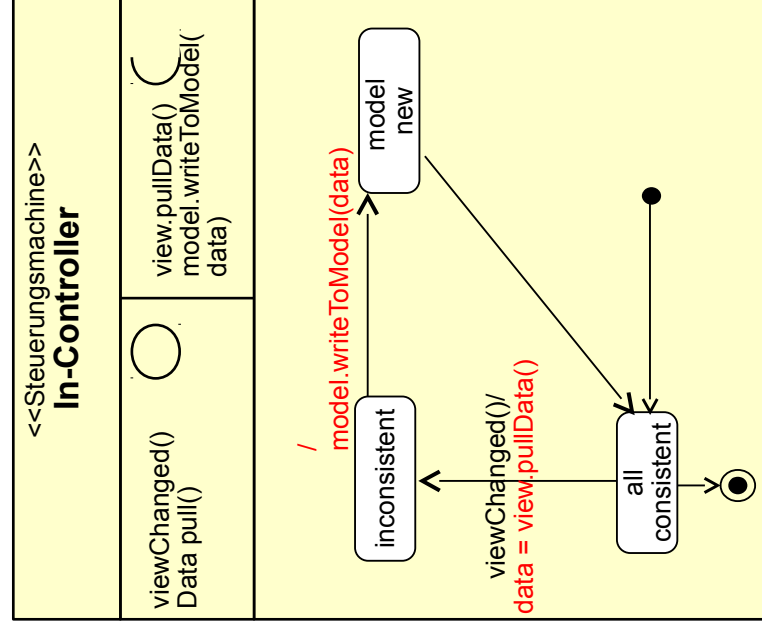
36



# Beispiele von Paaren von Controller-Steuerungsmaschinen

- ▶ Beide Steuerungsmaschinen stecken bilden Elemente einer oder mehrerer Kollaborationen

37



# 46.5 Implementierung mit Konnektoren

38



Softwaretechnologie, © Prof. Uwe Alsmann  
Technische Universität Dresden, Fakultät Informatik

## Ein MVC-Konnektor (Team)

```
class MVCconnector<Model,View,Controller>{
    List<myView> views;
    myModel model;
    myController controller;
    // Phase 1: creation of layers
    MVCconnector<View,Model,View,Controller>
    () {
        views = new ArrayList<myViews>();
        model = new myModel();
        connector = new myController();
    }
    class myView extends View {
        // Inherit the View methods
    }
    class myModel extends Model {
        // Inherit the Model methods
    }
}
```

```
class myController extends Controller {
    // phase 2:
    wireNet() {
        registerView(); registerModel();
    }
    registerView() { .. }
    registerModel() { .. }

    // Phase 3: dynamics
    run() {
        .. Controller state machines ..
    }
}
```

# Was haben wir gelernt?

40

- ▶ GUI-Programme laufen in 3 Phasen:
  - 1) Aufbau der Fensterfronten (widget hierarchies) durch Konstruktoraufrufe und Additionen (embodiment)
  - 2) Netzaufbau, Aufbau der Controller-Kollaborationen
    - Vorbereitung Play-Out: Anschluß des View-Reaktionscodes als jdk-Observer des Modells
    - Vorbereitung Play-In: Anschluß des Controller als widget-Observer der Views
  - 3) Reaktionsphase, bei der die Benutzeraktionen vom System als Ereignisobjekte ins Programm gegeben werden:
    - der Controller als Listener benachrichtigt und ausgeführt werden (Play-In)
    - die Views bzw. der Controller als Listener des Modells benachrichtigt werden (Play-Out)
- ▶ Der Kontrollfluß eines GUI-Programms wird *nie* explizit spezifiziert, sondern ergibt sich aus den Aktionen des Benutzers
  - Die Views reagieren auf Ereignisse im Screenbuffer, die von der Ablaufsteuerung gemeldet werden
  - Der Controller auf Widget-Veränderungen im View und Änderungen im Modell
  - Der Controller steuert alles

Prof. U. Almann, Softwaretechnologie, TU Dresden



# The End

41

- ▶ Diese Folien sind eine überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann, 2002. used by permission. Verbreitung, Kopieren nur mit Zustimmung der Autoren.

Prof. U. Almann, Softwaretechnologie, TU Dresden

