43. Architektur interaktiver Systeme
- der Controller als Konnektor zwischen GUI und Anwendungslogik

Prof. Dr. rer. nat. Uwe Aßmann

Institut für Software- und Multimediatechnik

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

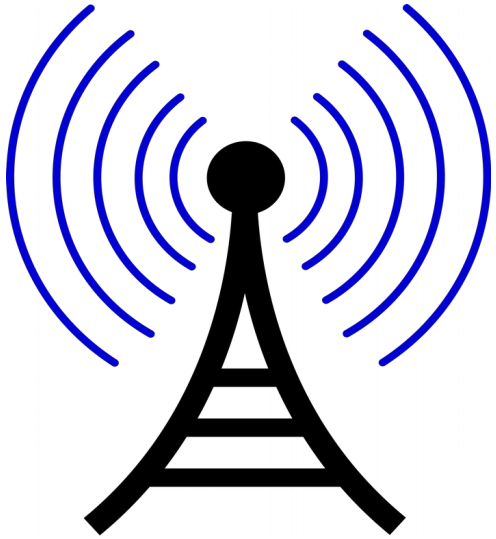
TU Dresden

Version 19-0.1, 08.07.19

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
6) Swing

Teil IV - Objektorientierter Entwurf (Object-Oriented Design, OOD)

1) 40: Überblick
2) 41: Einführung in die objektorientierte Softwarearchitektur
    1) Architekturprinzipien, Architekturstile, Perspektivenmodelle
    2) Modularität und Geheimnisprinzip
    3) BCD-Architekturstil (3-tier architectures)
3) 42: Verfeinerung mit querschneidender Objektanreicherung
4) **43: Architektur interaktiver Systeme**
5) 44: Punktweise Verfeinerung von Lebenszyklen
    - Verfeinerung von verschiedenen Steuerungsmaschinen



Literatur

- Obligatorisch:
  - [PassiveView] Martin Fowler. Passive View. http://www.martinfowler.com/eaaDev/PassiveScreen.html. Strikte Schichtung, aktiver Controller 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

Ziele

- Diese Vorlesung ist wichtig für das Winter-Praktikum, denn viele Gruppen müssen einen Swing-basierten Controller bauen
  - Swing ist ein komplexes Framework, das Einarbeitungszeit benötigt
  - GUI-Architekturen sind das Komplexeste, was wir bisher in dem Kurs besprochen haben – sie benötigen alle Kapitelinhalte

- 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
- Resourcen:
  - GUI/MVCModular.java
  - GUI/MVCModularDirectPlayOut.java

Die Bildung kommt nicht vom Lesen, sondern vom Nachdenken über das Gelesene.

Carl Hilty, 28.02.1833 - 12.10.1909

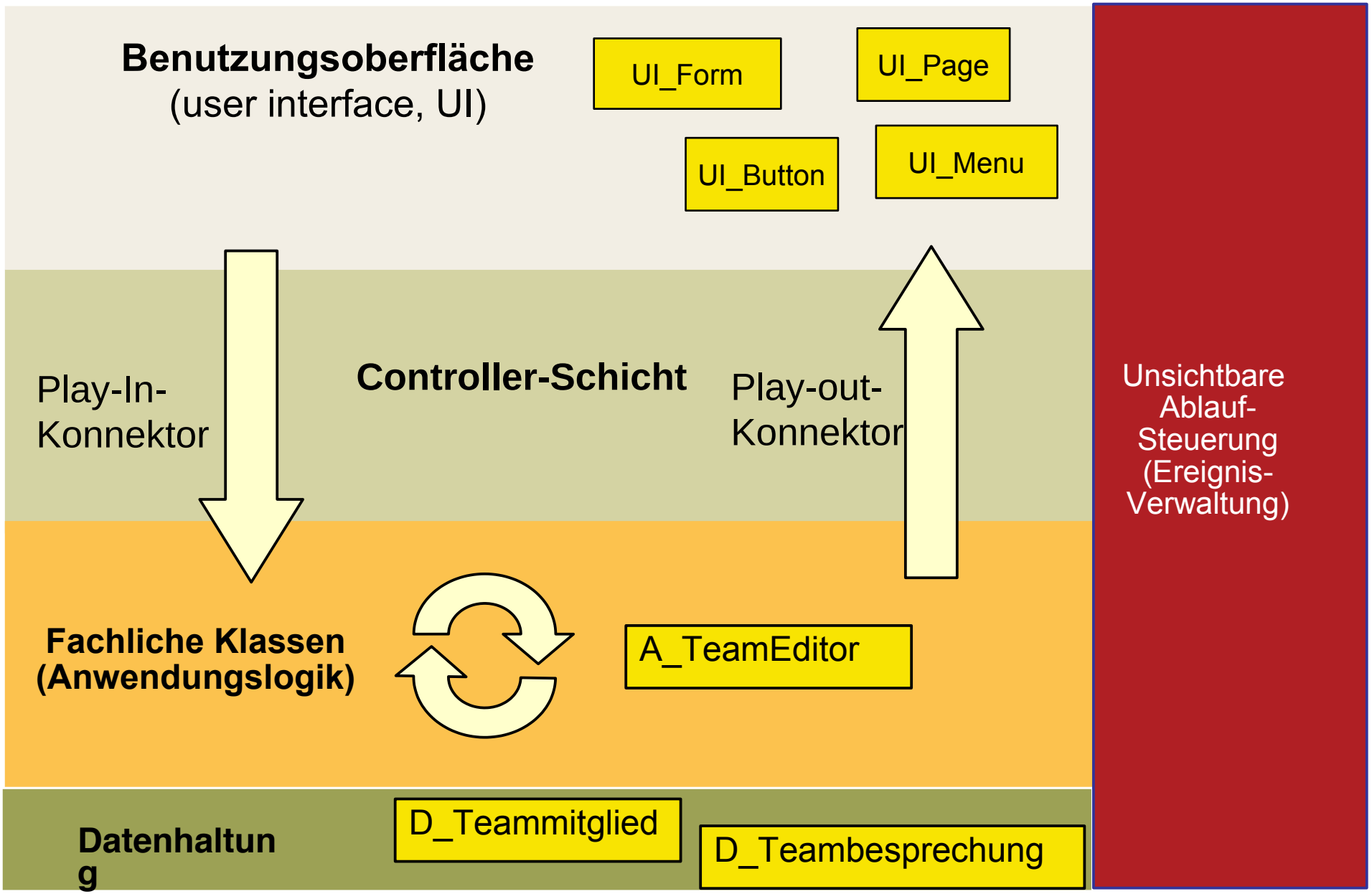
Schweizer Richter und Staatsrechtler, Buchautor und christl. Staatsrechts-Philosoph

Seine Bücher beeinflussten auch K. Adenauer

43.1 Benutzungsoberflächen (UI) und Anwendungslogik

Verschiedene Arten der Kopplung zwischen Benutzer und Software

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
  - Verteilte UI (Web UI)

    ==> dann muss der Controller die verteilte und parallele Verarbeitung steuern

- **Asynchrone UI:** die Anwendungslogik reagiert auf die UI
- Treiber ist die UI
  - Graphische UI (GUI)
  - Tangible UI (TUI)

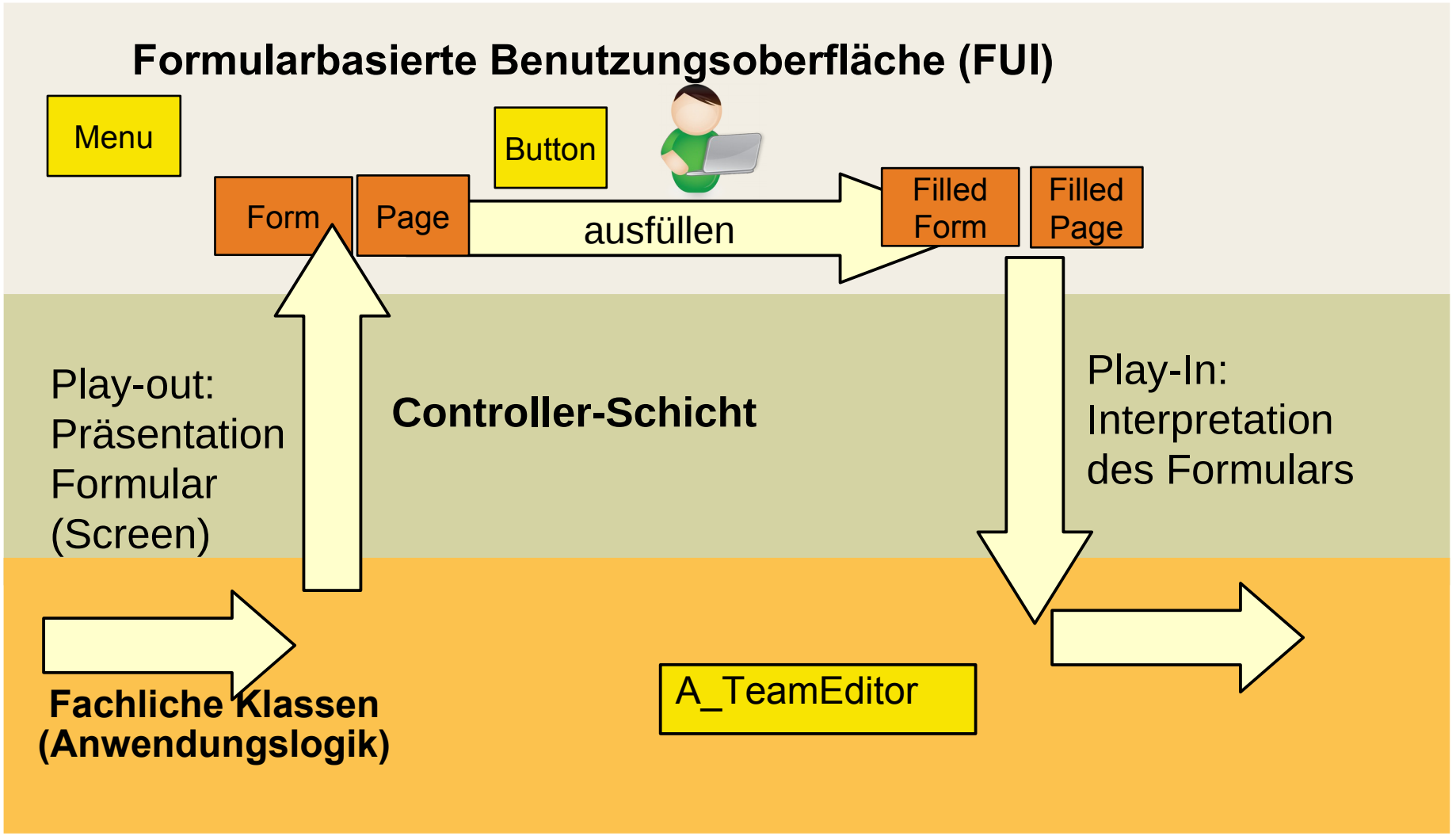
    ==> dann muss der Controller die parallele Verarbeitung steuern

43.2 Kopplung von synchronen Benutzeroberflächen mit der Anwendungslogik

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

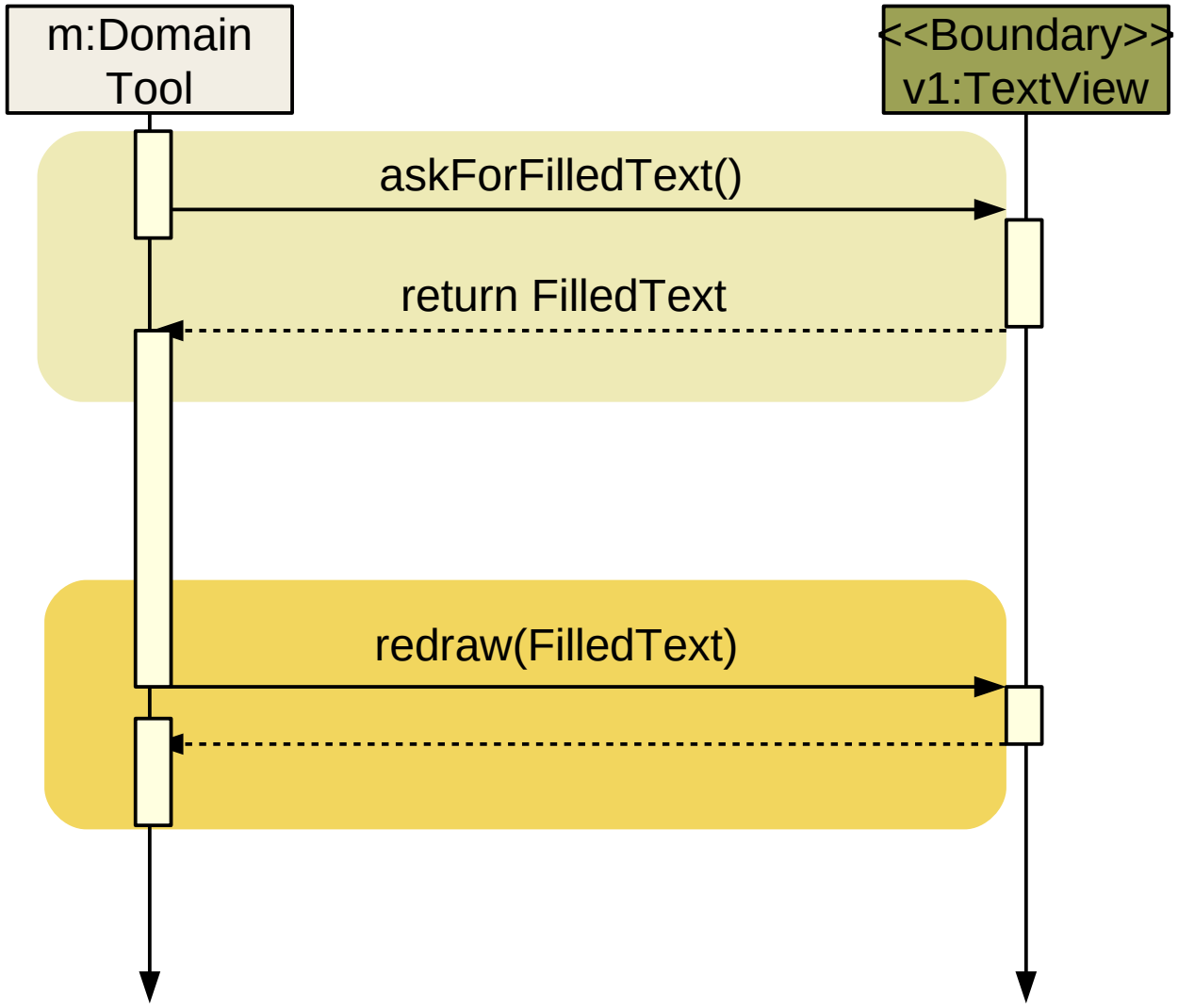
Synchrone Kopplung zwischen Anwendungslogik, Controllerschicht und FUI

- Die Anwendungslogik ruft das formularbasierte UI mit einem leeren Formular auf und warten auf das Ausfüllen des Benutzers (synchron)
- Die Play-In und Play-Out-Konnektoren sind besonders einfach



43.2.1. Textbasierte UI mit synchronem Update (*ein* View)

- In Java: Eingabe mit System.in, Ausgabe mit System.out
- Play-In und Play-Out-Konnektoren sind Prozeduraufrufe an die UI



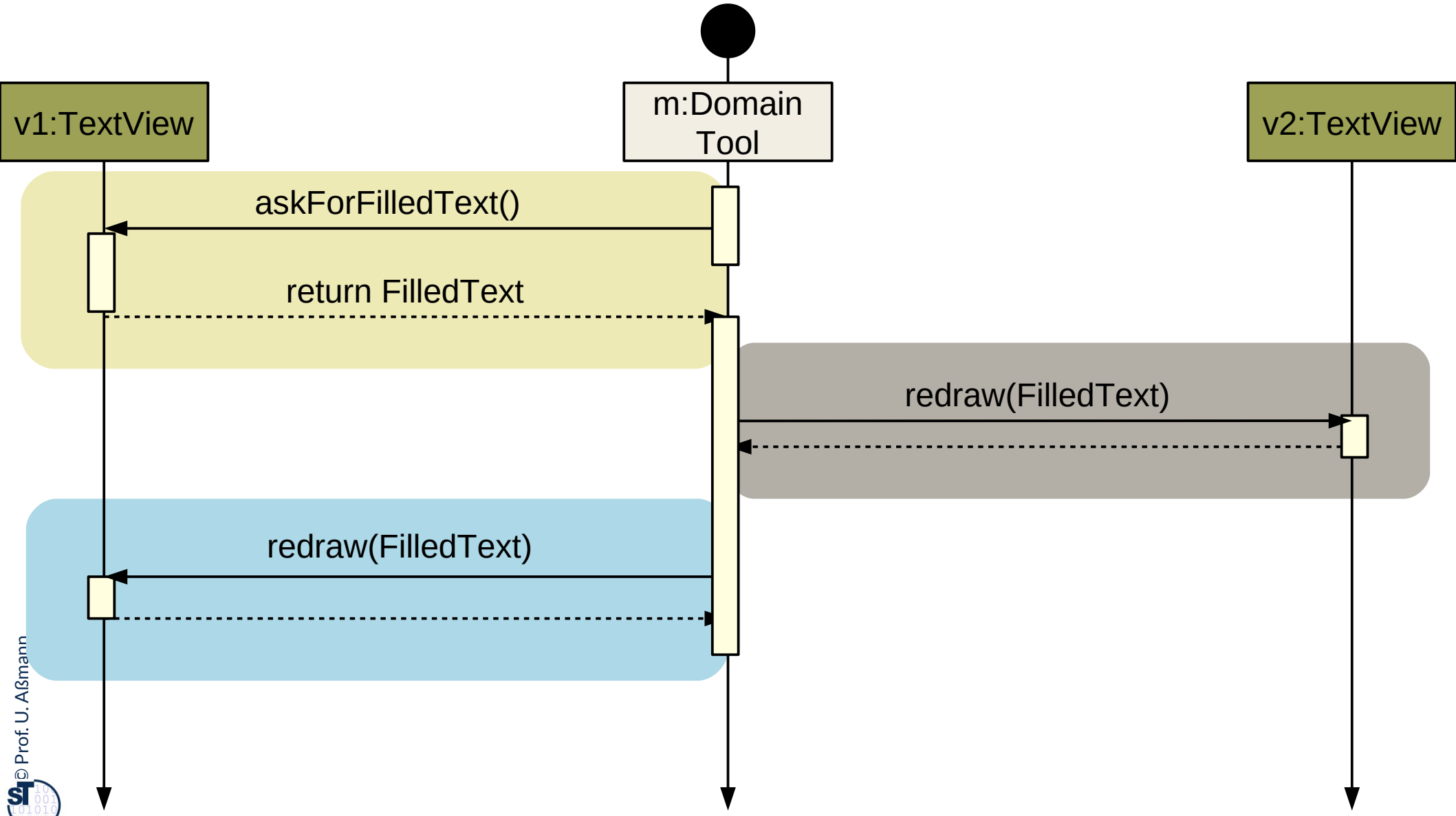
Einfache textuelle Sichten

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

```
class PersonTool {

  … activities of the tool …

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

  Person p.number = num;

  foreach (view ; tool.getViews()) {
    view.redraw(p);
  }

  … further activities of the tool …
}
```

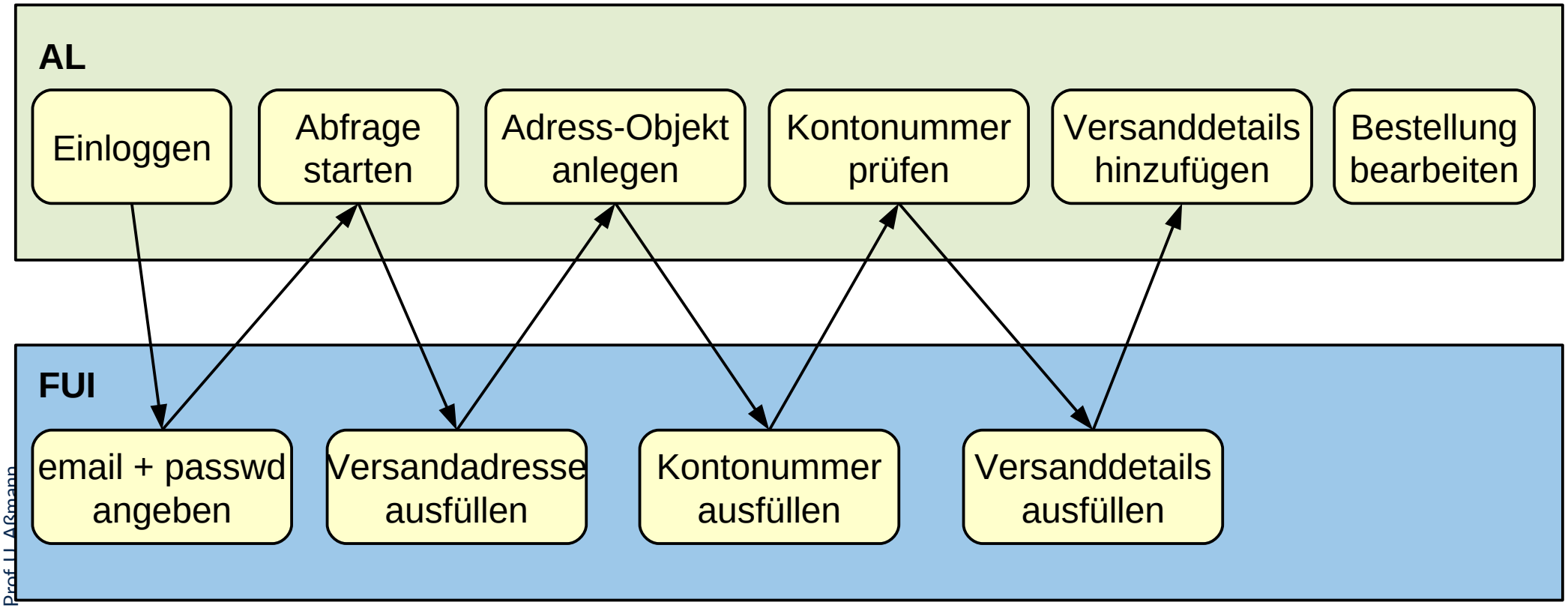
Textbasierte UI mit synchronem Update (mehrere Text-Views)

- Immer alles schön nacheinander (synchron)



Screen-Flow

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

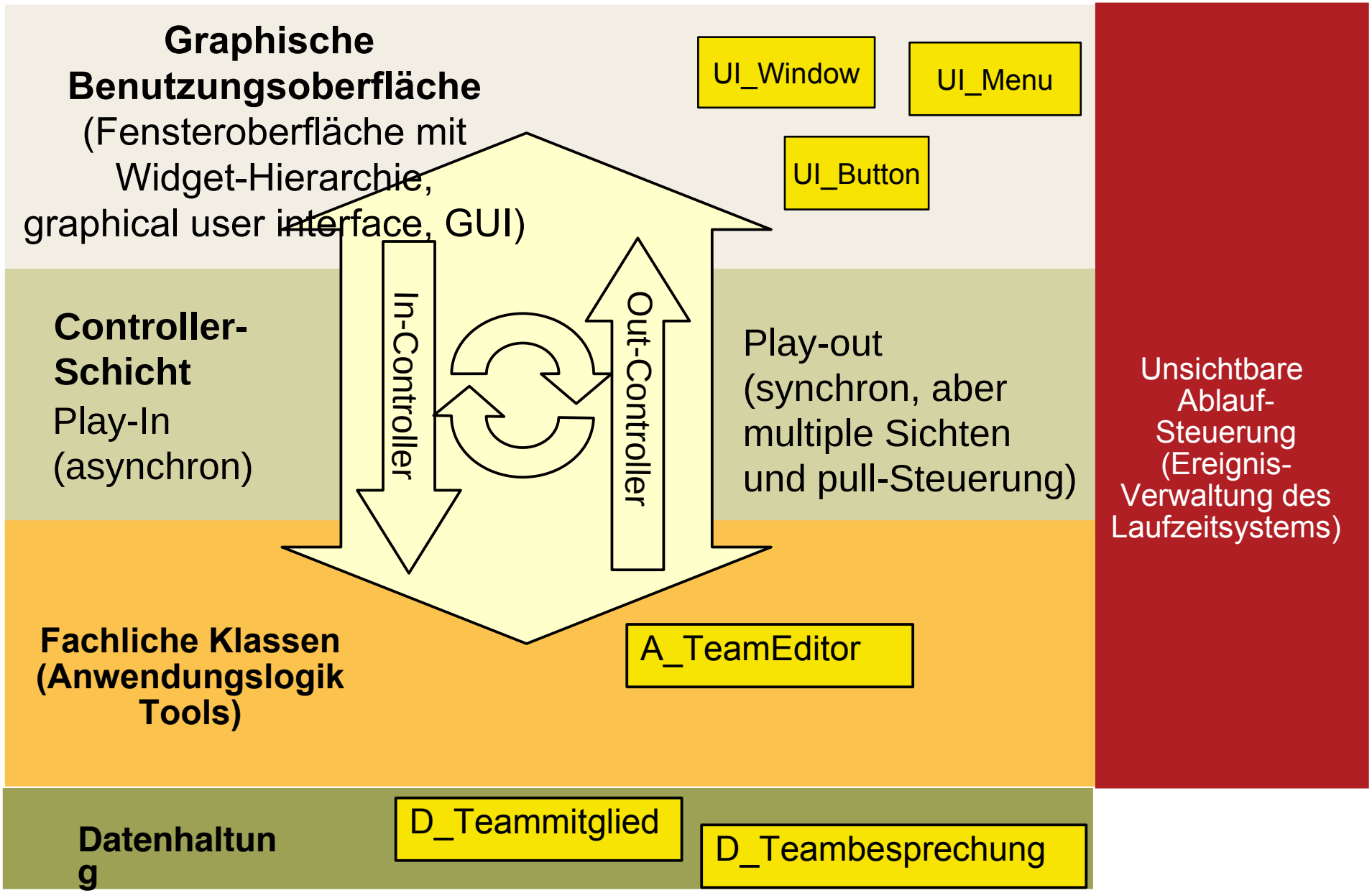


43.3 Überblick zu reaktiven graphischen Benutzeroberflächen (GUI)

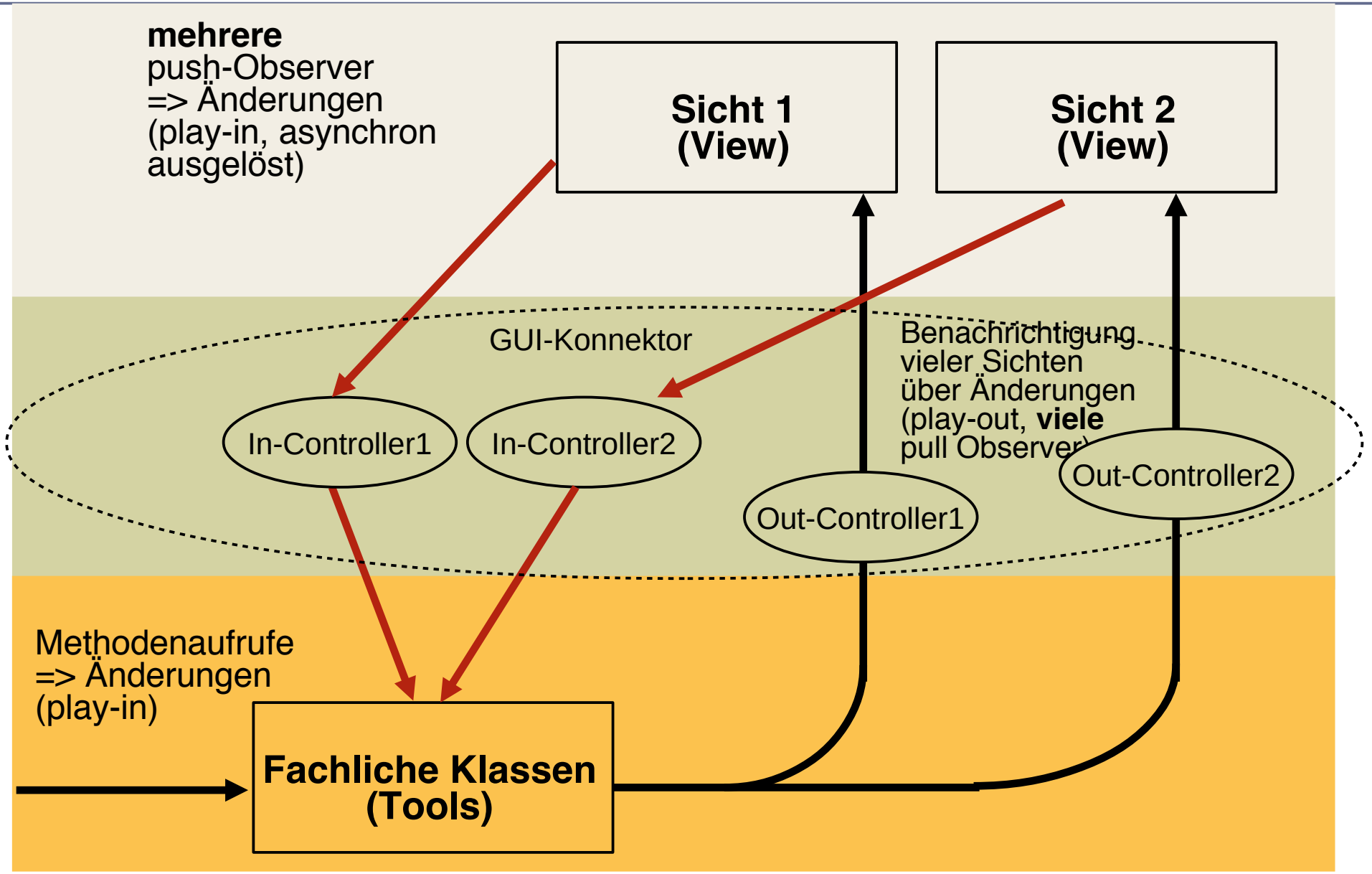
Kopplung der GUI und Anwendungslogik durch Controller

- Bislang war es einfach, aber auch unflexibel
- Im Normalfall ist es schwieriger, denn da bringt ein *Controller* bzw. eine *Controllerschicht* die Ereignisse, "auslösenden" Fensterelemente (Sicht) und Tool asynchron zusammen
  - Der Controller beherrscht und kapselt die Interaktion, die Initiative geht von ihm aus
  - View und Tool sind gegenüber ihm passiv

Schichtenarchitektur der reaktiven Benutzungsoberfläche (GUI)

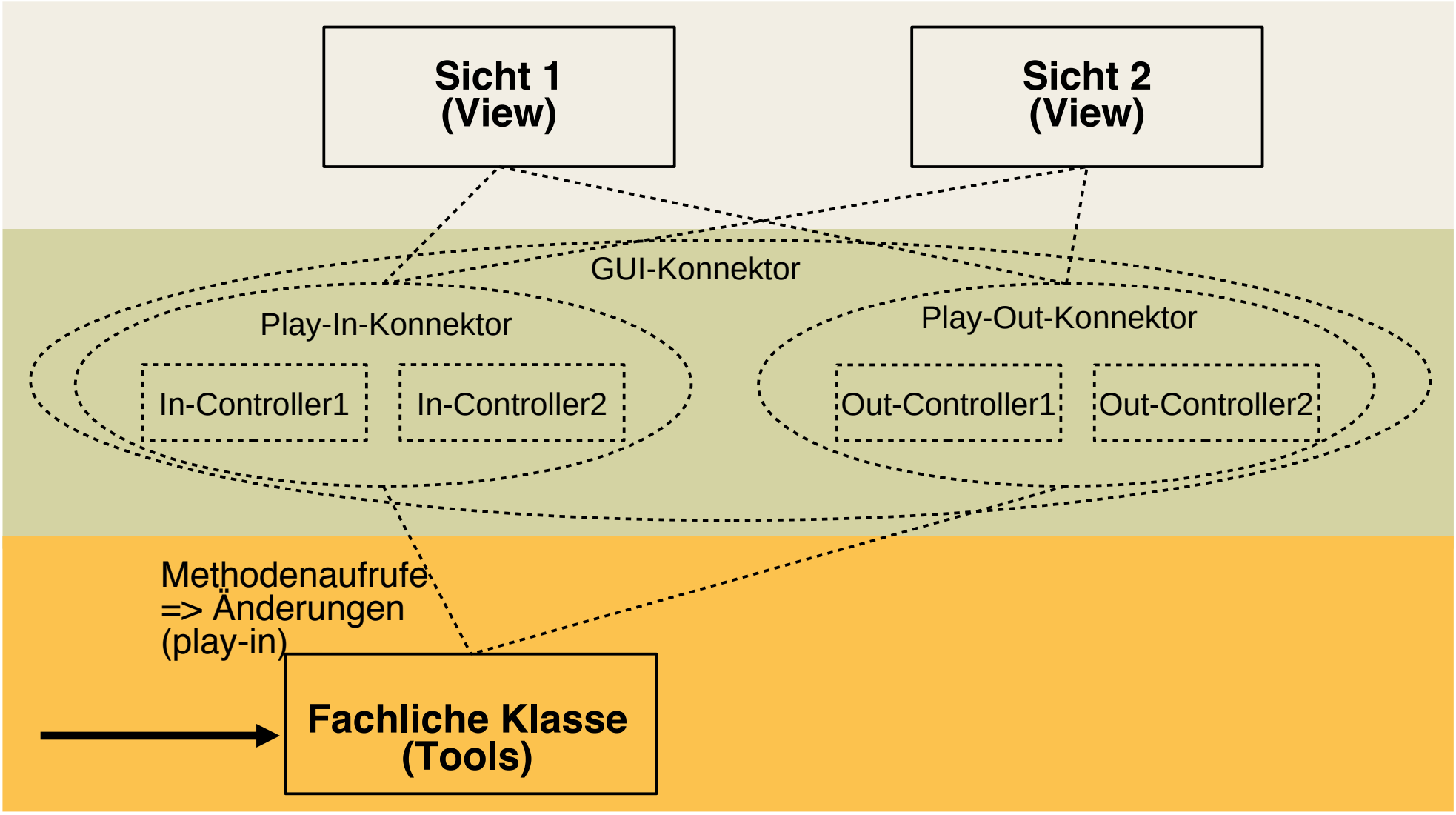


Tool, Controller und Views in strikter Schichtung



Controller sind Konnektoren zwischen Tool und View

- Meist existiert ein Hauptobjekt in der Kollaboration, womit der Controller einen Konnektor darstellt



43.4 Controller als Steuerungsmaschinen in Konnektoren

- Im Entwurfsmuster “PassiveView” bestehen die Controller aus Steuerungsmaschinen, die die Ereignisse der GUI in die Ereignisse der Anwendungslogik übersetzen und umgekehrt
- Die Controllerschicht ist aktiv; View und Tool bleiben passiv

Implementierung der Controller als Steuerungsmaschinen

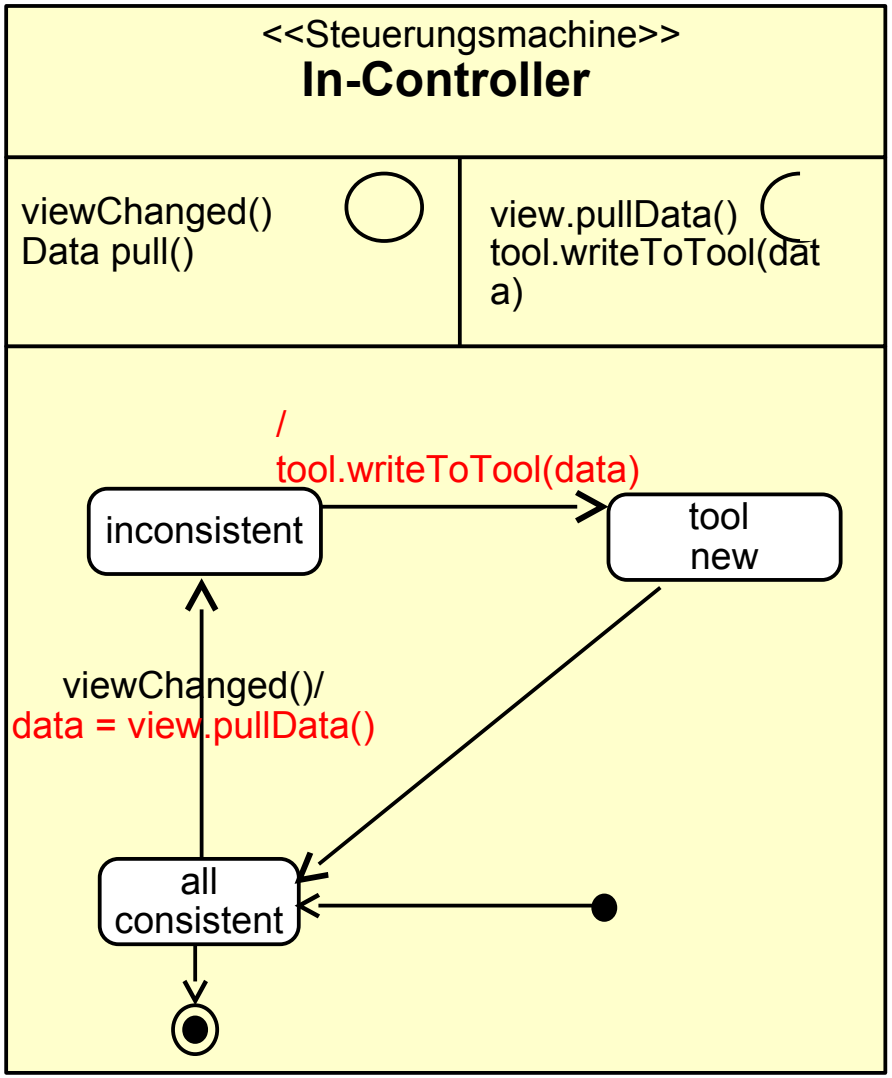
Ein Controller-Konnektor wird durch eine Steuerungsmaschine implementiert, die die Ereignisse auf der Fensterhierarchie (UI) in die Aufrufe an die Tools der Anwendungslogik *übersetzt* (u.u.)

- 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.

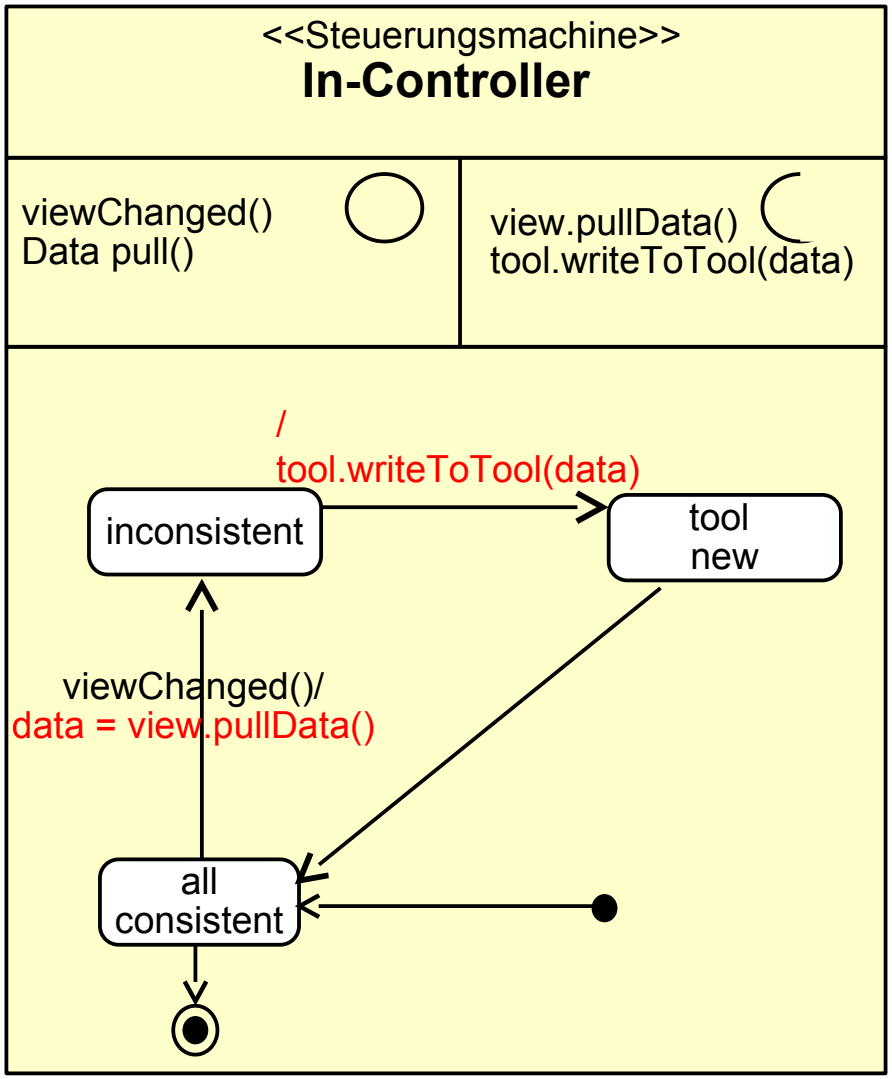
Input-Controller mit drei Zuständen

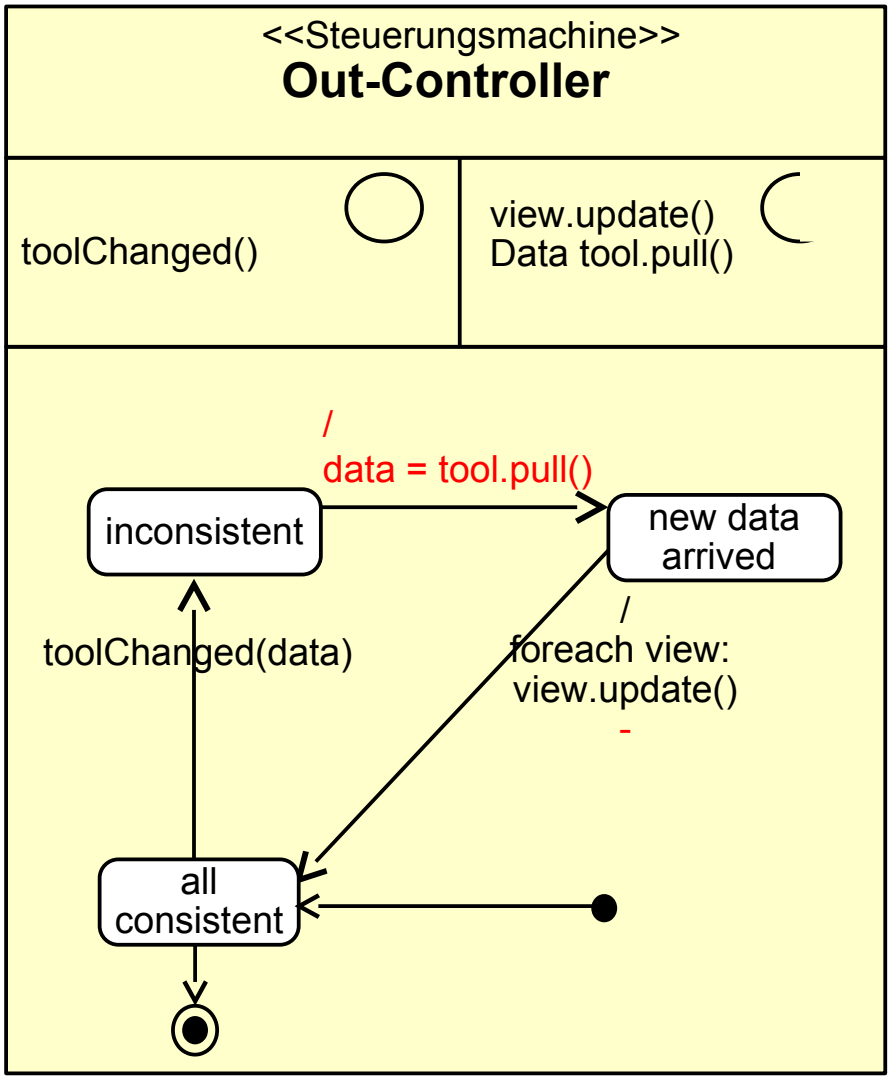
- Die entstehende Steuerungsmaschine steuert View und Tool an (“beherrscht” sie)
- Getriggert wird sie durch die Ereignisse viewChanged (update). Sie löst pullData und writeToTool aus



Variante 1: Paare von Controller-Steuerungsmaschinen

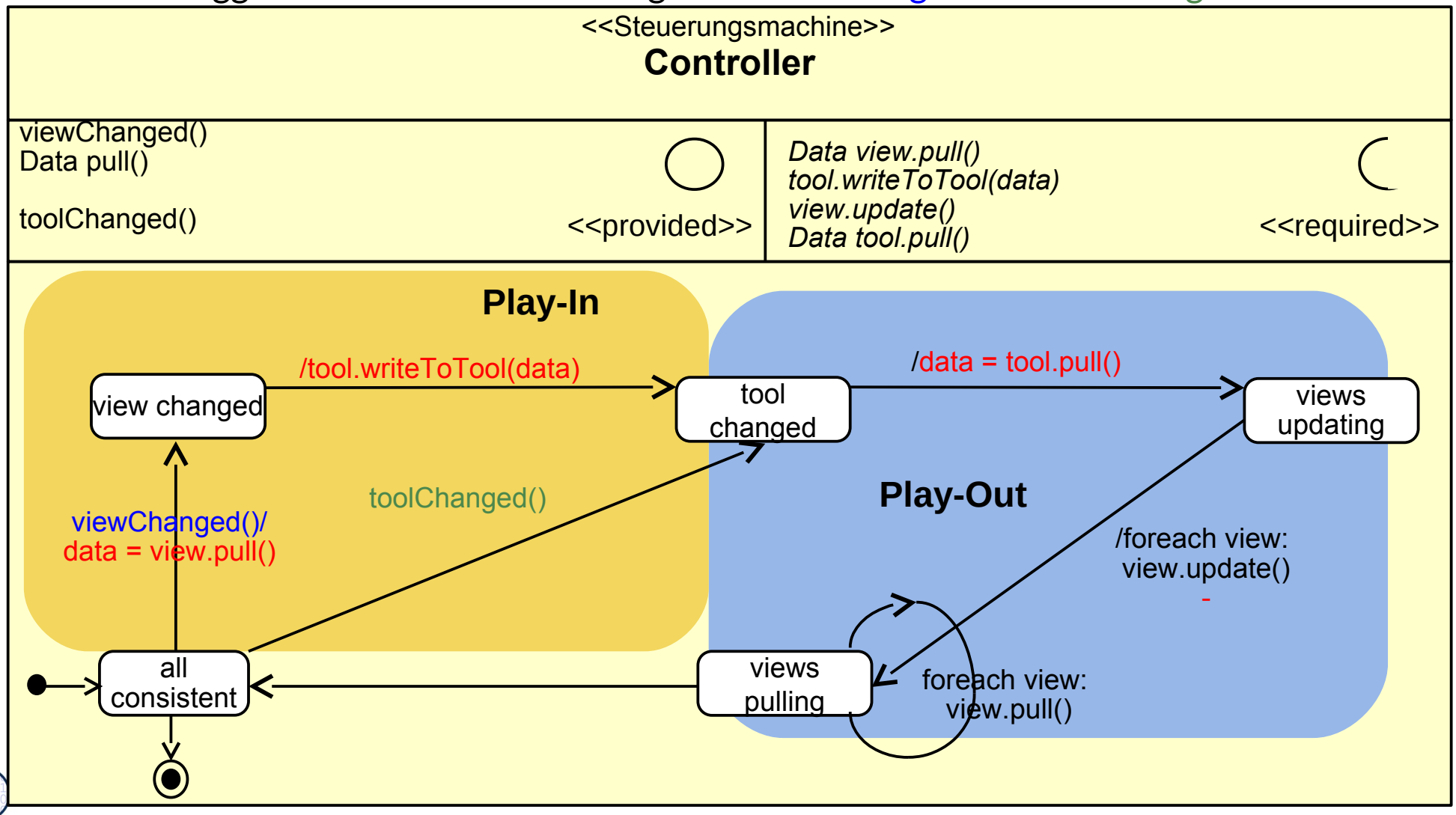
- In- und Out-Steuerungsmaschinen bilden Elemente einer oder mehrerer In- und Out-Kollaborationen





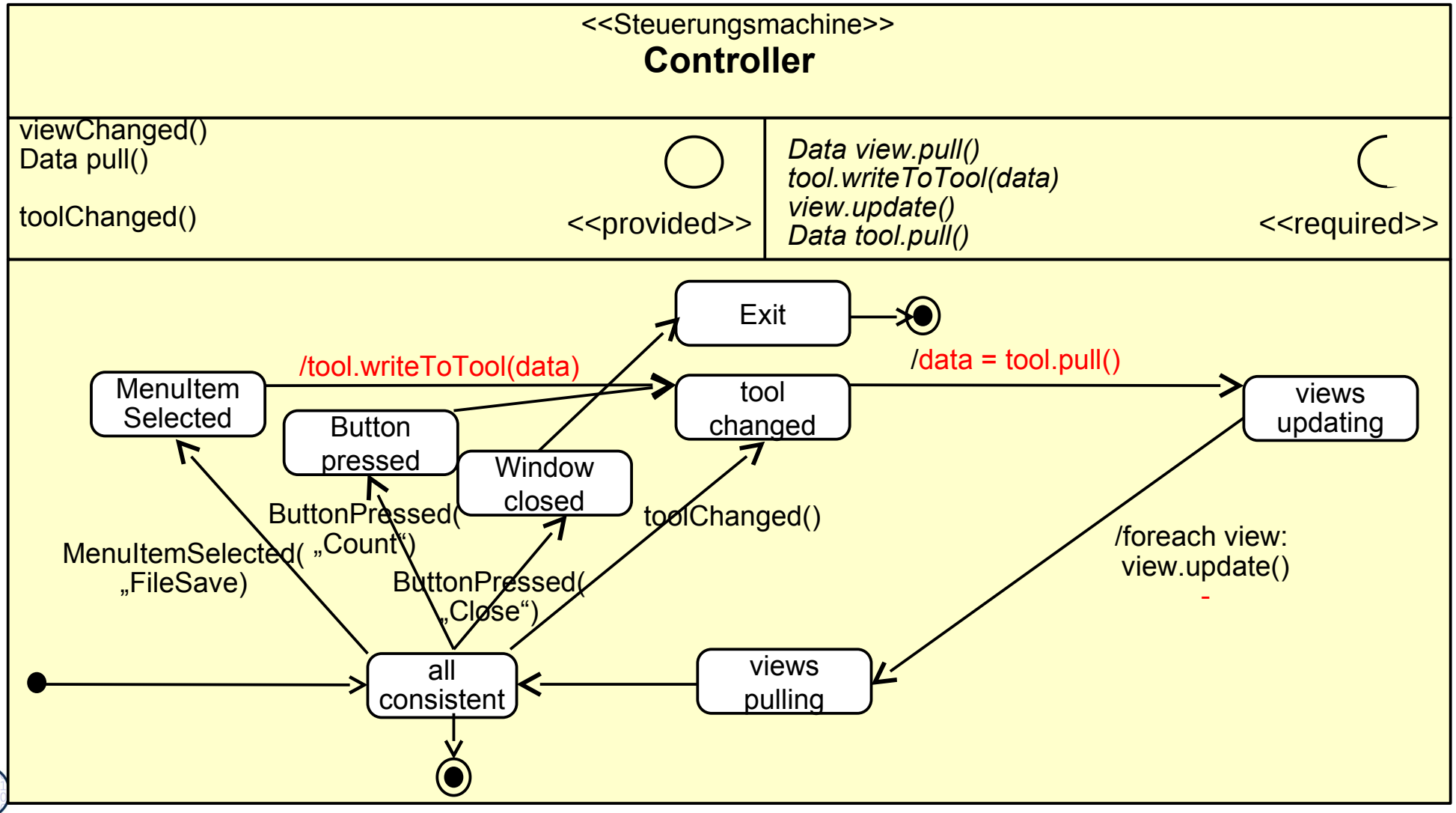
Variante 2: Controller als bidirektionale Inout-Konnektoren

- In- und Out-Controller können auch *zusammengelegt* sein (z.B. in Spring)
- Die entstehende Steuerungsmaschine steuert View und Tool an (“beherrscht” sie)
- Getriggert wird sie durch die Ereignisse viewChanged und toolChanged



Variante 2b: Prinzipieller Aufbau von InOut-Controllern

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



Implementierung der Controller

- Die Controllerschicht wird realisiert entweder als
  - Konnektor mit einer Steuerungsmaschine
  - Menge von In-/Out-Konnektoren mit kommunizierenden Steuerungsmaschinen
  - Oder einer Menge von bidirektionalen Konnektoren
- Controller können kombiniert (InOut-Controller), 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

43.5 Implementierung mit Konnektoren

Ein einfacher MVC-Konnektor (als Team mit inneren Klassen)


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

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


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

MVCModular.java

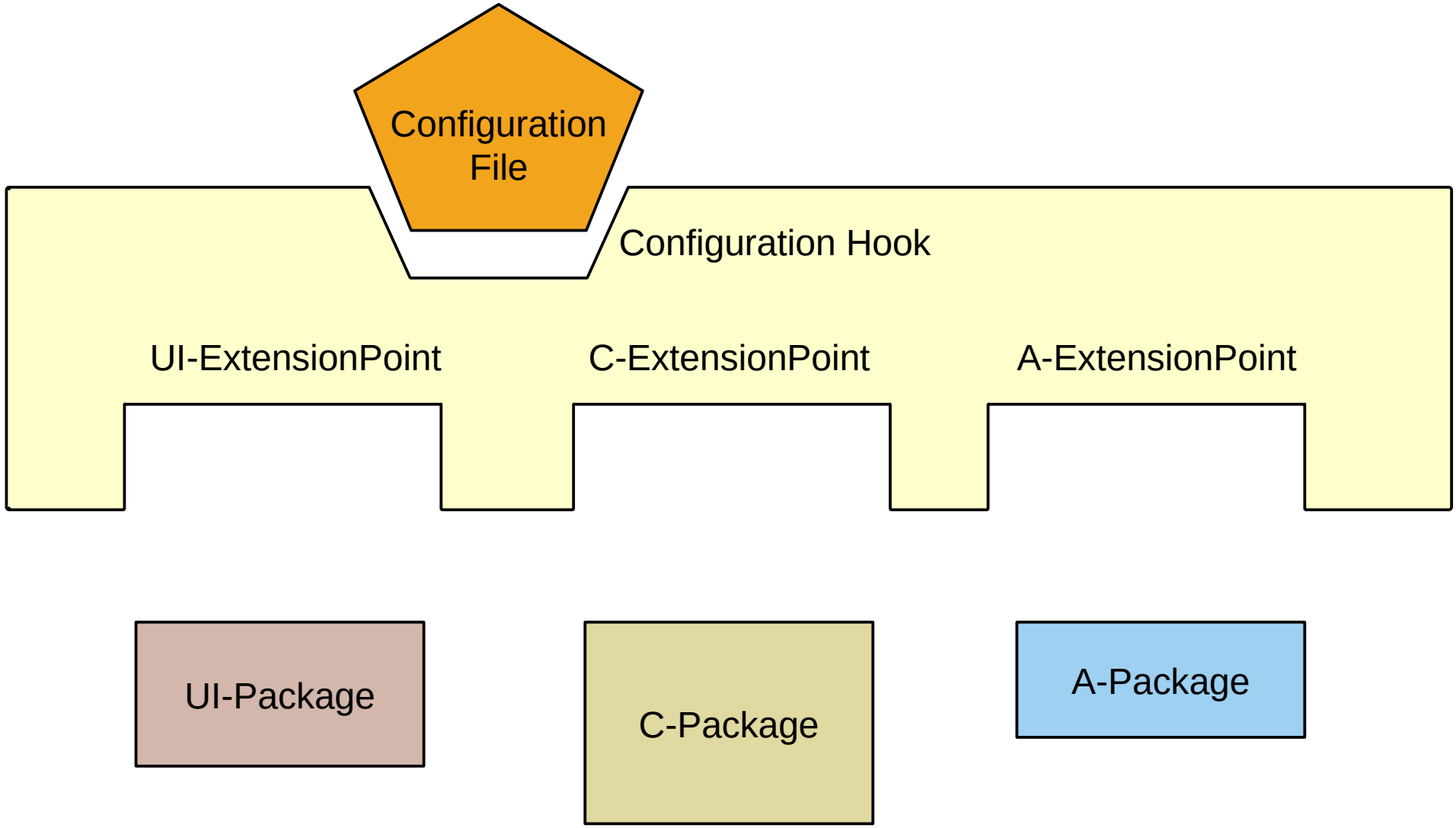
43.6. MVC Frameworks

- (Controller Frameworks)

MVC-Frameworks mit vorgefertigten GUI-AL-Konnektoren

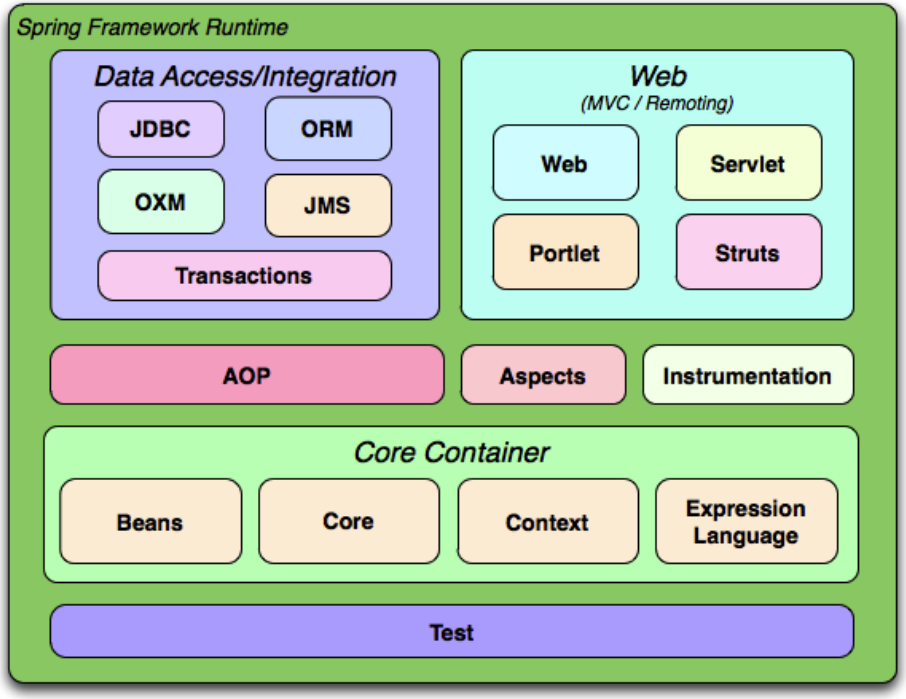
- Die Struktur einer Controllerschicht kann sich von Anwendungsklasse zu Anwendungsklasse sehr unterscheiden.
- Ein **MVC-Framework** definiert einen GUI-AL-Konnektor und gibt eine Struktur der Controllerschicht vor, definiert Protokolle für die Ereignismeldung und den Datenaustausch vor und kann durch den Entwickler erweitert werden.
  - MVC Frameworks benötigen Konfiguration und "Plugins"
  - **Oft folgt man dem Prinzip "Convention over configuration":** Konventionen über Dateiverzeichnisse und Konfigurationsdateien vereinfachen dem MVC-Framework das Auffinden von Controller-, View-, Anwendungsklassen, sowie Hinweise zu ihrer Verdrahtung
  - Konfigurationsdateien meist in XML oder Java-Property-Lists
- Berühmte Beispiele:
  - Java: Spring, Struts
  - Ruby: Ruby on Rails
  - Groovy: Grails

MVC Frameworks kennen “Plugins”



Spring Framework

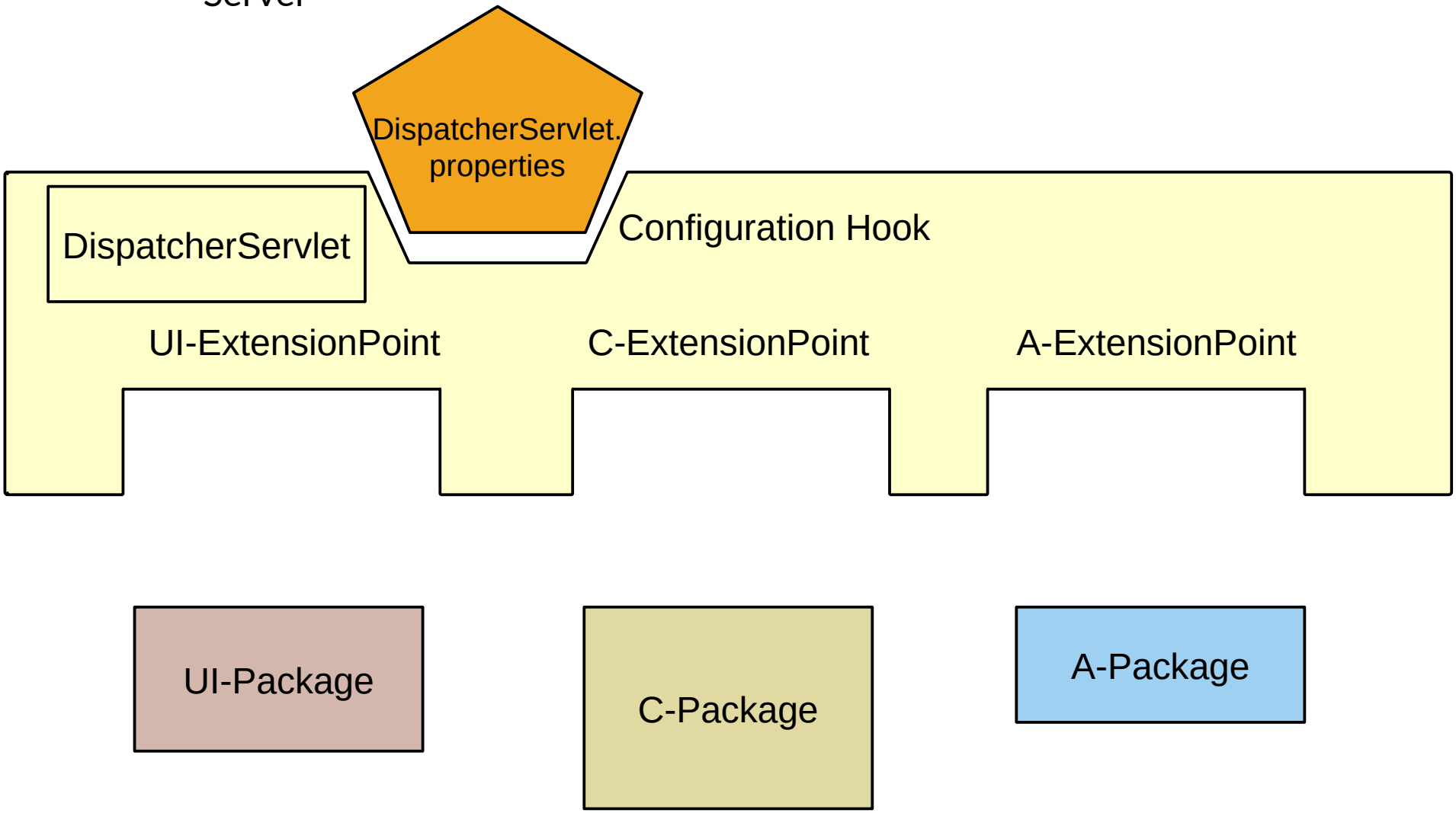
- *Spring* ist das im Praktikum im WS verwendete MVC-Framework
  - Webbasiert, d.h. Controllerschicht ist auf Client und Server verteilt implementiert
  - Konfigurierbar durch XML-Dateien und Java Property Files
  - Erweiterbar
- Das Salespoint-Framework nutzt als Konnektor zum GUI das Konnektor-Framework SPRING
  - Main controller, subcontroller
  - Web-MVC Frameworks brauchen *starke Schichtung*
  - Bietet sehr viele verschiedene Pakete, nicht nur für Web-UIs



- http://spring.io/guides

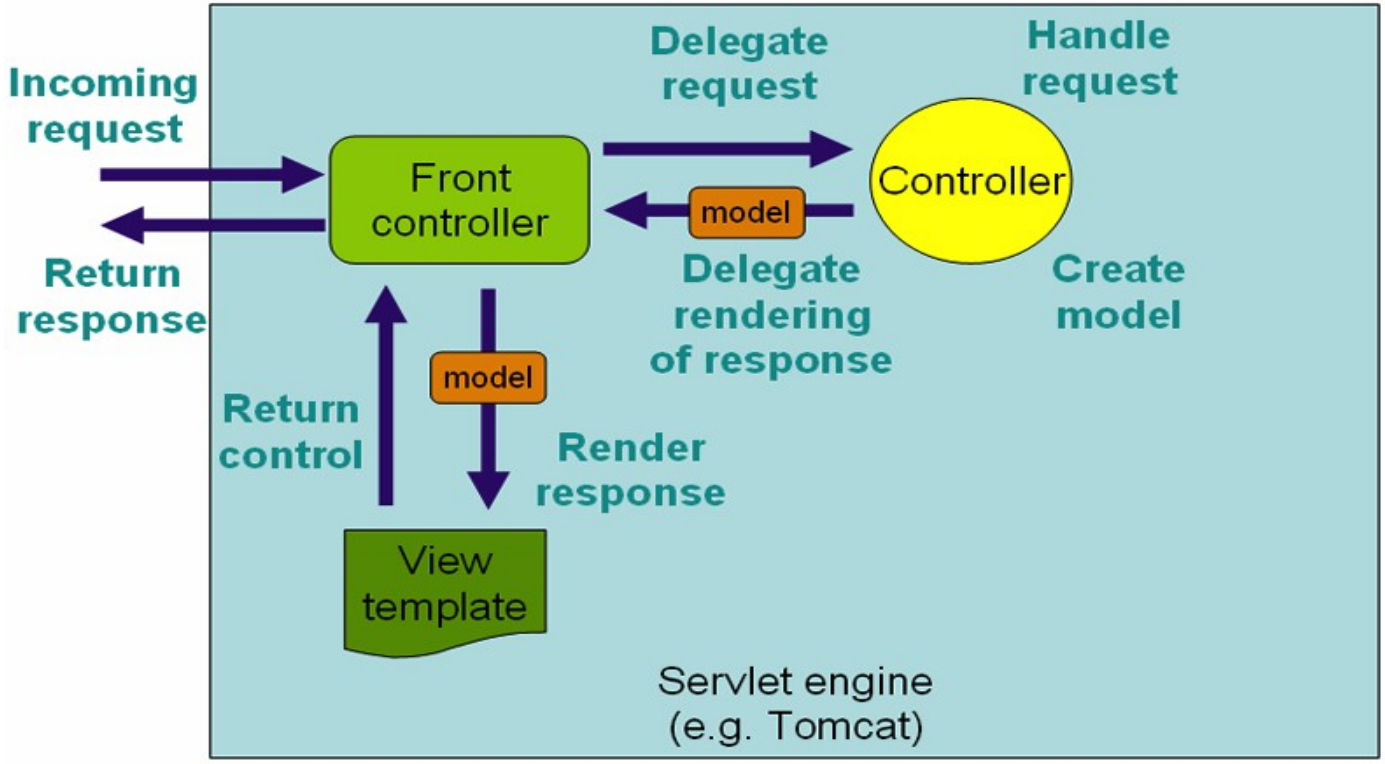
Spring Konfiguration

- Spring übernimmt das Management der Verteilung
  - Das Zusammenspiel zwischen Browser, Server und Anwendungslogik auf dem Server



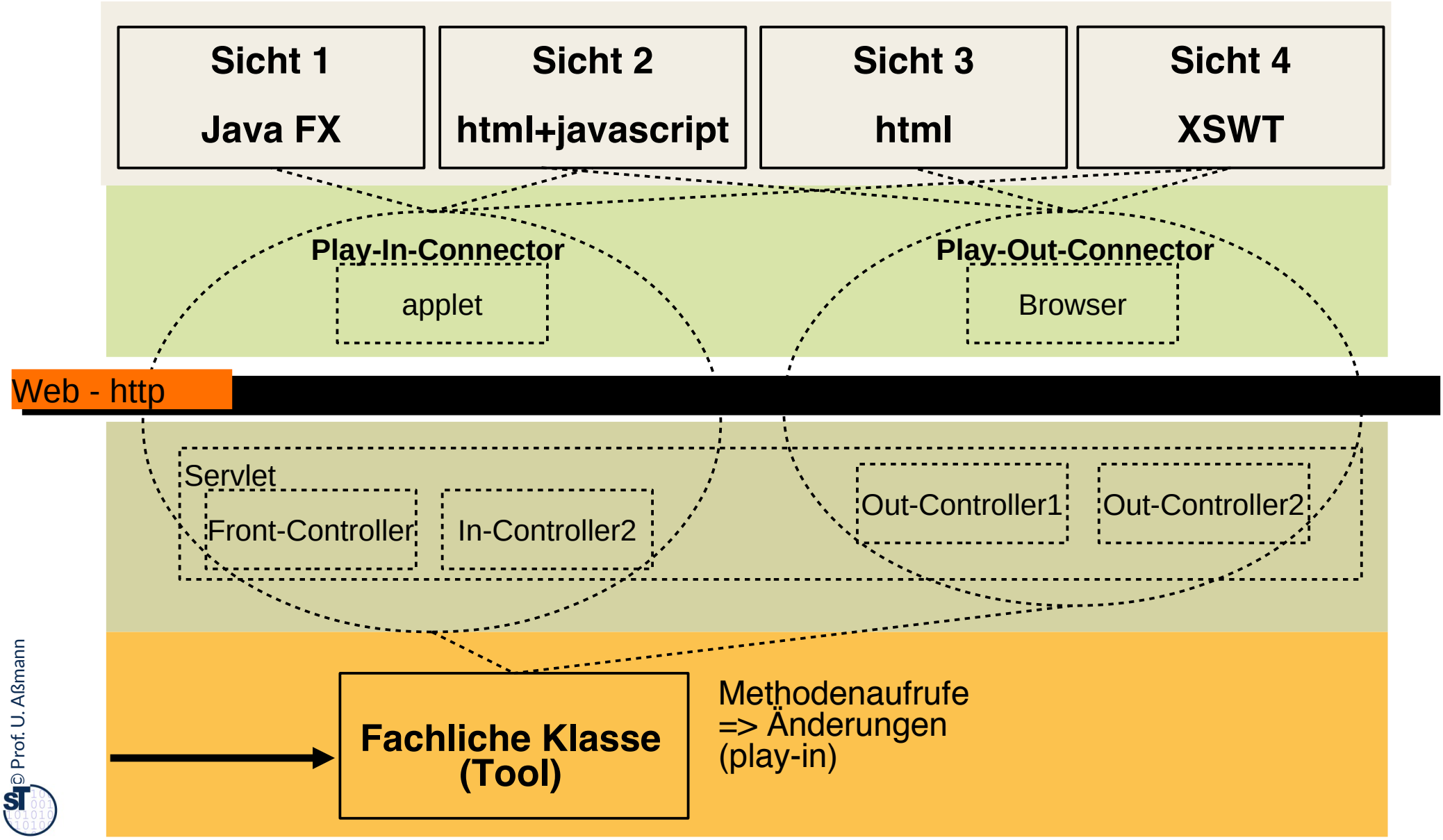
Struktur des String Controllers in Web- Systemen (Server Side)

- Der Spring Controller ist ein komposites Programm auf dem Server:
  - das Spring-DispatcherServlet enthält einen “FrontController”, der das ankommende Ereignis interpretiert (Steuerungsmaschine) und an untergeordnete Controller bzw. Steuerungsmaschinen weiter leitet



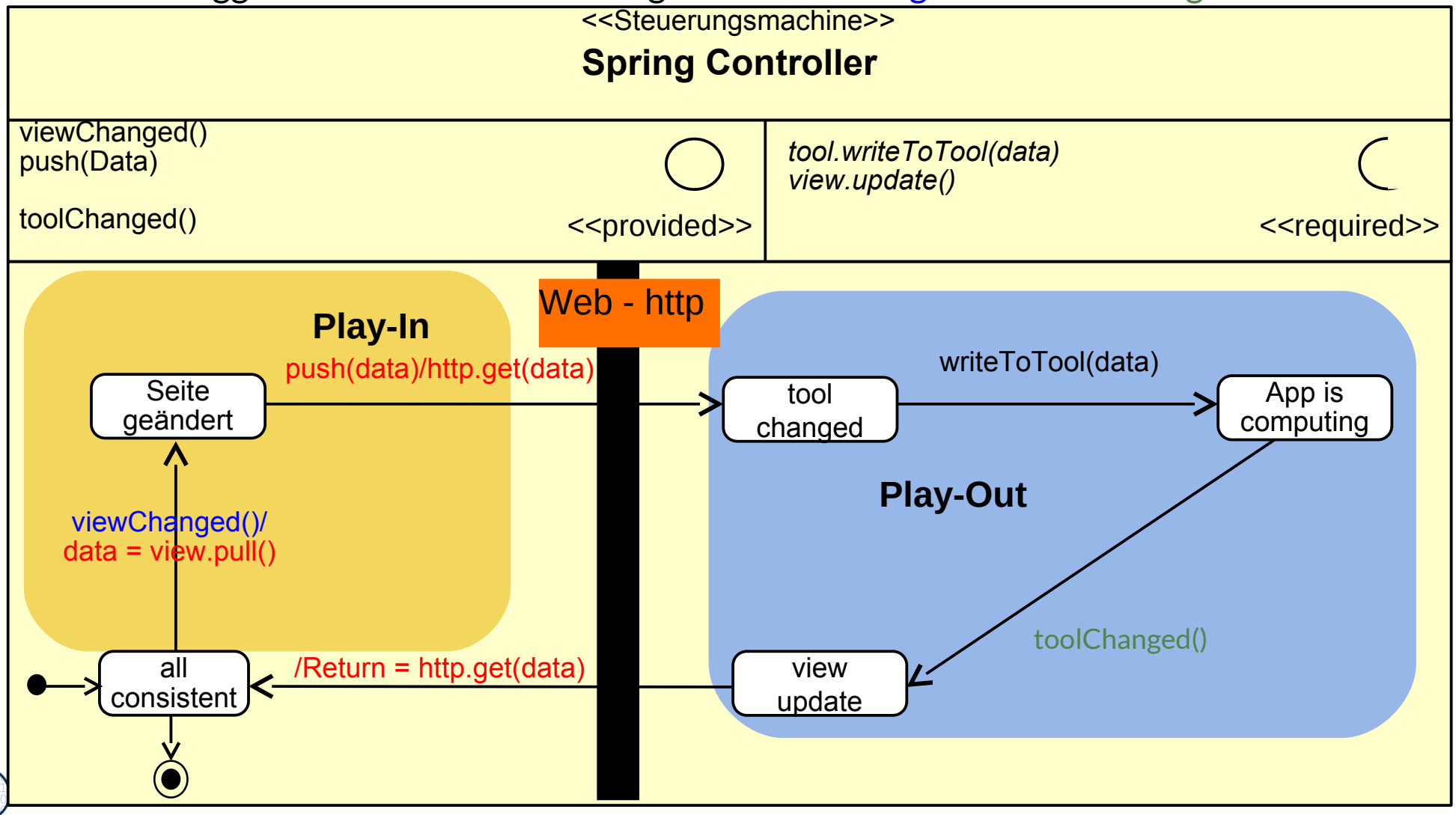
Controller sind Konnektoren zwischen Tool und View

- Im Folgenden gibt es ein Hauptobjekt, den Konnektor, der View, Controller und Tool verdrahtet



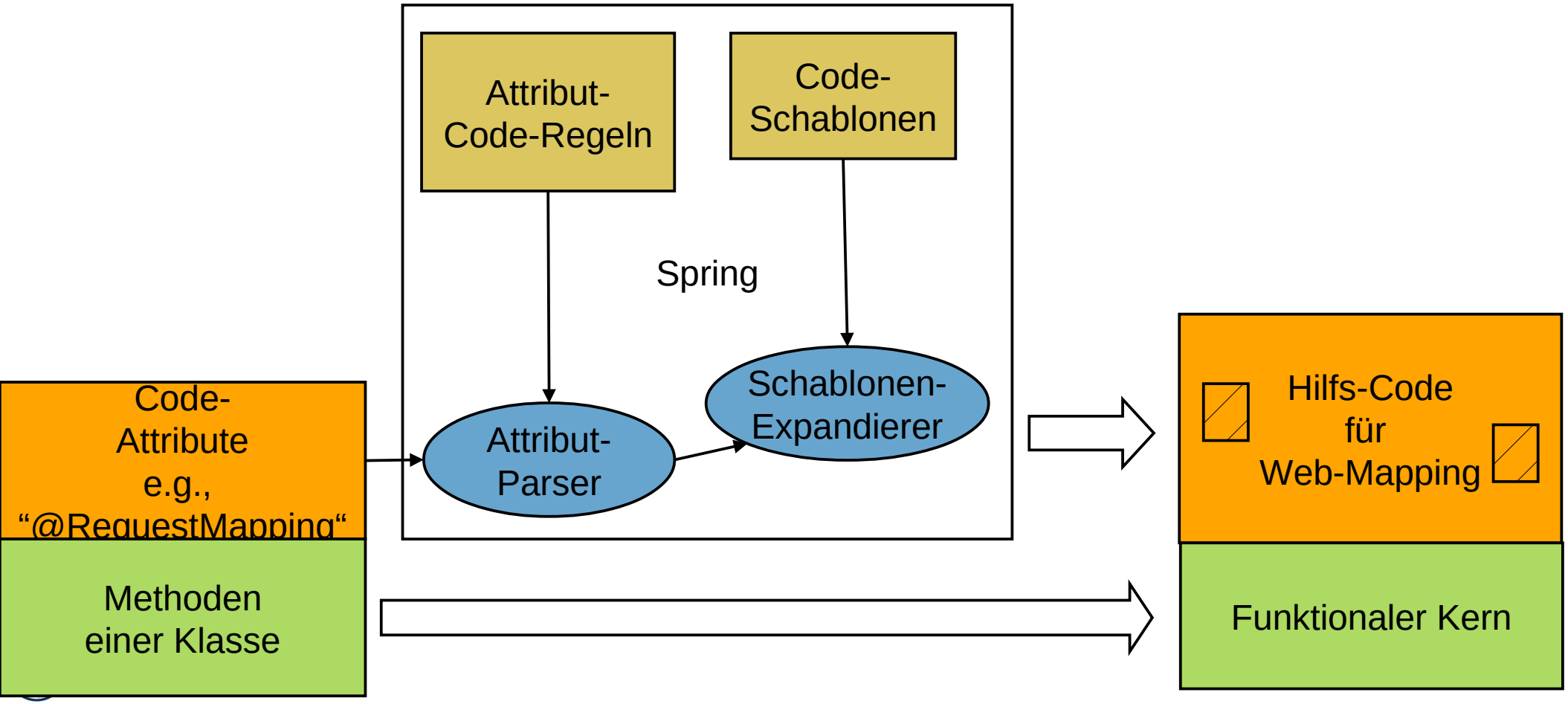
Server-seitiger Spring-Controller

- In- und Out-Controller können auch ***verteilt*** sein (z.B. in Spring)
- Die Steuerungsmaschine ist verteilt auf client und server
- Getriggert wird sie durch die Ereignisse viewChanged und toolChanged



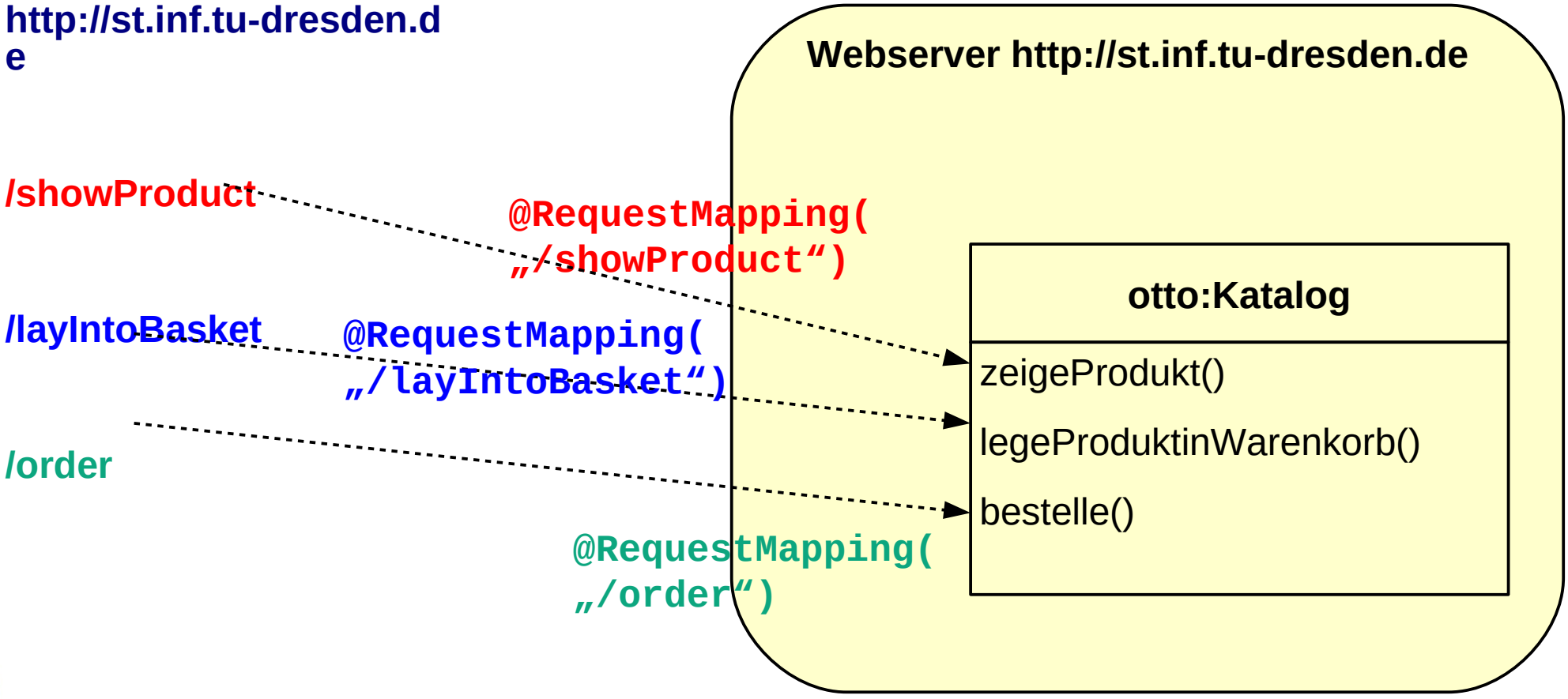
Spring nutzt @attribut-basierte Codegenerierung

- Spring wandet Java-@Attribute (sog. Metadaten) in Code um
- Attribute parameterisieren Schablonen (“templates”): Template-gesteuerte Codegenerierung
- Siehe auch Xdoclet, xdoclet.sf.net



@RequestMapping

- Ein **REST-Webservice** bildet URL (Web-Dateinamen) auf *aktive Methoden eines Webservice-Objekts* ab
  - `@RequestMapping("<relative URL>")`
- Wird die URL im Browser aufgerufen, wird die Methode aufgerufen und ihr Resultat als String im JSON-Format zurückgegeben



Was haben wir gelernt?

- GUI-Programme koppeln die GUI mit der Anwendungslogik mit Hilfe des Konnektor-Musters
  - Der Controller-Konnektor aktiviert die Views und die Anwendungs-Tools
- 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 Tool
  - Der Controller wird als Steuerungsmaschine implementiert und steuert alles (aktiver Konnektor)
- Das MVC-Framework Spring enthält eine stark geschichtete GUI-Anwendungskopplung
  - Enthält einen kompositen Controller (komposite Steuerungsmaschine)
  - Regelt den Verkehr zwischen Browser, Server, Servlet, Webservice-Methoden

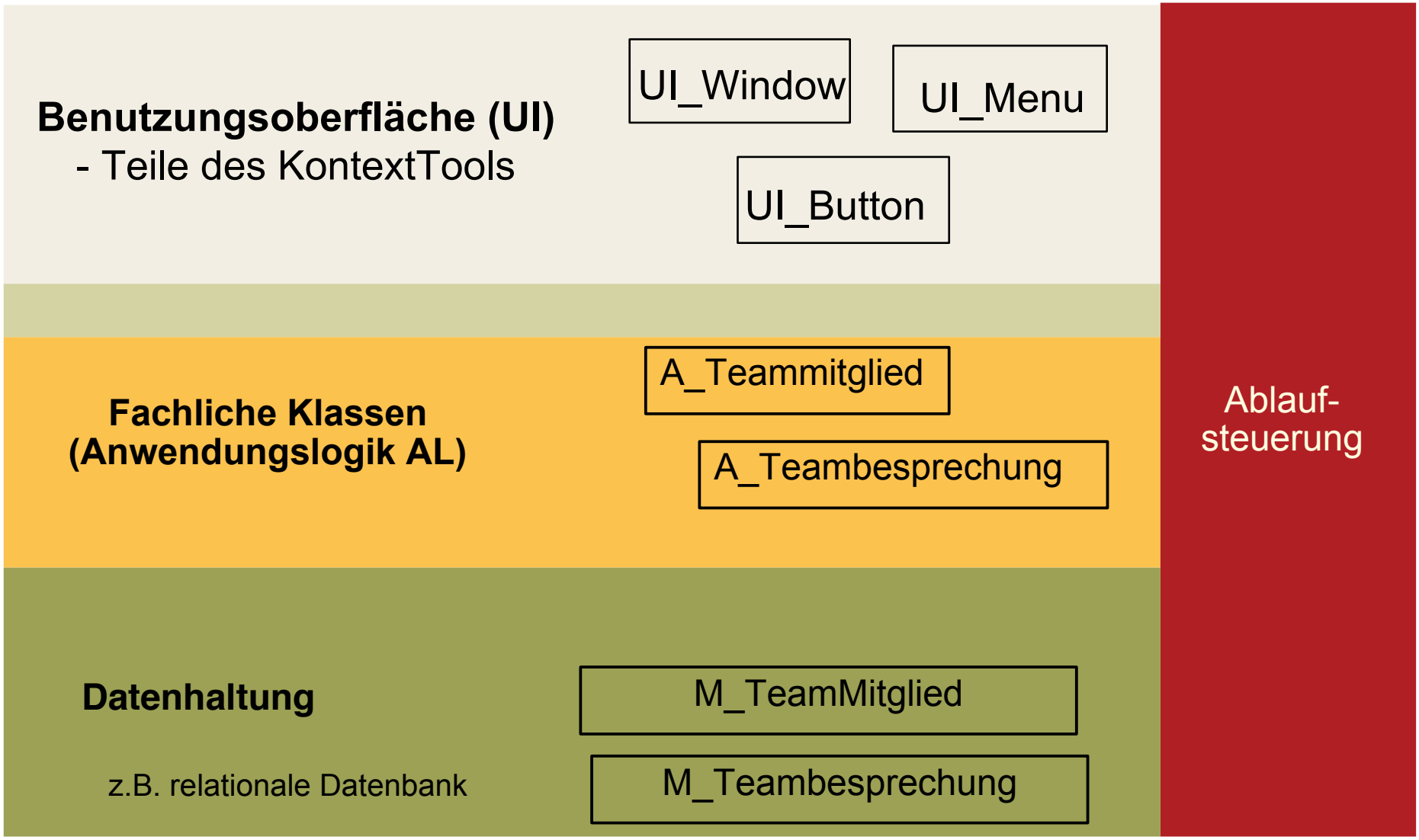
The End

- Diese Folien sind eine stark überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. used by permission. Verbreitung, Kopieren nur mit Zustimmung der Autoren.
- Wieso muss ein Konnektor zwischen GUI und Anwendungslogik vermitteln?
- Wie implementiert man einen Konnektor mit inneren Klassen?
- Wie werden die Rollen der Kollaboration des Konnektors realisiert?

43.A.1 Benutzungsoberflächen (UI) und Anwendungslogik

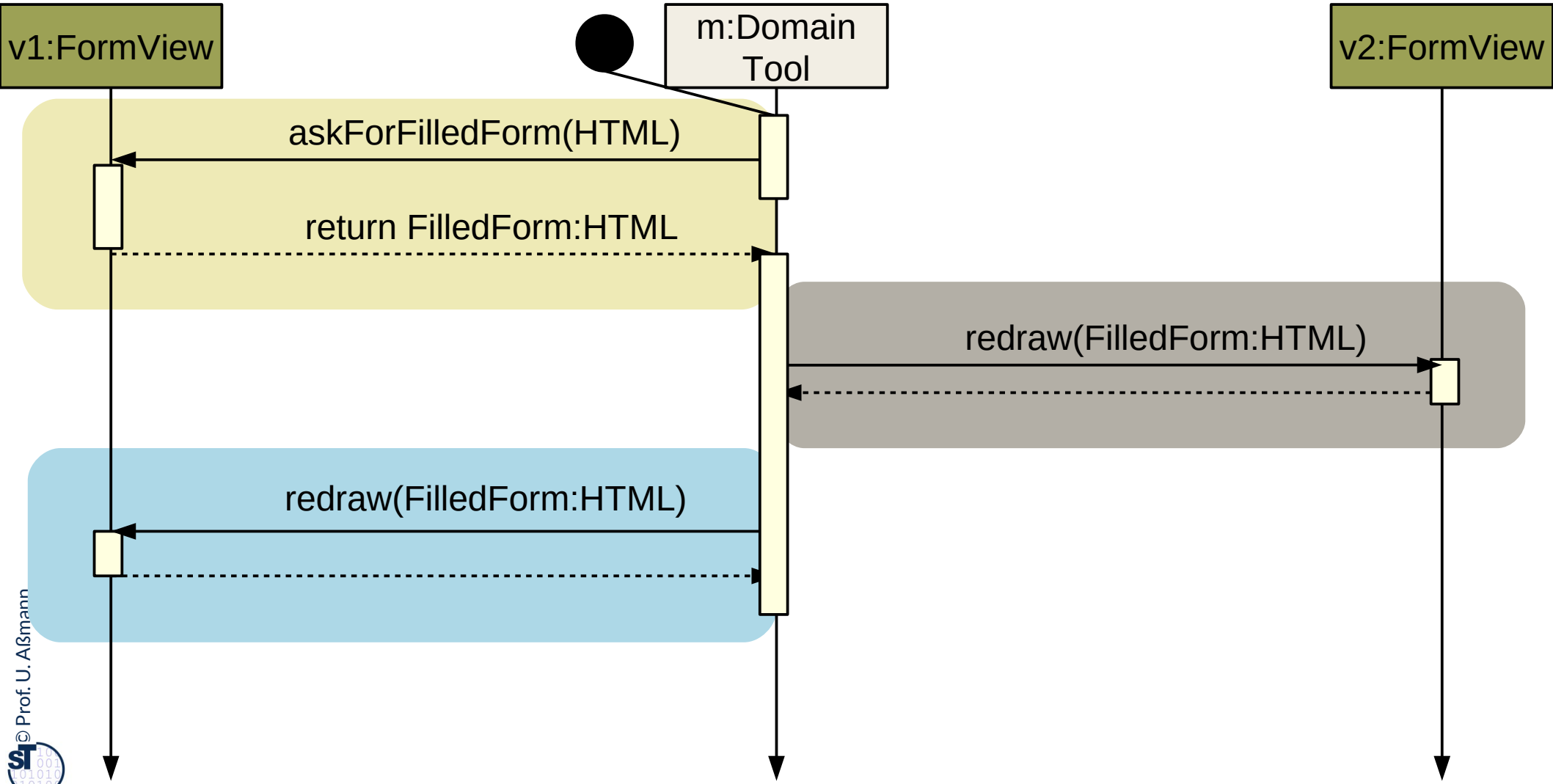
Verschiedene Arten der Kopplung zwischen Benutzer und Software

Schichtenarchitektur, grob



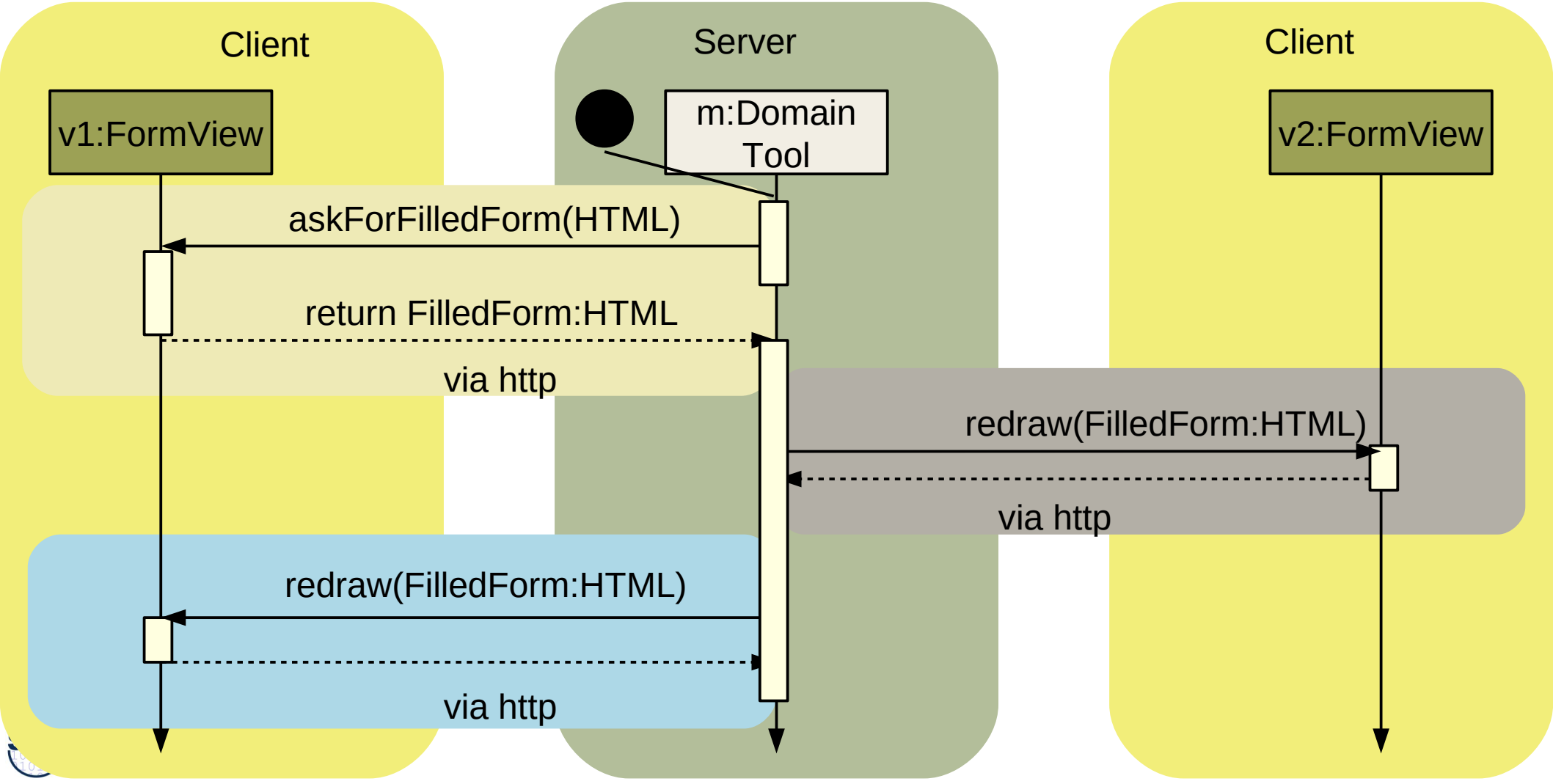
43.2.1. Formularbasierte UI mit XML

- HTML und XML bieten standardisierte Formate für Formulare an, die von Browsern dargestellt, interpretiert, und ausgefüllt werden können
- Die Play-In und Play-Out-Konnektoren transportieren XML-Dokumente



Formularbasierte UI mit XML übers Web

- HTML und XML können vom Client zum Server übertragen werden
- Kanalprotokoll `http` oder `https`



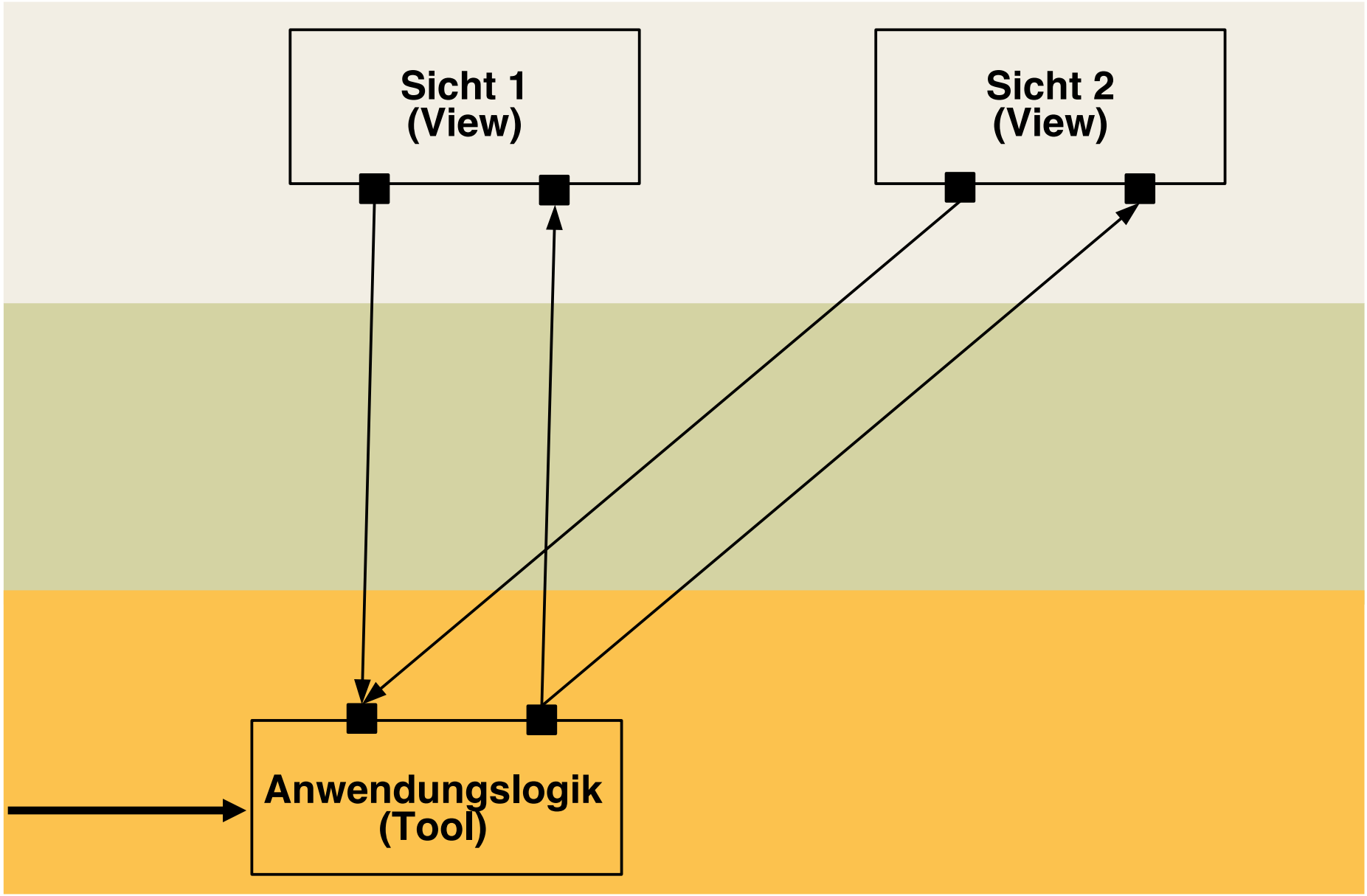
43.A.2 Phasen der reaktiven graphischen Benutzeroberflächen (GUI)

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 Tool asynchron zusammen
  - Der Controller beherrscht und kapselt die Interaktion, die Initiative geht von ihm aus
  - View und Tool sind gegenüber ihm passiv

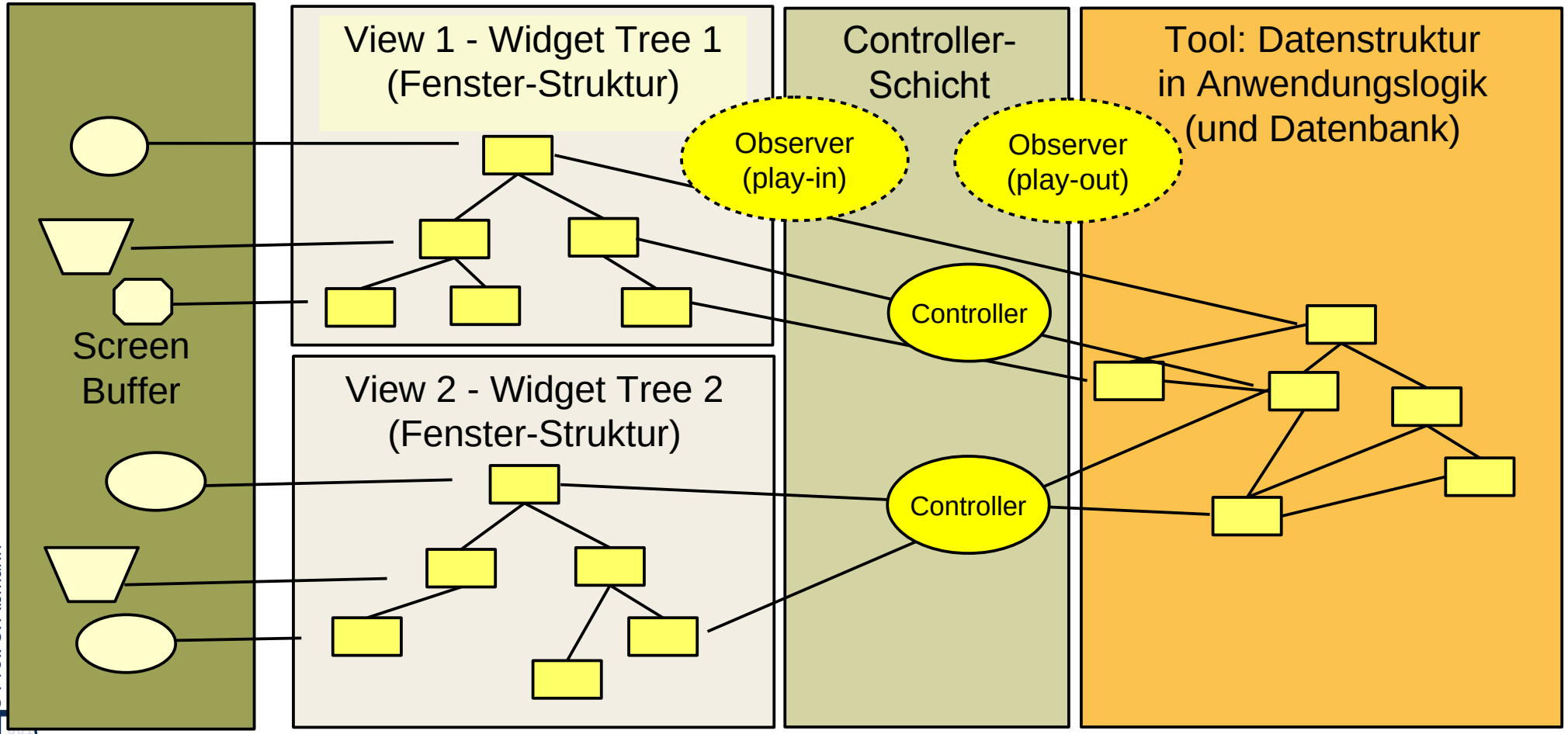
Controller sind Konnektoren zwischen Tool und View (gefaltet)

- Konnektoren, die Ports verbinden, abstrahieren die Kollaborationen



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



Programme mit asynchronen GUI laufen in 3 Phasen

1) **Aufbau der Schichten:** Aufbau der Datenstrukturen

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

2) Netzaufbau

1) **Vernetzung der Fensteroberfläche mit der Anwendungslogik** über die *Controller-Konnektoren*, um Reaktionen der Anwendungslogik zu ermöglichen
   1) a) **Play-Out-Kollaboration:** Anschluß des GUI-Reaktionscodes auf Veränderungen der Toolstruktur (View wird pull-Observer des Controller, indirekt des Tools, Vorbereitung des Play-Out)
   2) b) **Play-In-Kollaboration:** Anschluß des Tool-Reaktionscode auf Benutzereingaben (Controller ist push-Observer der Widgets, Vorbereitung des Play-In)

3) Reaktionsphase (Reaktive, asynchrone Phase)

s. nächste Folie

Zusammenspiel der Widget-Struktur und der Anwendungslogik

3) Reaktionsphase (Reaktive, asynchrone Phase)

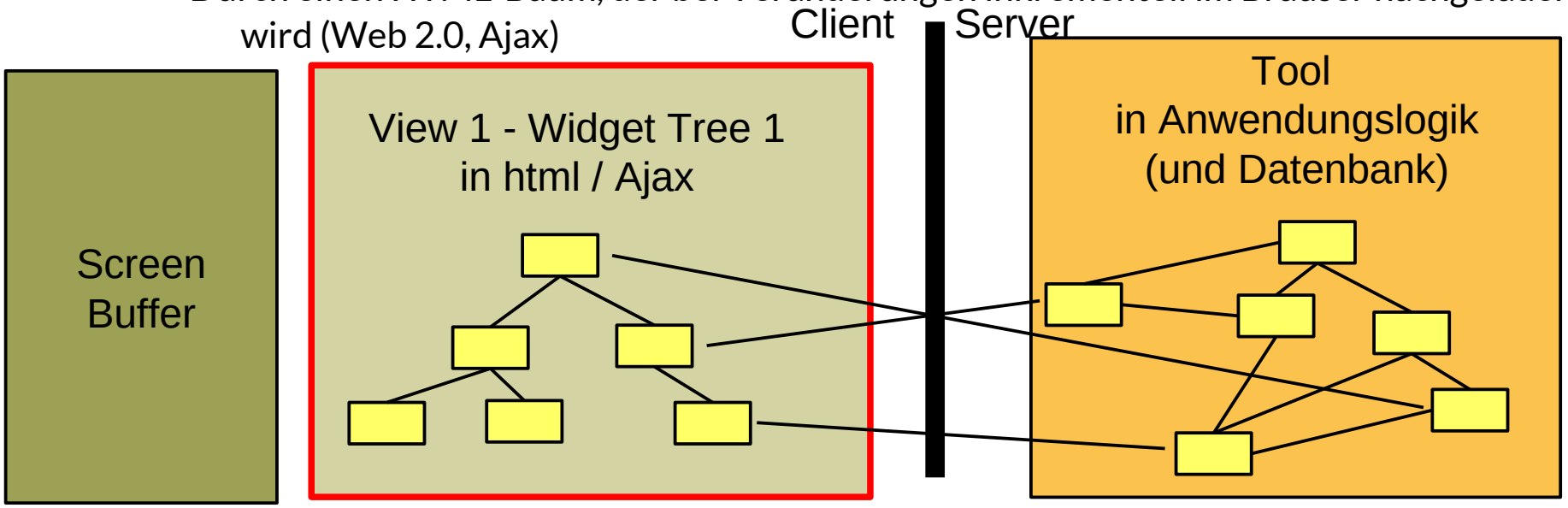
- **Play-In:** bei der die Benutzeraktionen vom System (Ereignisverwaltung) als Ereignisobjekte ins Programm gegeben werden
  - **Event notification:** Ereignismeldung, dass Benutzer etwas getan hat
  - **Data transmission:** etwaiger Transfer der Daten
- **Play-Out:** Bei der in der Anwendungslogik 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 Tools**
  - Die Controllerschicht hat die Kontrolle über das Verhalten
  - *reagiert* auf die Ereignisse im View und im AnwendungsTool (reaktives System)
  - *steuert* Redraw und Aktionen auf Tool

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

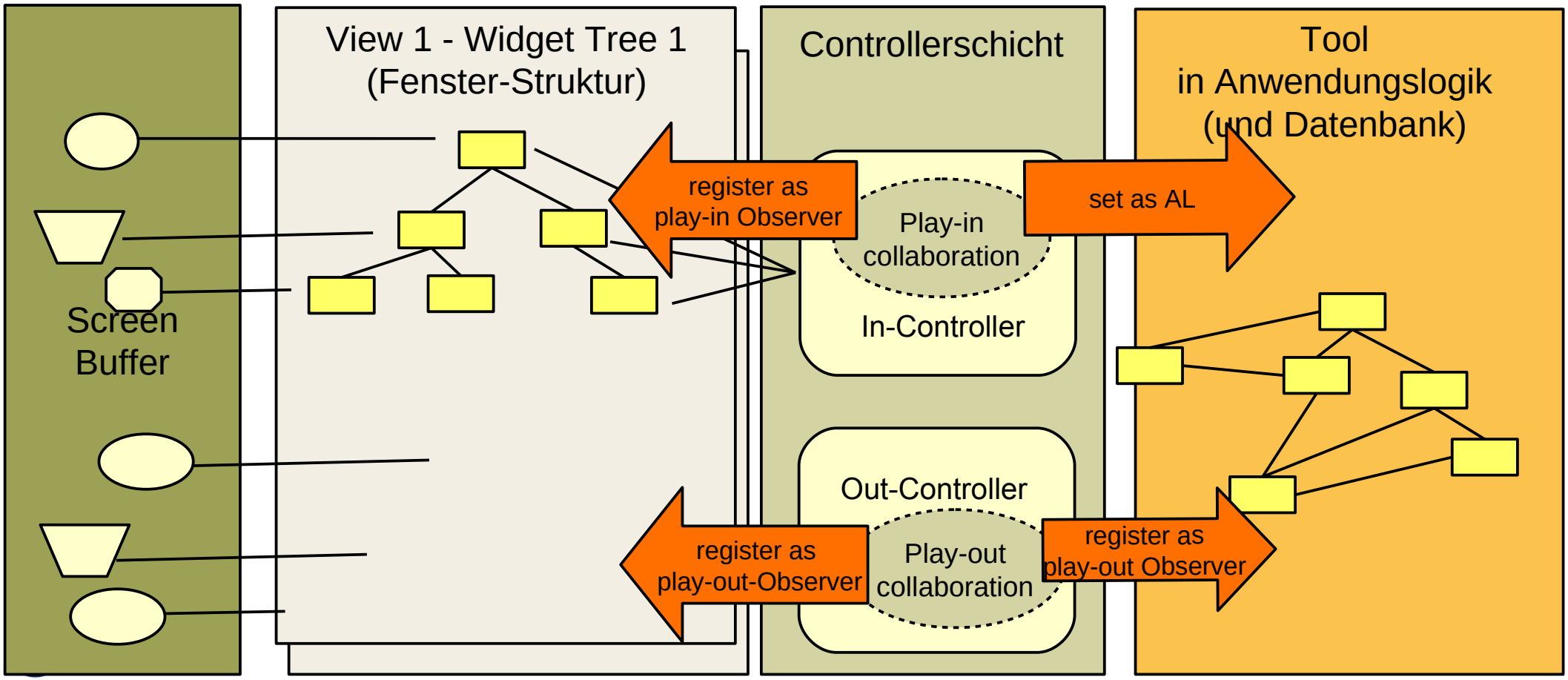
Verschiedene Techniken für den Aufbau der Datenstrukturen:

- **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)



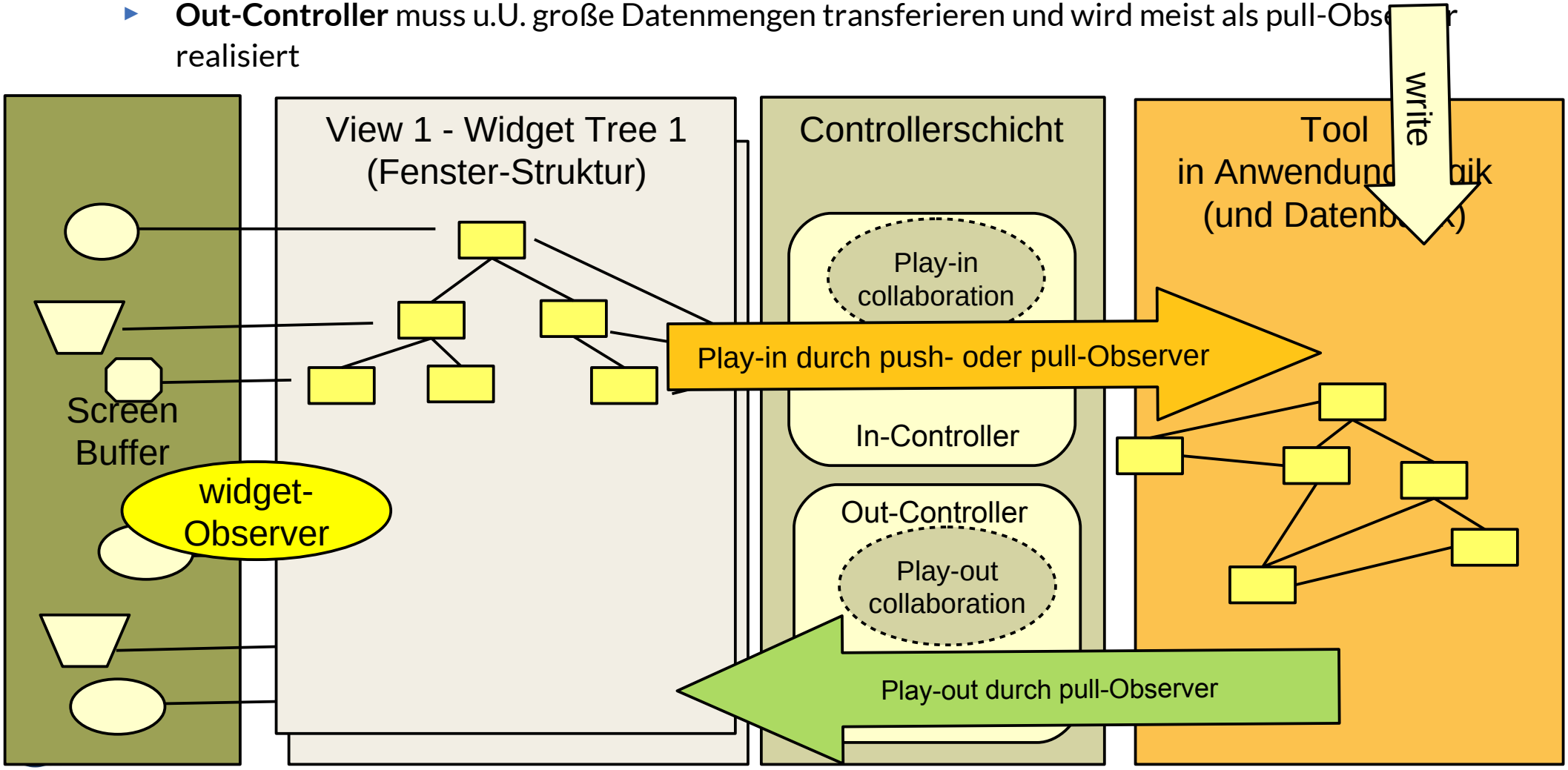
Phase 2) Netzaufbauphase: Aufbau der Verbindung

- 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) “Life” Überblick MVC Dynamik

- **Tool** ist passiv. Der Controller interpretiert die Eingaben und schreibt das Tool entsprechend
- **View** ist weitg. passiv. Controller benachrichtigt View, wenn sich was im Tool 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-Obse r realisiert



Die Dynamik der Phase 3 (“Life”)

Phase 3 (Dynamik) trennt zwischen Ereignisverarbeitung und Datentransport

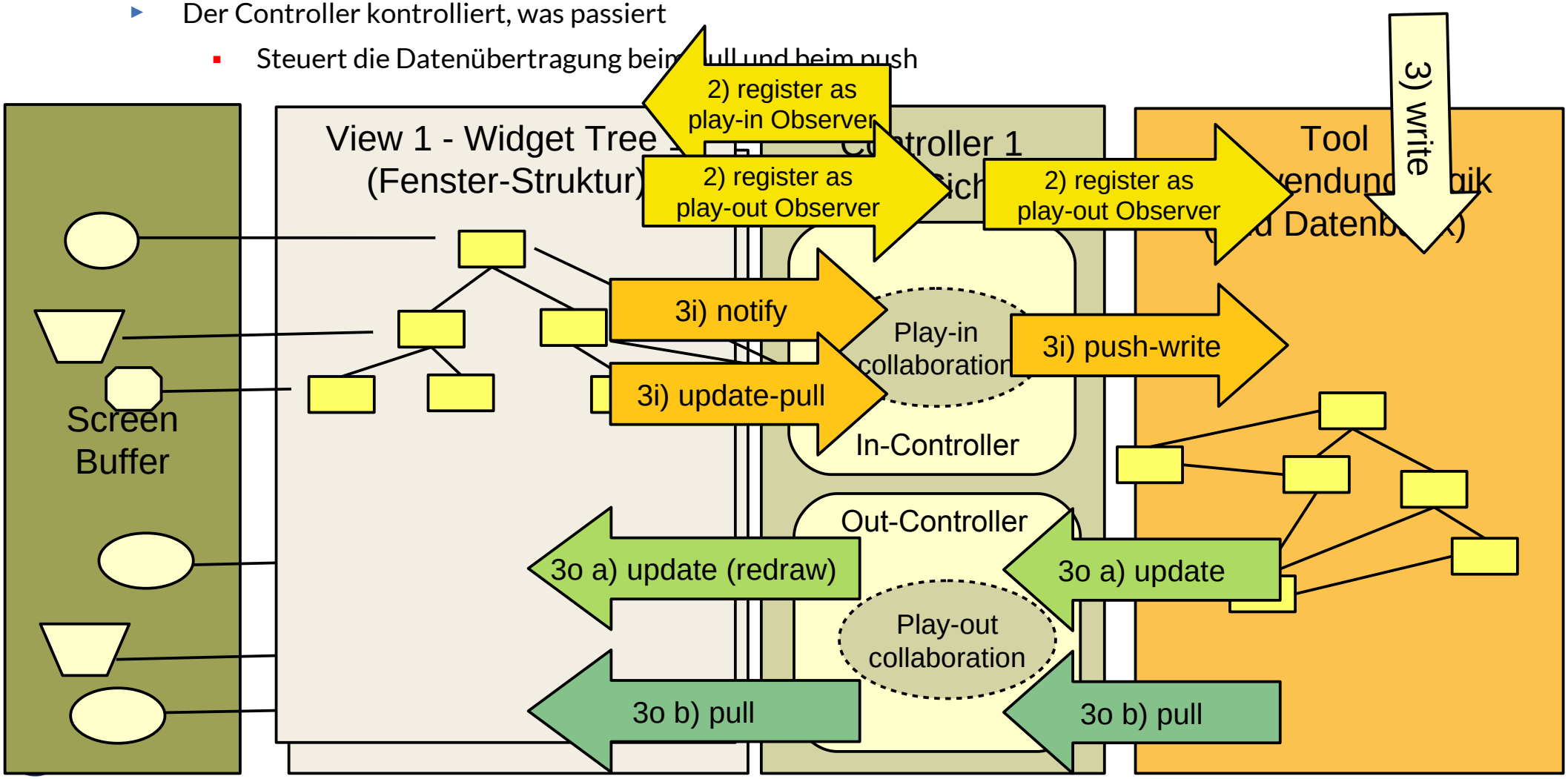
Die Konnektoren setzen push- oder pull-Observer-Muster ein

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

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

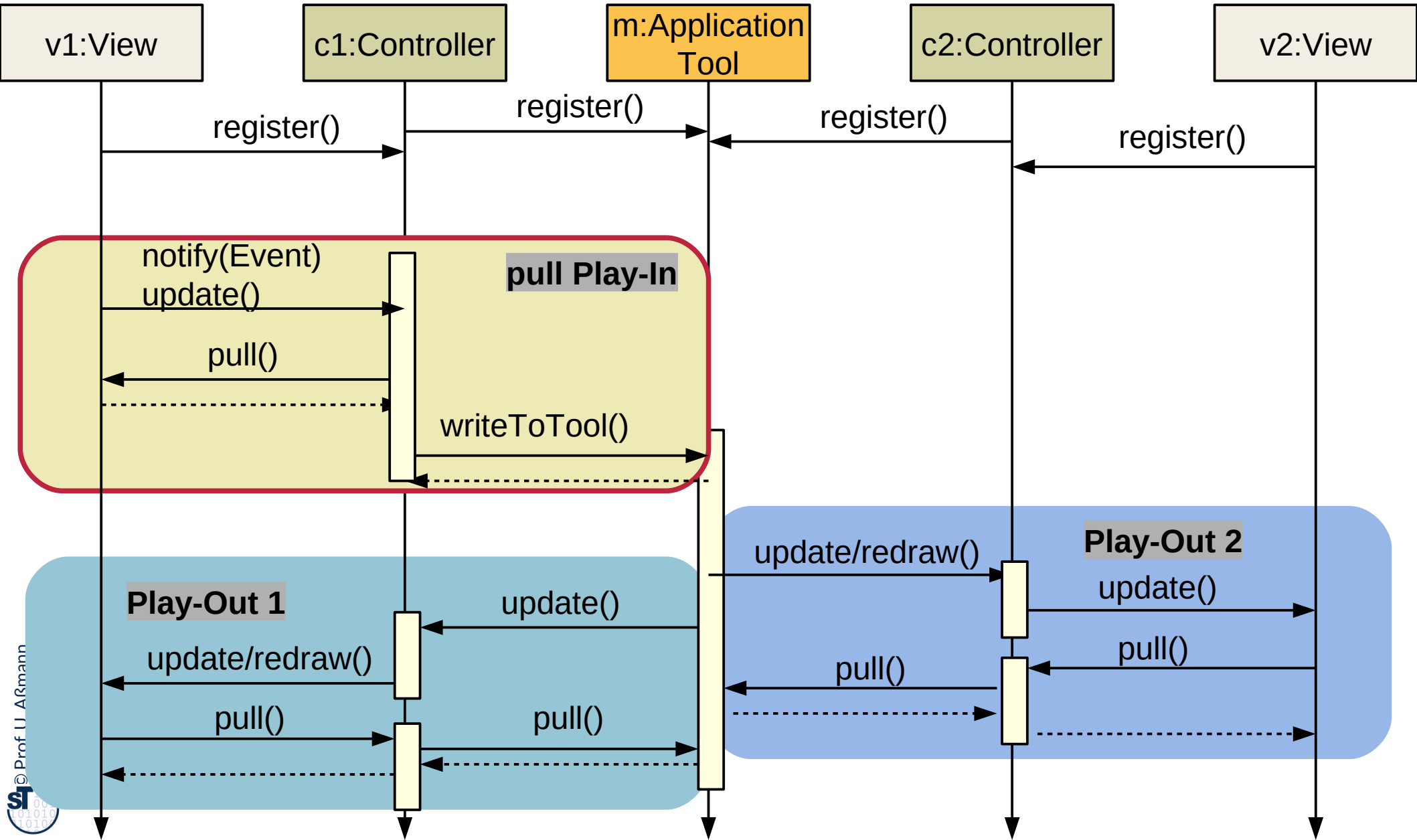
Gesamte MVC-Dynamik (indirektes Play-In und Play-Out)

- Tool 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 Tool 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]



43. Architektur interaktiver Systeme
- der Controller als Konnektor zwischen GUI und Anwendungslogik

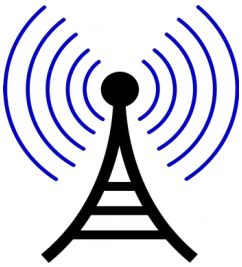
Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 19-0.1, 08.07.19

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
6) Swing

DRESDEN concept Exzellenz aus Wissenschaft und Kultur

Teil IV - Objektorientierter Entwurf (Object-Oriented Design, OOD)

1) 40: Überblick
2) 41: Einführung in die objektorientierte Softwarearchitektur
   1) Architekturprinzipien, Architekturstile, Perspektivenmodelle
   2) Modularität und Geheimnisprinzip
   3) BCD-Architekturstil (3-tier architectures)
3) 42: Verfeinerung mit querschneidender Objektanreicherung
4) **43: Architektur interaktiver Systeme**
5) 44: Punktweise Verfeinerung von Lebenszyklen
   - Verfeinerung von verschiedenen Steuerungsmaschinen



Literatur

- Obligatorisch:
  - [PassiveView] Martin Fowler. Passive View. http://www.martinfowler.com/eaaDev/PassiveScreen.html. Strikte Schichtung, aktiver Controller 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

Ziele

- Diese Vorlesung ist wichtig für das Winter-Praktikum, denn viele Gruppen müssen einen Swing-basierten Controller bauen
  - Swing ist ein komplexes Framework, das Einarbeitungszeit benötigt
  - GUI-Architekturen sind das Komplexeste, was wir bisher in dem Kurs besprochen haben – sie benötigen alle Kapitelinhalte

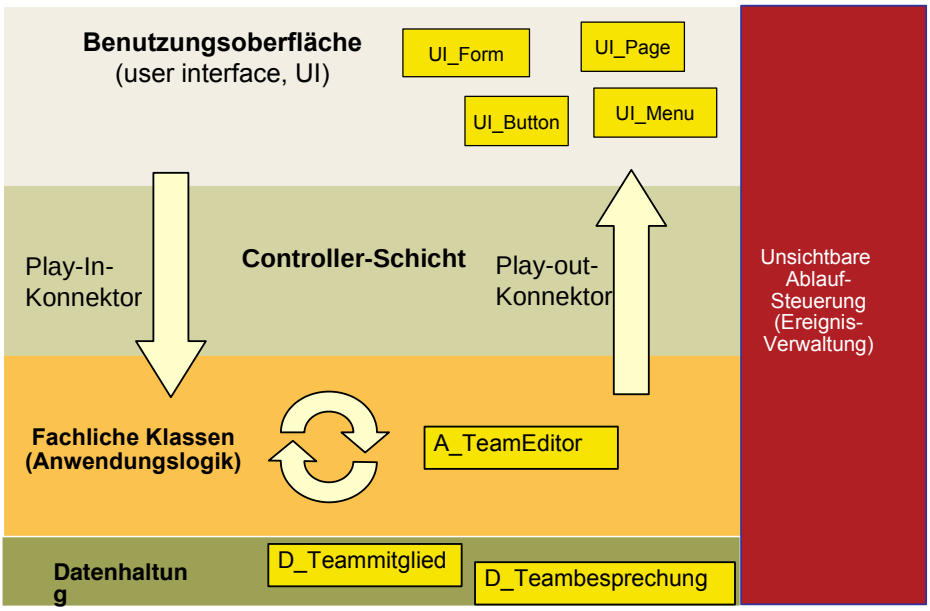
- Die Architektur interaktiver Anwendungen ist eines der komplexesten Gebiete der Software-Architektur
- Um sie zu verstehen, brauchen wir *alle Teile* des Kurses:
  - Kollaborationen und Konnektoren
  - Schichten
  - Steuerungs- und Protokollmaschinen
  - Sequenzdiagramme
  - Entwurfsmuster
- Resourcen: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

> Die Bildung kommt nicht vom Lesen, sondern vom Nachdenken über das Gelesene.
>
> Carl Hilty, 28.02.1833 - 12.10.1909
>
> Schweizer Richter und Staatsrechtler, Buchautor und christl. Staatsrechts-Philosoph
>
> Seine Bücher beeinflussten auch K. Adenauer

43.1 Benutzungsoberflächen (UI) und Anwendungslogik

Verschiedene Arten der Kopplung zwischen Benutzer und Software

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
  - Verteilte UI (Web UI)

    ==> dann muss der Controller die verteilte und parallele Verarbeitung steuern

- **Asynchrone UI:** die Anwendungslogik reagiert auf die UI
- Treiber ist die UI
  - Graphische UI (GUI)
  - Tangible UI (TUI)

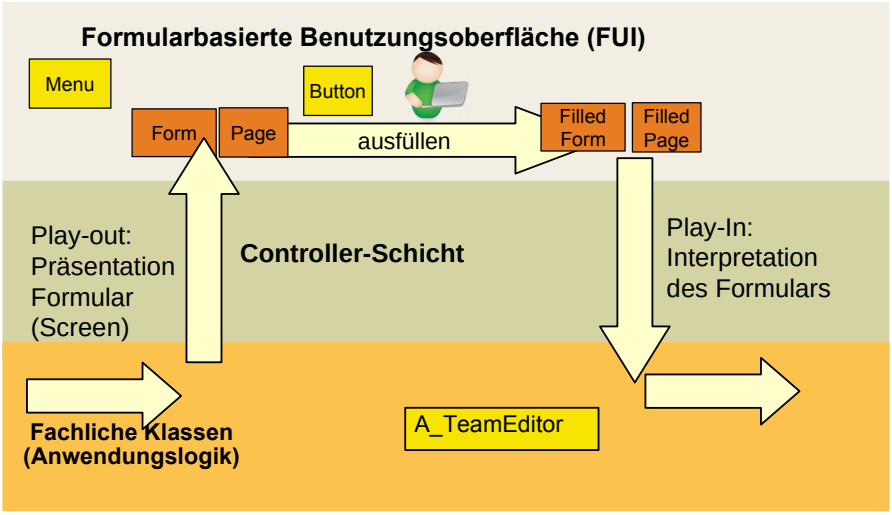
    ==> dann muss der Controller die parallele Verarbeitung steuern

43.2 Kopplung von synchronen Benutzeroberflächen mit der Anwendungslogik

- Text- und Formularbasierte Oberflächen (Form-Based UI, FUI) sind meist synchron mit der Anwendungslogik gekoppelt
- Konnektor sehr einfach: 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

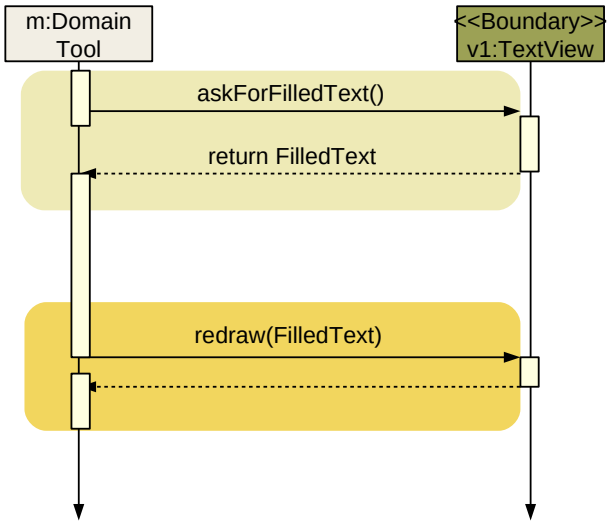
- Die Anwendungslogik ruft das formularbasierte UI mit einem leeren Formular auf und warten auf das Ausfüllen des Benutzers (synchron)
- Die Play-In und Play-Out-Konnektoren sind besonders einfach



In diesem Fall ist die Controller-Schicht ganz einfach, sie besteht aus synchronen Aufrufen an die FUI-Schicht.

43.2.1. Textbasierte UI mit synchronem Update (*ein* View)

- In Java: Eingabe mit System.in, Ausgabe mit System.out
- Play-In und Play-Out-Konnektoren sind Prozeduraufrufe an die UI



Einfache textuelle UI füllen Kommandozeilen oder Formulare aus.

In diesem Fall ruft man von der Anwendungslogik aus Ein- und Ausgaberoutinen für Texte auf.

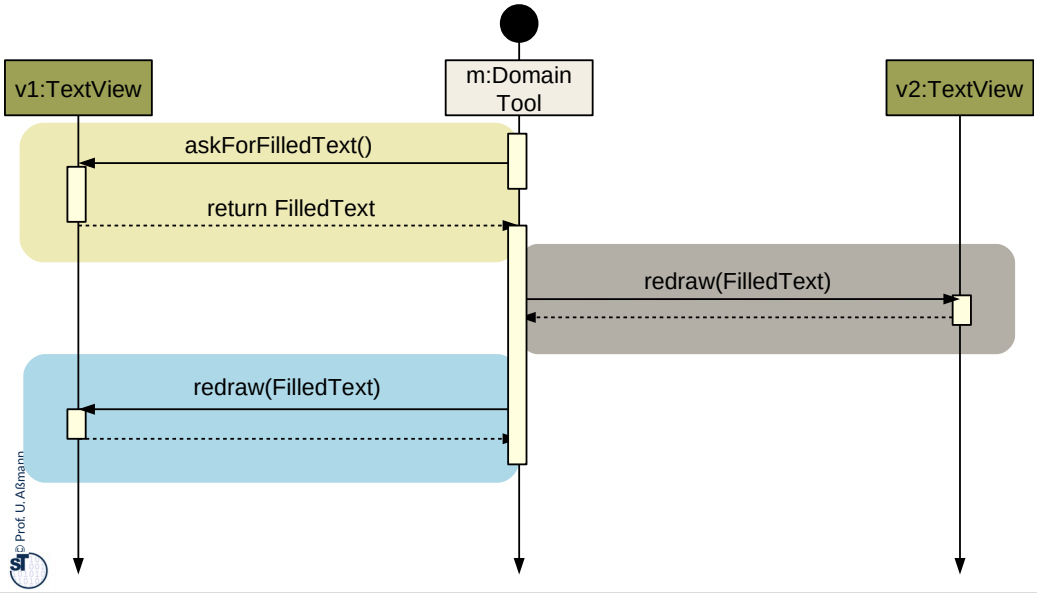
Einfache textuelle Sichten

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

```
class PersonTool {

  … activities of the tool …

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

  Person p.number = num;

  foreach (view ; tool.getViews()) {
    view.redraw(p);
  }

  … further activities of the tool …
}
```

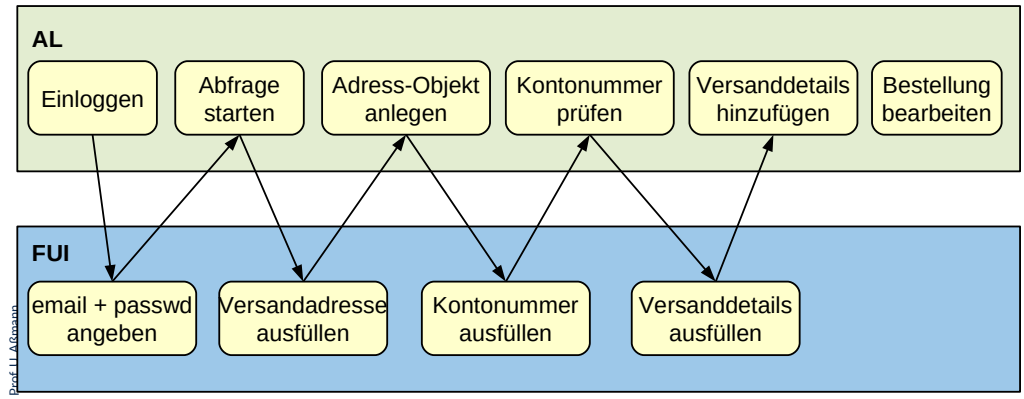
Textbasierte UI mit synchronem Update (mehrere Text-Views)

- Immer alles schön nacheinander (synchron)



Screen-Flow

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



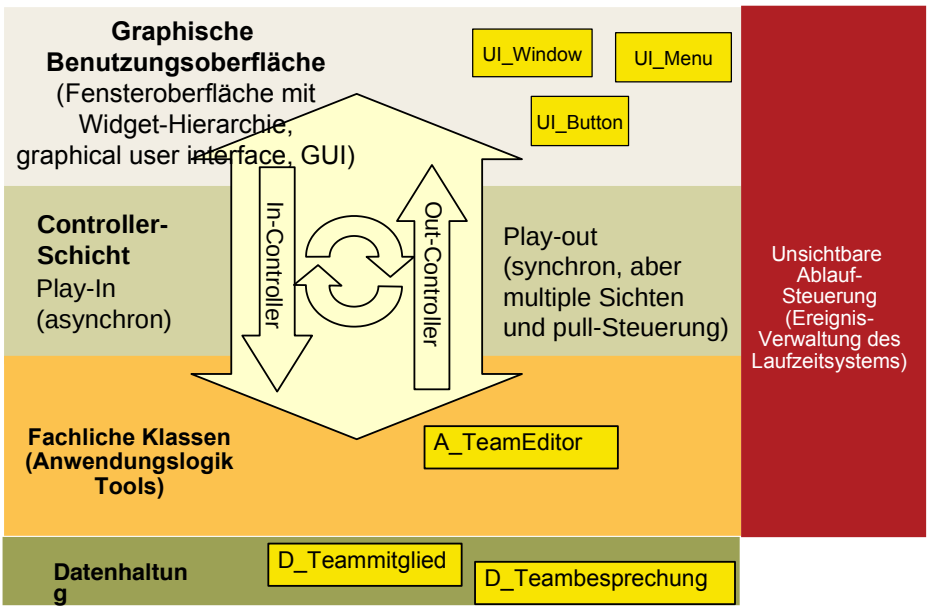
Für den Bau einer formularbasierten Anwendung sollte man zuerst den ScreenFlow festlegen, das ist die Interaktion zwischen der FUI, den Formularen und den Aktionen der Anwendungslogik auf dem Server

43.3 Überblick zu reaktiven graphischen Benutzeroberflächen (GUI)

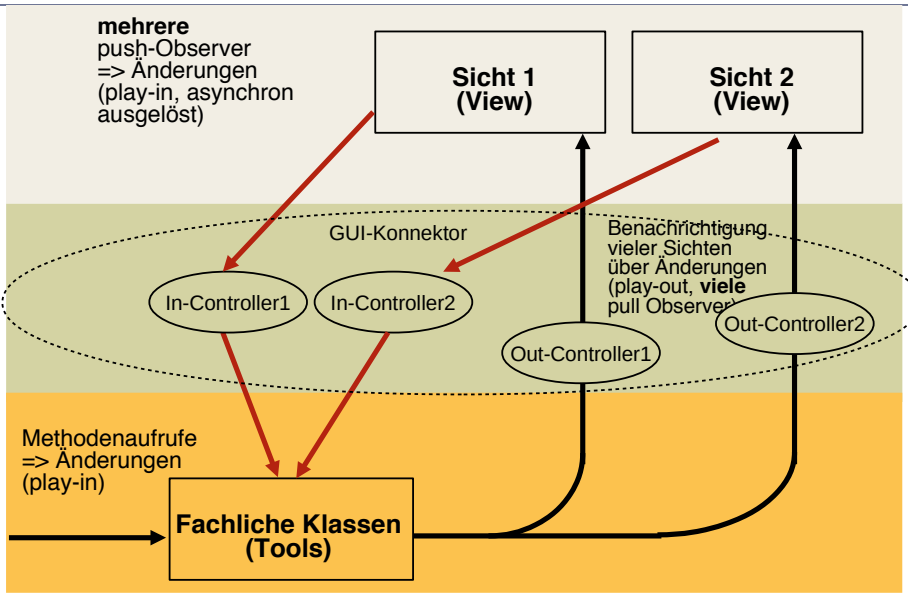
Kopplung der GUI und Anwendungslogik durch Controller

- Bislang war es einfach, aber auch unflexibel
- Im Normalfall ist es schwieriger, denn da bringt ein *Controller* bzw. eine *Controllerschicht* die Ereignisse, "auslösenden" Fensterelemente (Sicht) und Tool asynchron zusammen
  - Der Controller beherrscht und kapselt die Interaktion, die Initiative geht von ihm aus
  - View und Tool sind gegenüber ihm passiv

Schichtenarchitektur der reaktiven Benutzungsoberfläche (GUI)

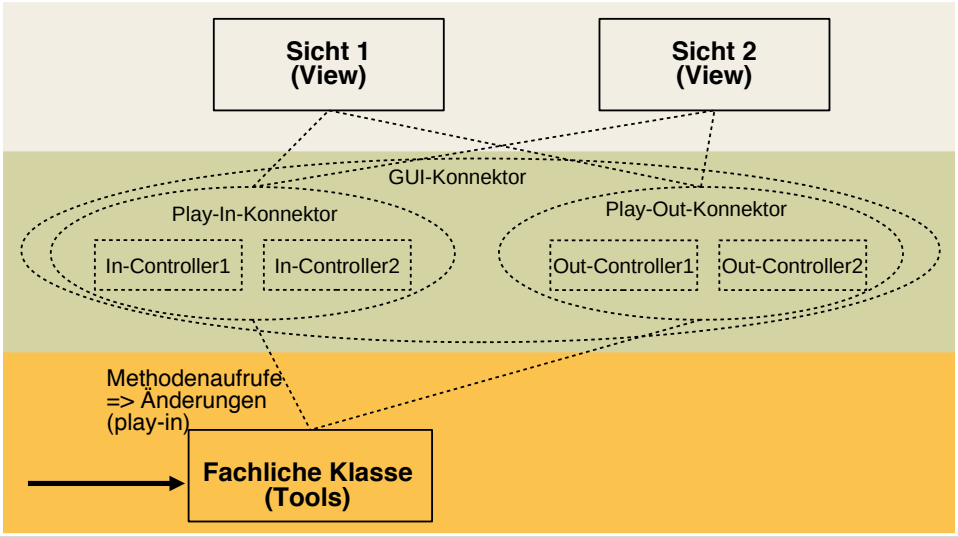


Tool, Controller und Views in strikter Schichtung



Controller sind Konnektoren zwischen Tool und View

- Meist existiert ein Hauptobjekt in der Kollaboration, womit der Controller einen Konnektor darstellt



43.4 Controller als Steuerungsmaschinen in Konnektoren

- Im Entwurfsmuster “PassiveView” bestehen die Controller aus Steuerungsmaschinen, die die Ereignisse der GUI in die Ereignisse der Anwendungslogik übersetzen und umgekehrt
- Die Controllerschicht ist aktiv; View und Tool bleiben passiv

Implementierung der Controller als Steuerungsmaschinen

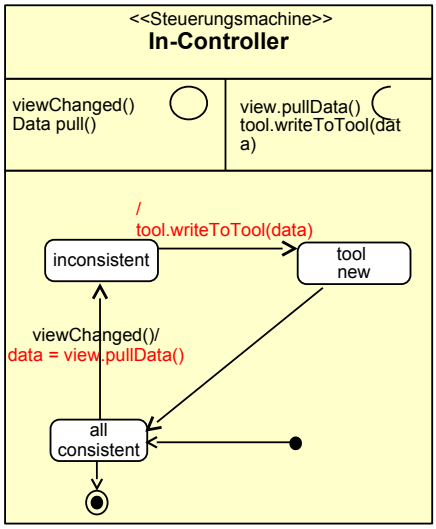
Ein Controller-Konnektor wird durch eine Steuerungsmaschine implementiert,
die die Ereignisse auf der Fensterhierarchie (UI)
in die Aufrufe an die Tools der Anwendungslogik *übersetzt* (u.u.)

- 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.

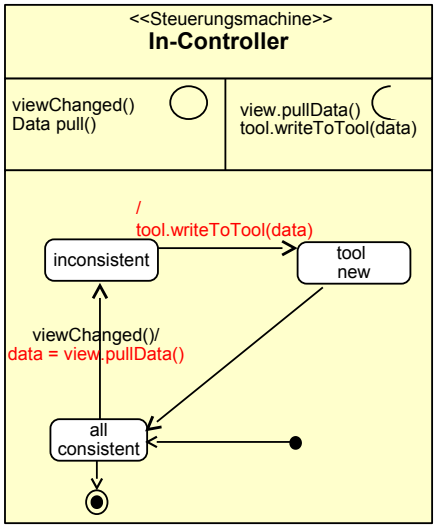
Input-Controller mit drei Zuständen

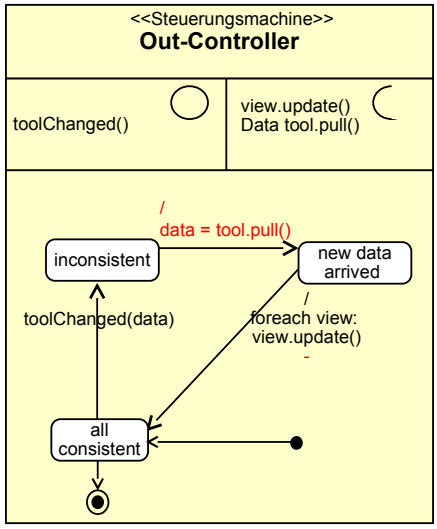
- Die entstehende Steuerungsmaschine steuert View und Tool an (“beherrscht” sie)
- Getriggert wird sie durch die Ereignisse viewChanged (update). Sie löst pullData und writeToTool aus



Variante 1: Paare von Controller-Steuerungsmaschinen

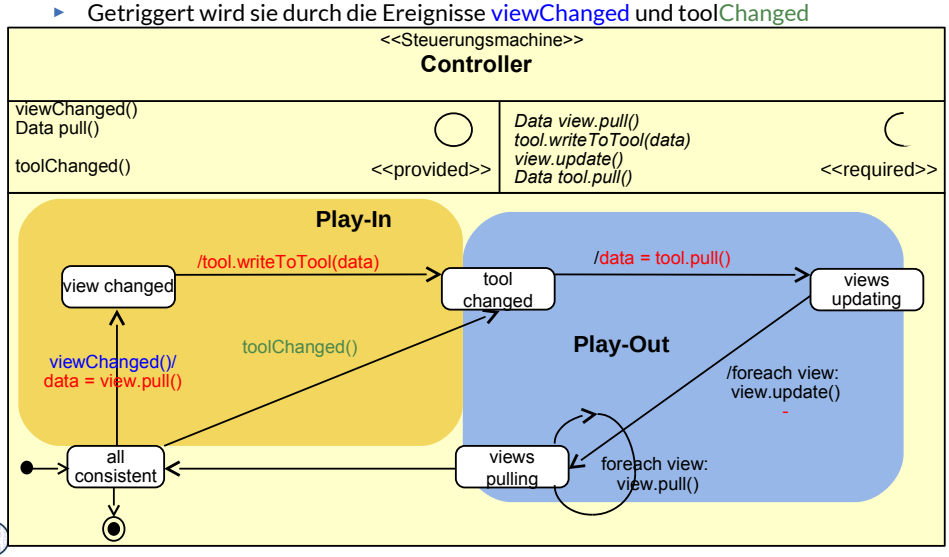
- In- und Out-Steuerungsmaschinen bilden Elemente einer oder mehrerer In- und Out-Kollaborationen





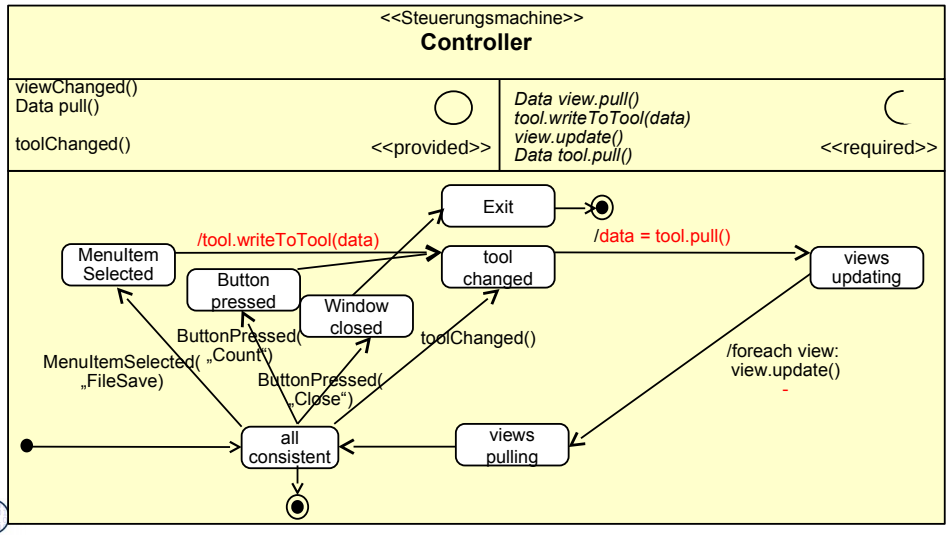
Variante 2: Controller als bidirektionale Inout-Konnektoren

- In- und Out-Controller können auch *zusammengelegt* sein (z.B. in Spring)
- Die entstehende Steuerungsmaschine steuert View und Tool an (“beherrscht” sie)
- Getriggert wird sie durch die Ereignisse viewChanged und toolChanged



Variante 2b: Prinzipieller Aufbau von InOut-Controllern

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



Implementierung der Controller

- Die Controllerschicht wird realisiert entweder als
  - Konnektor mit einer Steuerungsmaschine
  - Menge von In-/Out-Konnektoren mit kommunizierenden Steuerungsmaschinen
  - Oder einer Menge von bidirektionalen Konnektoren
- Controller können kombiniert (InOut-Controller), 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

43.5 Implementierung mit Konnektoren

Ein einfacher MVC-Konnektor (als Team mit inneren Klassen)


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

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


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

MVCModular.java

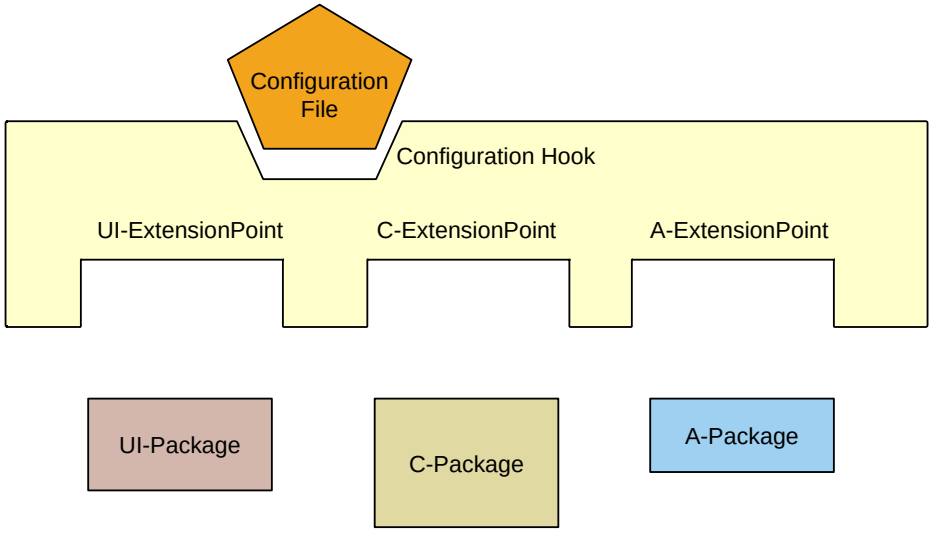
43.6. MVC Frameworks

- (Controller Frameworks)

MVC-Frameworks mit vorgefertigten GUI-AL-Konnektoren

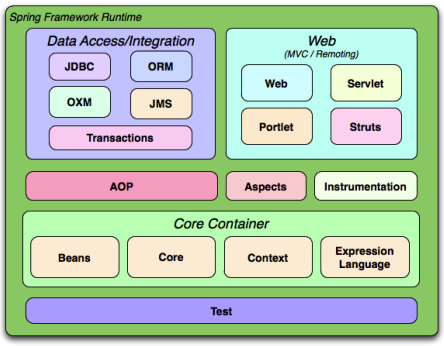
- Die Struktur einer Controllerschicht kann sich von Anwendungsklasse zu Anwendungsklasse sehr unterscheiden.
- Ein **MVC-Framework** definiert einen GUI-AL-Konnektor und gibt eine Struktur der Controllerschicht vor, definiert Protokolle für die Ereignismeldung und den Datenaustausch vor und kann durch den Entwickler erweitert werden.
  - MVC Frameworks benötigen Konfiguration und "Plugins"
  - **Oft folgt man dem Prinzip "Convention over configuration":** Konventionen über Dateiverzeichnisse und Konfigurationsdateien vereinfachen dem MVC-Framework das Auffinden von Controller-, View-, Anwendungsklassen, sowie Hinweise zu ihrer Verdrahtung
  - Konfigurationsdateien meist in XML oder Java-Property-Lists
- Berühmte Beispiele:
  - Java: Spring, Struts
  - Ruby: Ruby on Rails
  - Groovy: Grails

MVC Frameworks kennen “Plugins”



Spring Framework

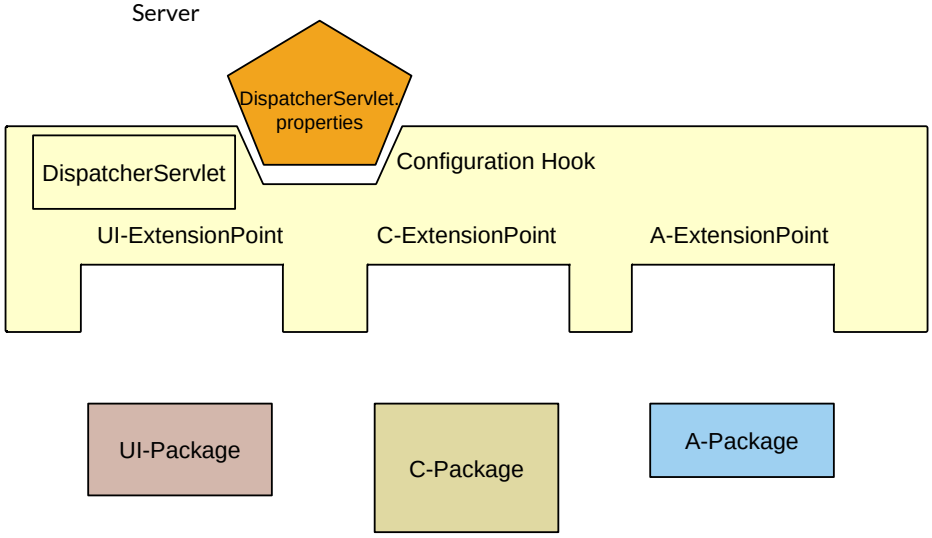
- *Spring* ist das im Praktikum im WS verwendete MVC-Framework
  - Webbasiert, d.h. Controllerschicht ist auf Client und Server verteilt implementiert
  - Konfigurierbar durch XML-Dateien und Java Property Files
  - Erweiterbar
- Das Salespoint-Framework nutzt als Konnektor zum GUI das Konnektor-Framework SPRING
  - Main controller, subcontroller
  - Web-MVC Frameworks brauchen *starke Schichtung*
  - Bietet sehr viele verschiedene Pakete, nicht nur für Web-UIs



- http://spring.io/guides

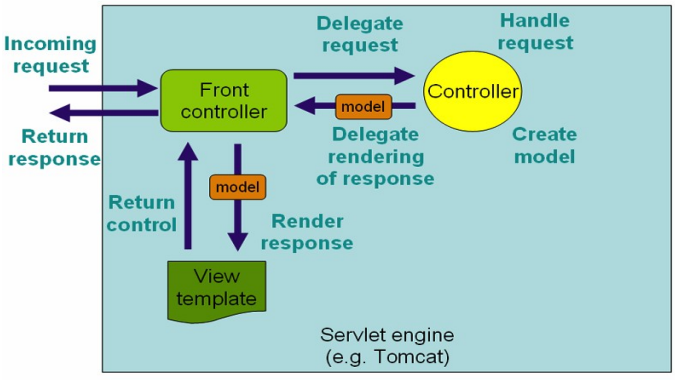
Spring Konfiguration

- Spring übernimmt das Management der Verteilung
  - Das Zusammenspiel zwischen Browser, Server und Anwendungslogik auf dem Server



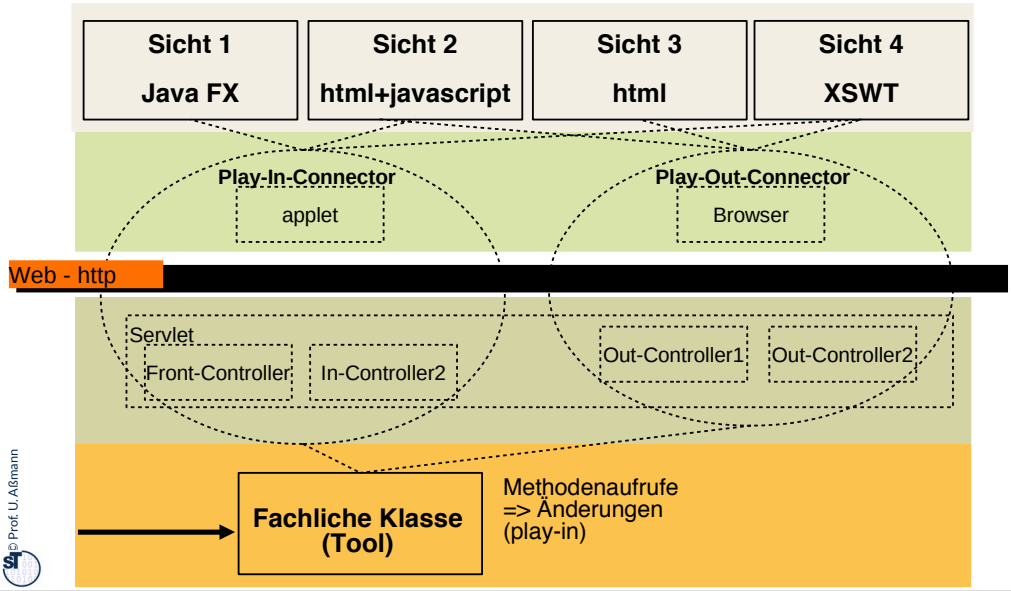
Struktur des String Controllers in Web- Systemen (Server Side)

- Der Spring Controller ist ein komposites Programm auf dem Server:
  - das Spring-DispatcherServlet enthält einen “FrontController”, der das ankommende Ereignis interpretiert (Steuerungsmaschine) und an untergeordnete Controller bzw. Steuerungsmaschinen weiter leitet



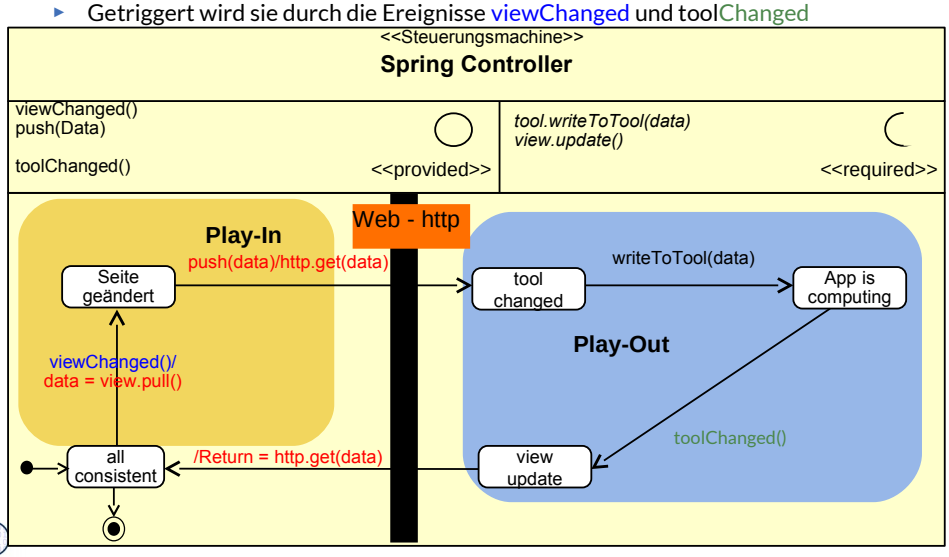
Controller sind Konnektoren zwischen Tool und View

- Im Folgenden gibt es ein Hauptobjekt, den Konnektor, der View, Controller und Tool verdrahtet



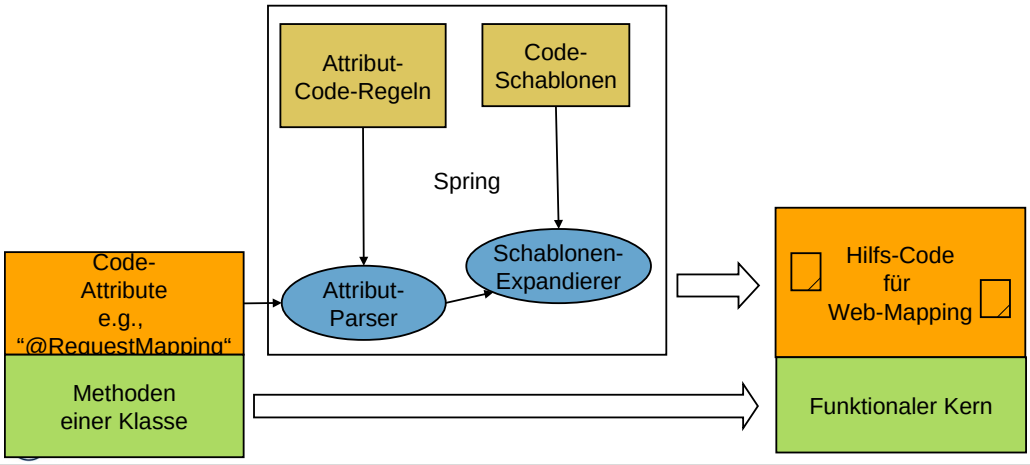
Server-seitiger Spring-Controller

- In- und Out-Controller können auch ***verteilt*** sein (z.B. in Spring)
- Die Steuerungsmaschine ist verteilt auf client und server
- Getriggert wird sie durch die Ereignisse viewChanged und toolChanged



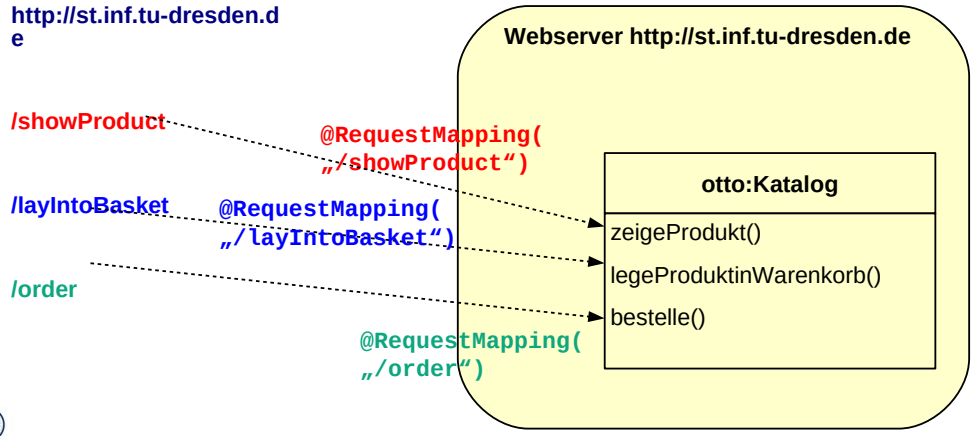
Spring nutzt @attribut-basierte Codegenerierung

- Spring wandet Java-@Attribute (sog. Metadaten) in Code um
- Attribute parameterisieren Schablonen (“templates”): Template-gesteuerte Codegenerierung
- Siehe auch Xdoclet, xdoclet.sf.net



@RequestMapping

- Ein **REST-Webservice** bildet URL (Web-Dateinamen) auf *aktive Methoden eines Webservice-Objekts* ab
  - `@RequestMapping("<relative URL>")`
- Wird die URL im Browser aufgerufen, wird die Methode aufgerufen und ihr Resultat als String im JSON-Format zurückgegeben



Was haben wir gelernt?

- GUI-Programme koppeln die GUI mit der Anwendungslogik mit Hilfe des Konnektor-Musters
  - Der Controller-Konnektor aktiviert die Views und die Anwendungs-Tools
- 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 Tool
  - Der Controller wird als Steuerungsmaschine implementiert und steuert alles (aktiver Konnektor)
- Das MVC-Framework Spring enthält eine stark geschichtete GUI-Anwendungskopplung
  - Enthält einen kompositen Controller (komposite Steuerungsmaschine)
  - Regelt den Verkehr zwischen Browser, Server, Servlet, Webservice-Methoden

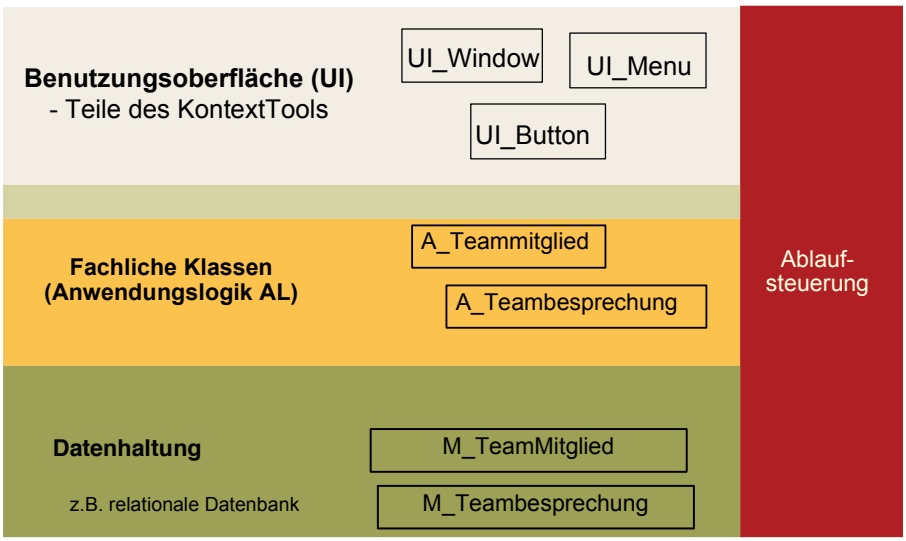
The End

- Diese Folien sind eine stark überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. used by permission. Verbreitung, Kopieren nur mit Zustimmung der Autoren.
- Wieso muss ein Konnektor zwischen GUI und Anwendungslogik vermitteln?
- Wie implementiert man einen Konnektor mit inneren Klassen?
- Wie werden die Rollen der Kollaboration des Konnektors realisiert?

43.A.1 Benutzungsoberflächen (UI) und Anwendungslogik

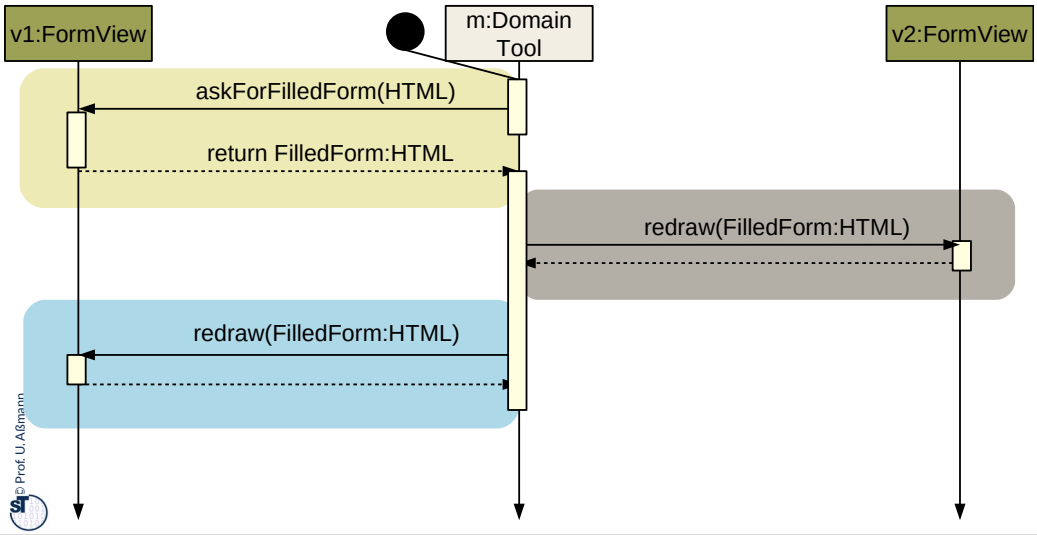
Verschiedene Arten der Kopplung zwischen Benutzer und Software

Schichtenarchitektur, grob



43.2.1. Formularbasierte UI mit XML

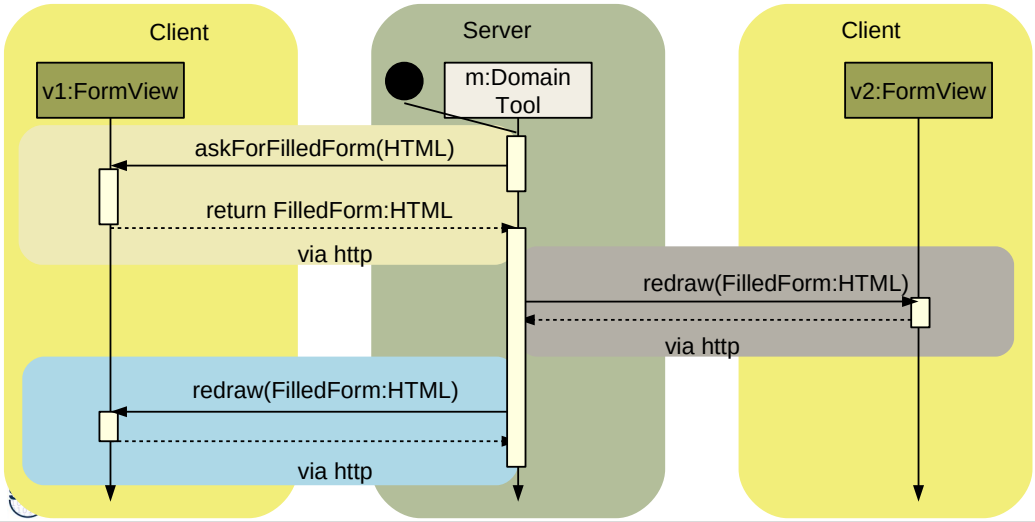
- HTML und XML bieten standardisierte Formate für Formulare an, die von Browsern dargestellt, interpretiert, und ausgefüllt werden können
- Die Play-In und Play-Out-Konnektoren transportieren XML-Dokumente



XML-basierte Formularanwendungen führen den synchronen Aufruf vom Server auf den Client übers Web aus (Webprotokolle wie http, webdav, REST).

Formularbasierte UI mit XML übers Web

- HTML und XML können vom Client zum Server übertragen werden
- Kanalprotokoll `http` oder `https`



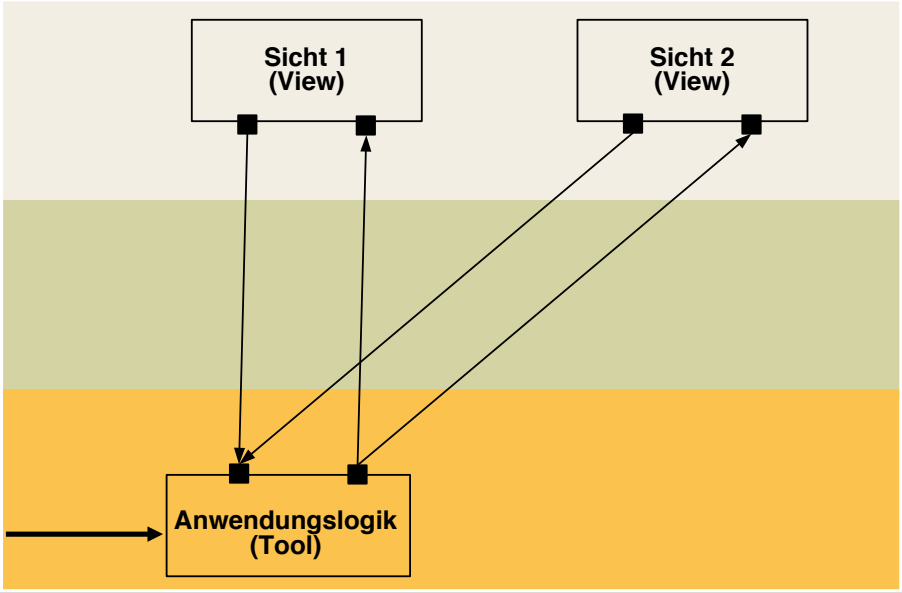
43.A.2 Phasen der reaktiven graphischen Benutzeroberflächen (GUI)

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 Tool asynchron zusammen
  - Der Controller beherrscht und kapselt die Interaktion, die Initiative geht von ihm aus
  - View und Tool sind gegenüber ihm passiv

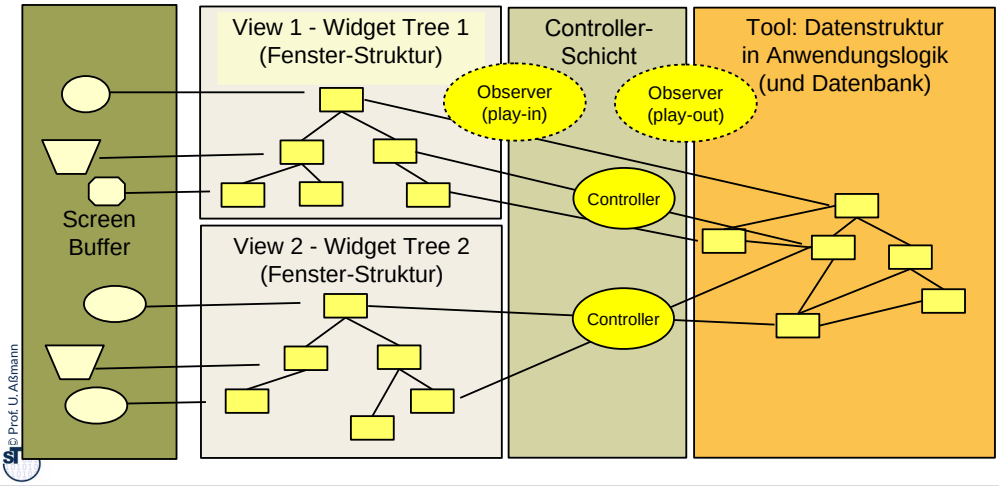
Controller sind Konnektoren zwischen Tool und View (gefaltet)

- Konnektoren, die Ports verbinden, abstrahieren die Kollaborationen



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



Programme mit asynchronen GUI laufen in 3 Phasen

1)Aufbau der Schichten: Aufbau der Datenstrukturen

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

2)Netzaufbau

1) **Vernetzung der Fensteroberfläche mit der Anwendungslogik** über die *Controller-Konnektoren*, um Reaktionen der Anwendungslogik zu ermöglichen
   1) a) **Play-Out-Kollaboration:** Anschluß des GUI-Reaktionscodes auf Veränderungen der Toolstruktur (View wird pull-Observer des Controller, indirekt des Tools, Vorbereitung des Play-Out)
   2) b) **Play-In-Kollaboration:** Anschluß des Tool-Reaktionscode auf Benutzereingaben (Controller ist push-Observer der Widgets, Vorbereitung des Play-In)

3)Reaktionsphase (Reaktive, asynchrone Phase)

s. nächste Folie

Zusammenspiel der Widget-Struktur und der Anwendungslogik

3) Reaktionsphase (Reaktive, asynchrone Phase)

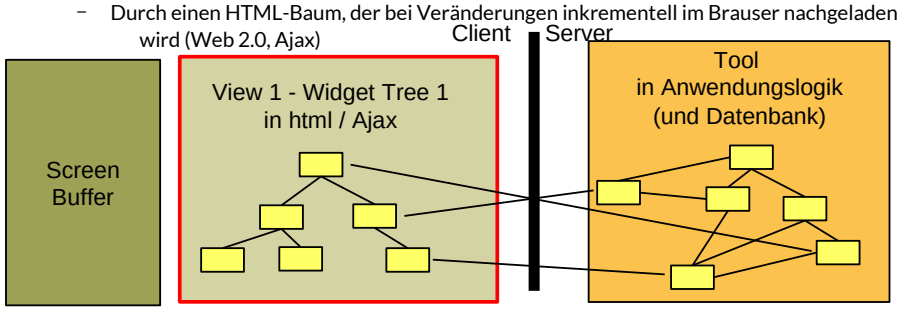
- **Play-In:** bei der die Benutzeraktionen vom System (Ereignisverwaltung) als Ereignisobjekte ins Programm gegeben werden
  - **Event notification:** Ereignismeldung, dass Benutzer etwas getan hat
  - **Data transmission:** etwaiger Transfer der Daten
- **Play-Out:** Bei der in der Anwendungslogik 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 Tools**
  - Die Controllerschicht hat die Kontrolle über das Verhalten
  - *reagiert* auf die Ereignisse im View und im AnwendungsTool (reaktives System)
  - *steuert* Redraw und Aktionen auf Tool

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

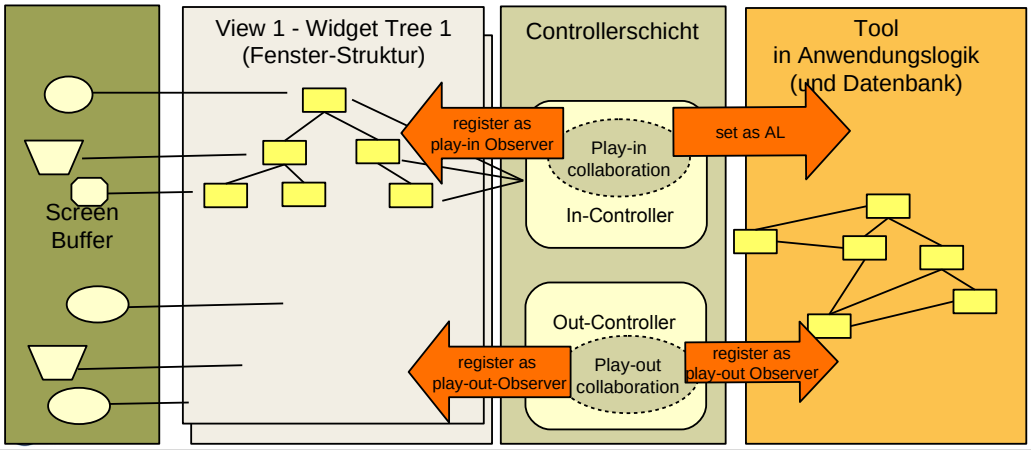
Verschiedene Techniken für den Aufbau der Datenstrukturen:

- **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)



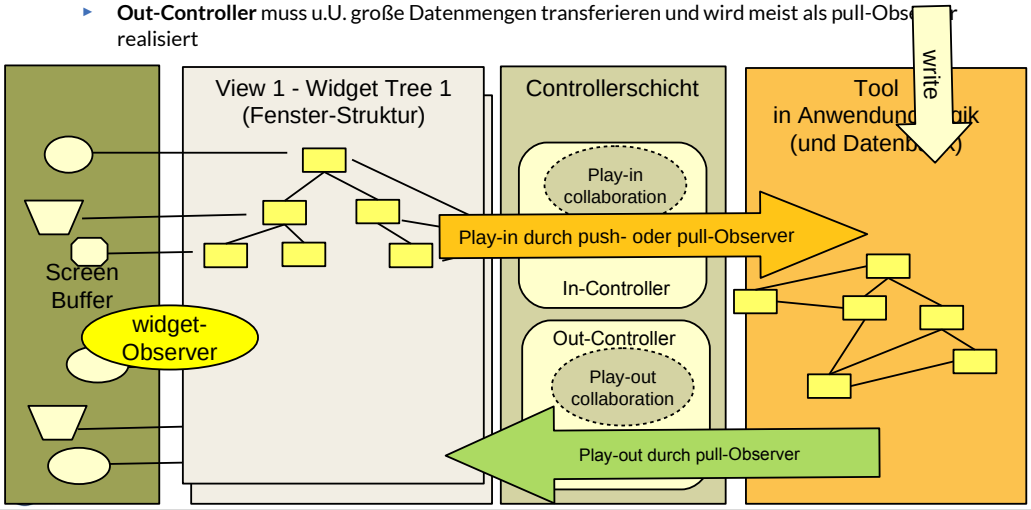
Phase 2) Netzaufbauphase: Aufbau der Verbindung

- 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) “Life” Überblick MVC Dynamik

- **Tool** ist passiv. Der Controller interpretiert die Eingaben und schreibt das Tool entsprechend
- **View** ist weitg. passiv. Controller benachrichtigt View, wenn sich was im Tool 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-Obs[illegible]r realisiert



Die Dynamik der Phase 3 (“Life”)

Phase 3 (Dynamik) trennt zwischen Ereignisverarbeitung und Datentransport

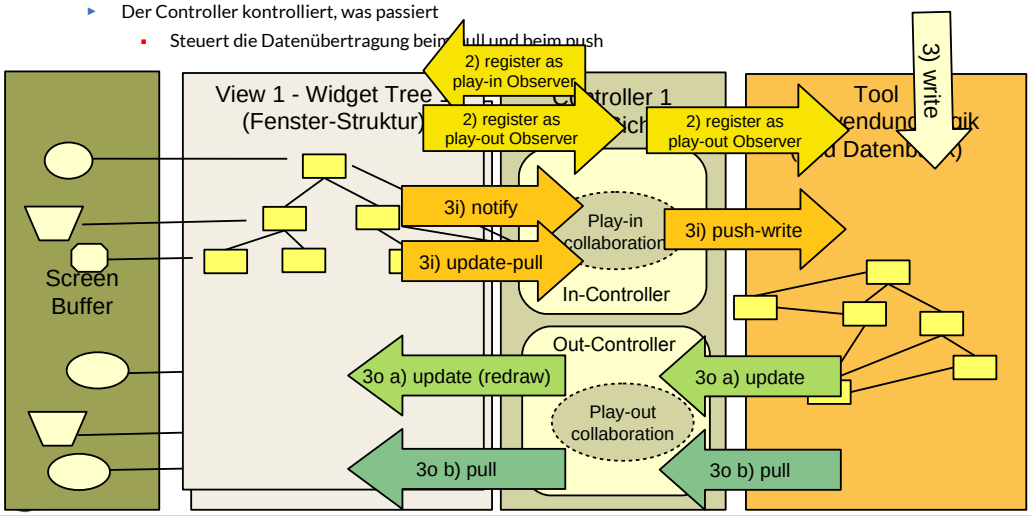
Die Konnektoren setzen push- oder pull-Observer-Muster ein

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)

- Tool 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 Tool 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]