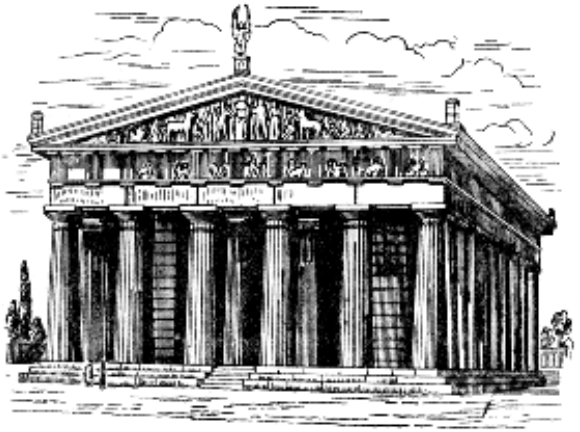
Softwarearchitektur

(Architektur: *αρχή* = Anfang, Ursprung + *tectum* = Haus, Dach)

3. Beschreibungstechniken



Vorlesung Wintersemester 2008 / 2009

Technische Universität München
Institut für Informatik
Lehrstuhl von Prof. Dr. Manfred Broy

Dr. Klaus Bergner, Prof. Dr. Manfred Broy, Dr. Marc Sihling

Inhalt

- Motivation und Grundlagen
- Ein Architekturbeispiel
- Graphische Beschreibungstechniken
  - Box-and-Line Diagramme
  - UML
- Beschreibungstechniken (Überblick und Beispiele)
  - Interface Description Language (IDL) und Programmiersprachen
  - Komponentenbasierte Spezifikation von Softwarearchitekturen
  - Architecture Description Languages (ADLs)
  - Formale Spezifikation von Softwarearchitekturen
- Zusammenfassung

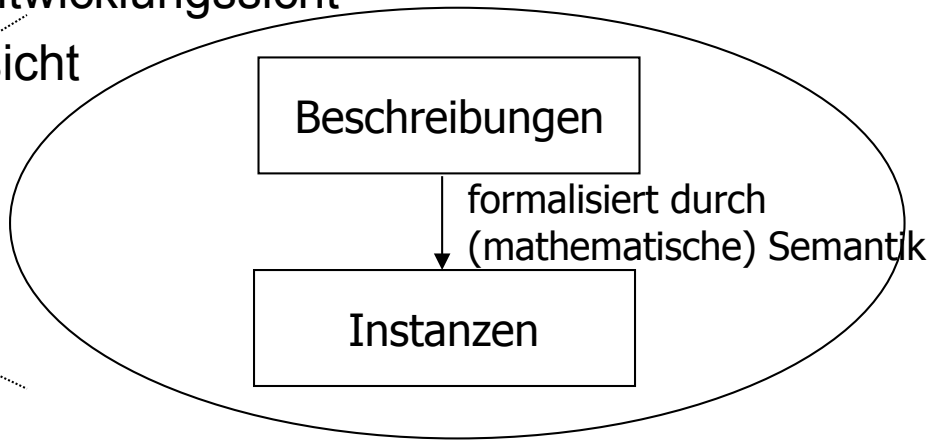
Inhalt

- Motivation und Grundlagen
- Ein Architekturbeispiel
- Graphische Beschreibungstechniken
  - Box-and-Line Diagramme
  - UML
- Beschreibungstechniken (Überblick und Beispiele)
  - Interface Description Language (IDL) und Programmiersprachen
  - Komponentenbasierte Spezifikation von Softwarearchitekturen
  - Architecture Description Languages (ADLs)
  - Formale Spezifikation von Softwarearchitekturen
- Zusammenfassung

Wiederholung: Beschreibung und Instanz

- Architekten sollten
  - Systemstrukturen und -Abläufe zur Laufzeit verstehen/entwerfen
  - Strukturen und Abläufe so beschreiben, dass sie von allen einheitlich interpretiert werden und auswertbar sind!
- Systemmodell enthält zwei Grundkonzepte
  - Beschreibungen repräsentieren Entwicklungssicht
  - Instanzen repräsentieren Laufzeitsicht

Mathematik
+ Bilder



- Lösung: Systemmodell als semantische Basis der Beschreibungs- /Spezifikationstechniken

Die zwei Ebenen des Systemmodells

beschreiben

Instanzen

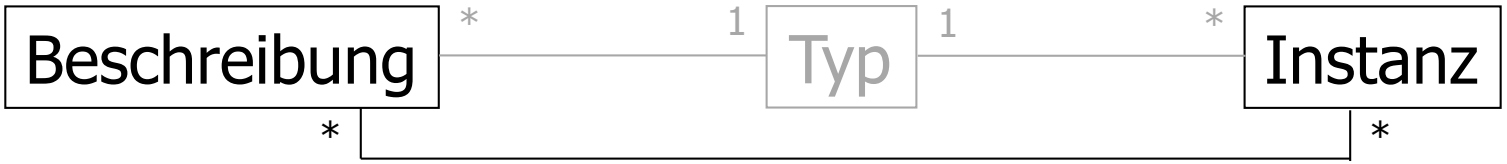
- repräsentieren Laufzeitsicht
- stellen konkrete Abläufe und Strukturen dar
- Darstellungen im Systemmodell sind oft umfangreich und unhandlich

Beschreibungen

- repräsentieren Entwicklersicht
- sind Prädikate z.B. über
  - der Struktur der Anwendung oder einzelner Komponenten
  - dem Verhalten von Komponenten
  - Protokollen auf einzelnen oder mehreren Nachrichtenkanälen

Beschreibungen und Typen

- Das Konzept eines Typs detailliert – ähnlich wie unsere Beschreibungen – Eigenschaften einer Gruppe von Instanzen
  - Typen bilden meist Klassen auf Instanzen (1-zu-n Beziehung) – eine Beschreibung umfasst dagegen nicht nur mehrere Instanzen, sondern eine Instanz kann auch von mehreren Beschreibungen detailliert werden (m-zu-n Beziehung)
  - Typen müssten gleichfalls beschrieben werden
  - Mit Beschreibungen könnten Typen dargestellt werden – ein eigenständiges Konzept ist somit nicht nötig



Beziehungen zwischen Beschreibungen

- Für den methodischen Umgang mit Beschreibungen sind ihre Beziehungen untereinander von großer Bedeutung:
  - Verfeinerung: eine Beschreibung geht von einer anderen aus und reichert diese um zusätzliche Aussagen an
  - Komposition: eine Beschreibung wird durch Zusammensetzen aus einer Menge weiterer Beschreibungen erzeugt
  - und einige mehr …
- Damit sind die Möglichkeiten geschaffen, um die vollständige Spezifikation eines Systems in optimaler Weise auf eine Menge von Beschreibungen aufzuteilen (Kunst!)
  - Ist die damit erreichte Beschreibung des Systems vollständig?
  - Widersprechen sich einzelne Beschreibungen?
- Offen:
  - Die Dokumentation aller sinnvollen Beschreibungen ist komplex und zeitaufwändig… ☹

Beschreibungsarten

- Für die Beschreibung von Softwarearchitekturen bieten sich neben Programmiersprachen
  - graphische Notationen
    - z.B. UML Diagramme, Box-and-Line Diagramme
  - semiformale Ansätze
    - z.B. UML (inkl. OCL)
  - formale Ansätze
    - z.B. Rapid, FOCUS

Anforderungen an Beschreibungen

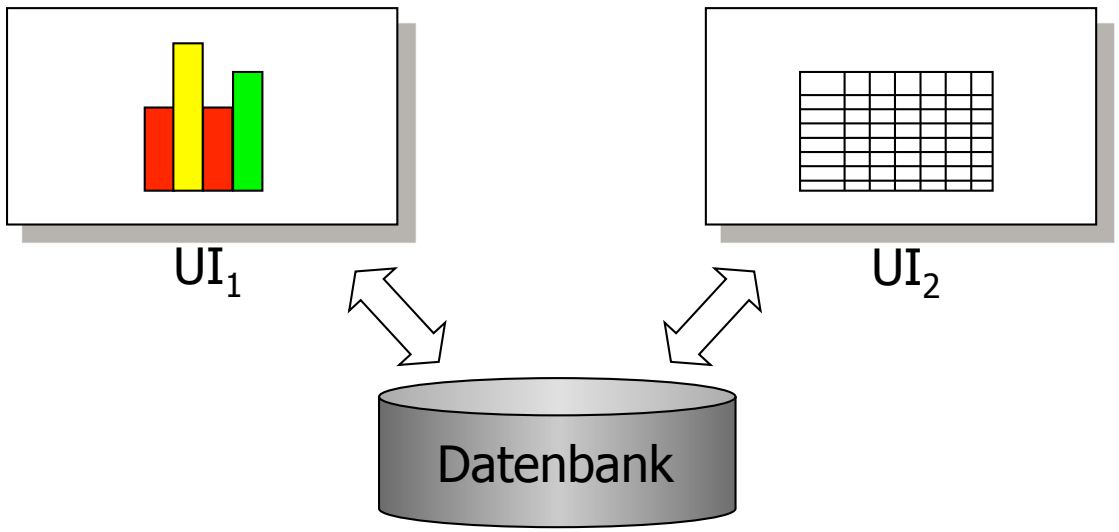
Gute Beschreibungen sind:

- Angemessen in der Wahl der eingesetzten Beschreibungstechniken. Die zu vermittelnde Information wird so optimal präsentiert.
- Aussagekräftig: Die Darstellung weist nur soviel Komplexität auf wie nötig
- Klar: Beschreibungen dienen der Kommunikation und müssen daher auch komplexe Sachverhalte schnell vermitteln können

Inhalt

- Motivation und Grundlagen
- Ein Architekturbeispiel
- Graphische Beschreibungstechniken
  - Box-and-Line Diagramme
  - UML
- Beschreibungstechniken (Überblick und Beispiele)
  - Interface Description Language (IDL) und Programmiersprachen
  - Komponentenbasierte Spezifikation von Softwarearchitekturen
  - Architecture Description Languages (ADLs)
  - Formale Spezifikation von Softwarearchitekturen
- Zusammenfassung

Beispiel: das Model-View-Controller Pattern [BMR+96]



Eine MVC-Architektur bewährt sich bei Informationssystemen

- mit mehreren, unterschiedlichen Benutzerschnittstellen
- mit dem Bedarf nach flexiblen Benutzerschnittstellen
- bei denen Veränderungen in der Datenbank sofort sichtbar sein müssen

Die Architektur besteht aus Model, View und Controller-Komponenten

Die MVC-Architektur: Model (CRC-Darstellung)

Komponente	**Zusammenspiel mit**
Model	▪ View ▪ Controller
Verantwortung ▪ Kapselt fachliche Objekte der Anwendung zusammen mit zugehöriger Basisfunktionalität ▪ Kennt seine Views und Controller, die sich an- und abmelden können ▪ Benachrichtigt angemeldete Komponenten von Zustands-änderungen	

Die MVC-Architektur: View (CRC-Darstellung)

Komponente	**Zusammenspiel mit**
View	▪ Model ▪ Controller
Verantwortung	
▪ Bereitet Informationen vom Model für den Benutzer auf und stellt diese geeignet dar ▪ Realisiert einen Update-Callback	

Die MVC-Architektur: Controller (CRC-Darstellung)

Komponente	**Zusammenspiel mit**
Controller	▪ Model ▪ View
Verantwortung	
▪ Akzeptiert Benutzereingaben als Nachrichten ▪ Übersetzt Nachrichten in Anfragen an Model oder View ▪ Realisiert einen Update-Callback	

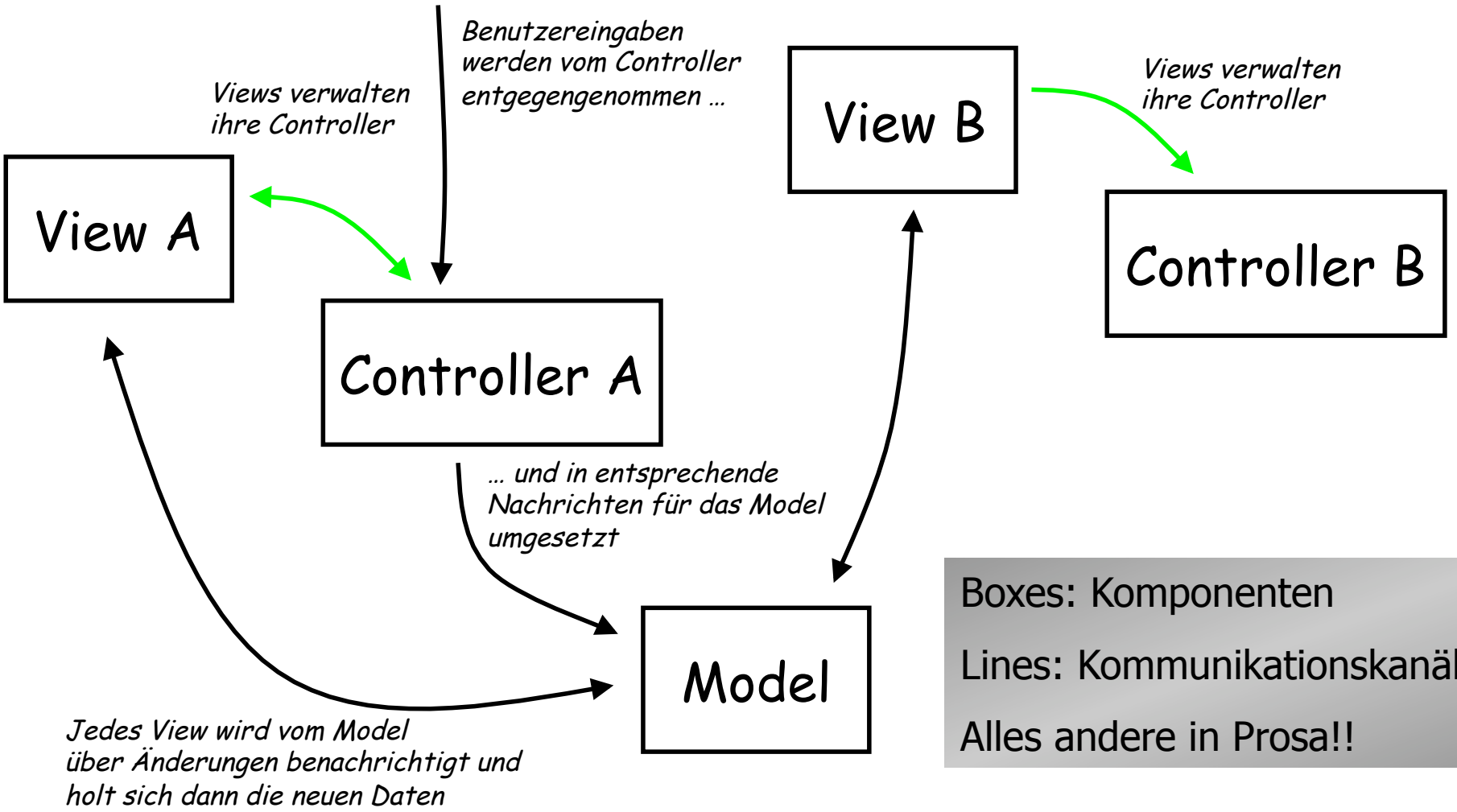
Darstellung der MVC-Architektur

- Beispiel: CRC-Karten (Class-Responsibility-Collaboration) verdeutlichen das Zusammenspiel zwischen den einzelnen Systemteilen
- Für ein klares Verständnis der Architektur fehlt jedoch
  - Darstellung der strukturellen Einschränkungen (beispielsweise die 1-zu-n Beziehung zwischen Model und View)
  - Präzise Festlegung der Abläufe zur Laufzeit (beispielsweise des Update-Mechanismus zwischen Model und View/Controller)
  - Signatur der Schnittstellen der Komponenten
  - Beispiele (sowohl Beispielabläufe als auch „Snapshots“, die die Struktur eines Systems zu einem bestimmten Zeitpunkt zeigen)
- Wie können diese Informationen geeignet durch graphische Darstellungen vermittelt werden?

Inhalt

- Motivation und Grundlagen
- Ein Architekturbeispiel
- Graphische Beschreibungstechniken
  - Box-and-Line Diagramme
  - UML
- Beschreibungstechniken (Überblick und Beispiele)
  - Interface Description Language (IDL) und Programmiersprachen
  - Komponentenbasierte Spezifikation von Softwarearchitekturen
  - Architecture Description Languages (ADLs)
  - Formale Spezifikation von Softwarearchitekturen
- Zusammenfassung

Box-and-Line Diagramme: Beispiel



Boxes: Komponenten

Lines: Kommunikationskanäle

Alles andere in Prosa!!

Box-and-Line Diagramme: Zusammenfassung

Box-and-Line Diagramme wurden ursprünglich mangels besserer Notationen für die Darstellung von Softwarearchitekturen verwendet

- Vorteile:
  - Pragmatische Darstellung für schnelle Skizzen (z.B. am Whiteboard während einer Besprechung)
- Nachteile:
  - Meist kein einheitliches Verständnis über die Bedeutung von „Boxes and Lines“
  - Erklärende Prosa gerät schnell sehr umfangreich

Einordnung

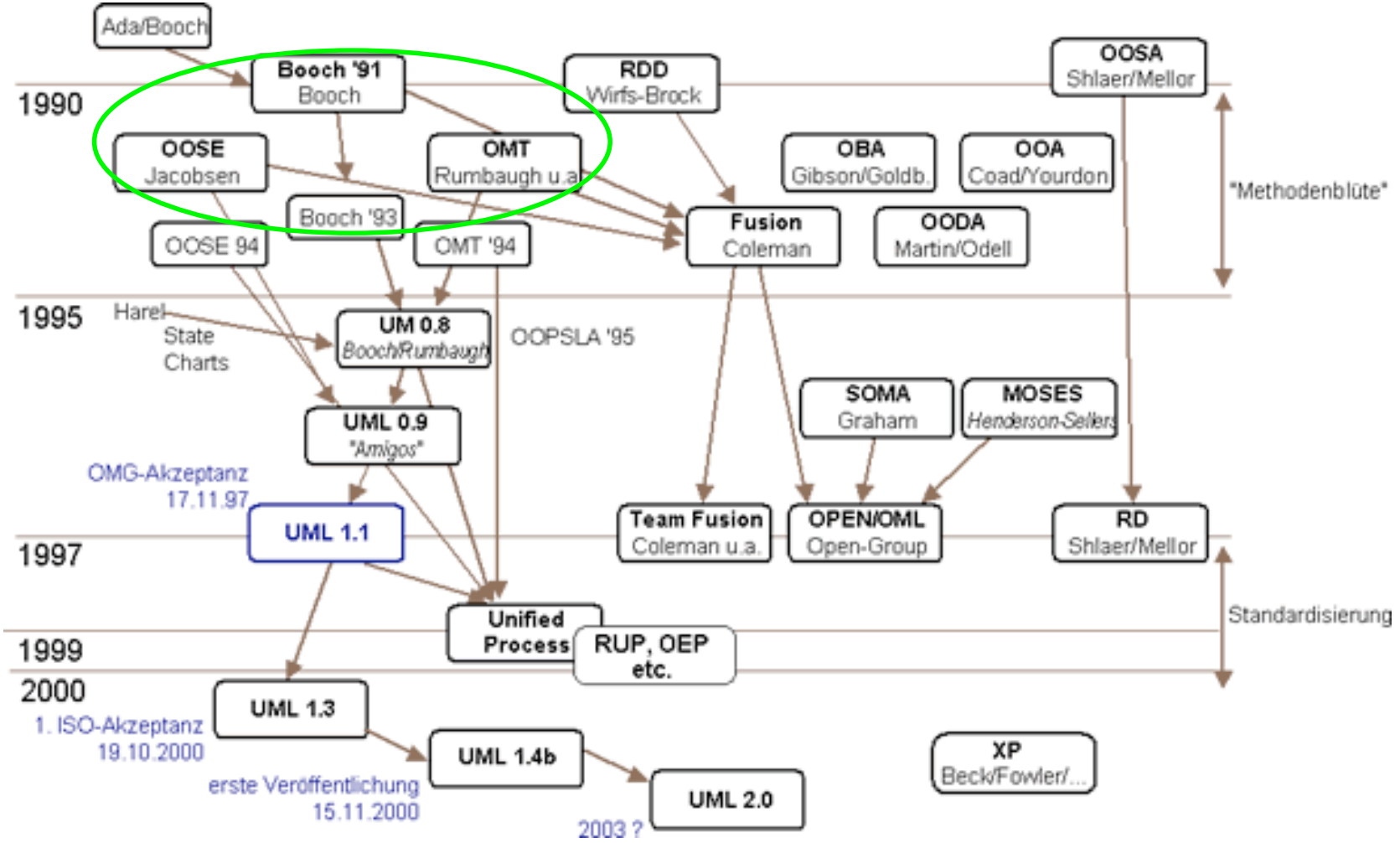
	Boxes and Lines
Komponenten	typischerweise als Kästchen
Schnittstellen	informelle Texte an den Pfeilen
Verbindungen	über Pfeile
Konfigurationen	über verschachtelte Kästchen
Strukturänderung	schlecht abbildbar
Kommunikation	informell beschrieben über Texte an den Pfeilen
Datenänderung	informell in den Texten

Inhalt

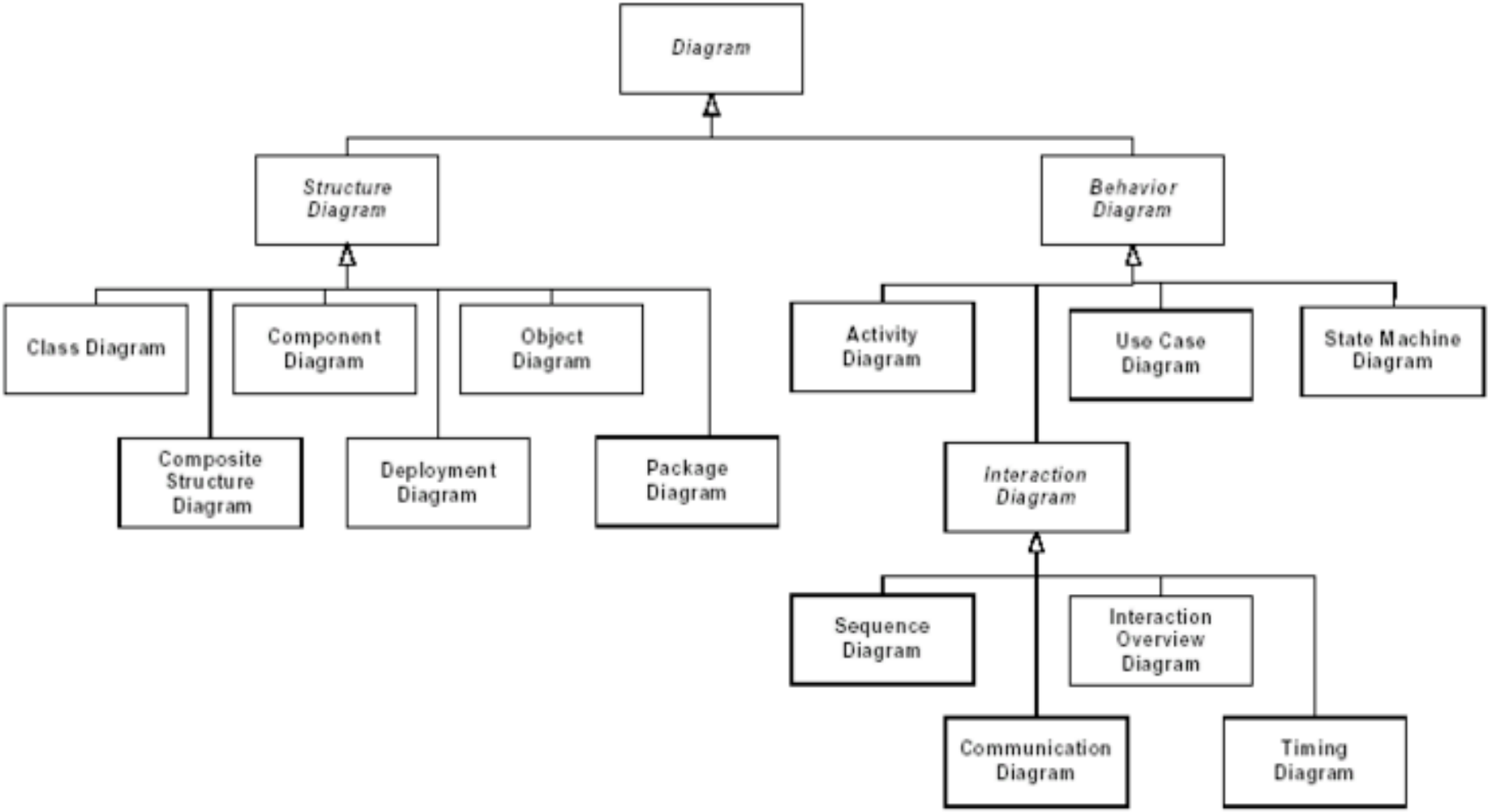
- Motivation und Grundlagen
- Ein Architekturbeispiel
- Graphische Beschreibungstechniken
  - Box-and-Line Diagramme
  - UML
- Beschreibungstechniken (Überblick und Beispiele)
  - Interface Description Language (IDL) und Programmiersprachen
  - Komponentenbasierte Spezifikation von Softwarearchitekturen
  - Architecture Description Languages (ADLs)
  - Formale Spezifikation von Softwarearchitekturen
- Zusammenfassung

UML: Entstehung

- Die „Unified Modeling Language“ geht auf die „3 Amigos“ zurück…



UML2.0 Diagramm-Arten (Überblick)



UML: Überblick (Auszug)

- Die UML bietet eine Fülle unterschiedlicher Diagrammtypen. Die Auswahl erfolgt nach Bedarf…

statisch

- Um ausgewählte Zustände oder Zustandsveränderungen als Folge einzelner „Snapshots“ darzustellen, eignen sich Instanzdiagramme
- Klassen- und Komponentendiagramme zeigen die statische Struktur eines Systems
- Für eine erste Präsentation der analysierten Funktionalitäten eines Systems bieten sich Anwendungsfalldiagramme an

dynamisch

- Beispielabläufe ergeben sich aus Sequenz- und Kommunikations-diagrammen als Zusammenspiel mehrerer Komponenten
- Das interne Verhalten einer Komponente wird durch Zustandsdiagramme abstrahiert

UML: Überblick (Auszug)

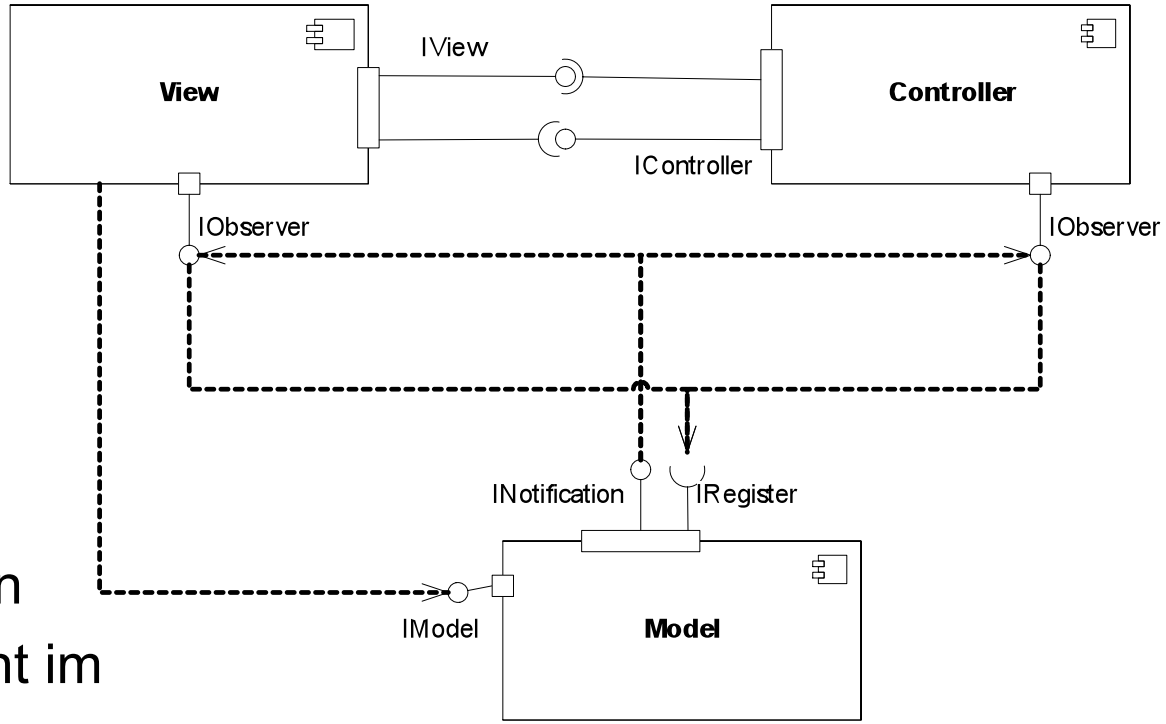
- Die UML bietet eine Fülle unterschiedlicher Diagrammtypen. Die Auswahl erfolgt nach Bedarf…
  - Um ausgewählte Zustände oder Zustandsveränderungen als Folge einzelner „Snapshots“ darzustellen, eignen sich Instanzdiagramme
  - Klassen- und Komponentendiagramme zeigen die statische Struktur eines Systems
  - Für eine erste Präsentation der analysierten Funktionalitäten eines Systems bieten sich Anwendungsfalldiagramme an
  - Beispielabläufe ergeben sich aus Sequenz- und Kommunikations-diagrammen als Zusammenspiel mehrerer Komponenten
  - Das interne Verhalten einer Komponente wird durch Zustandsdiagramme abstrahiert

statisch

dynamisch

UML: MVC – Gesamtheitliche Sicht

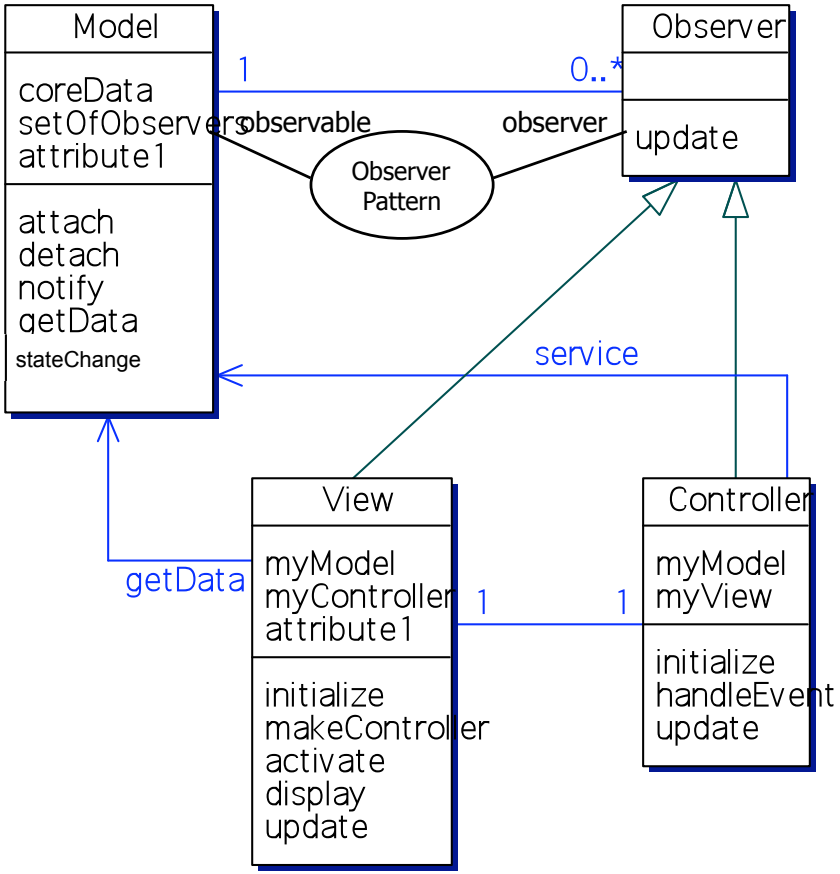
- Komponentendiagramm



- Beschreibt Strukturen kompositer „Bausteine“
  → Hierarchisierbar
- Kopplung über klar definierte Schnittstellen
- „Innenleben“ steht nicht im Vordergrund
  → statische Sicht
- Verfeinerung bspw. auf Klassen möglich

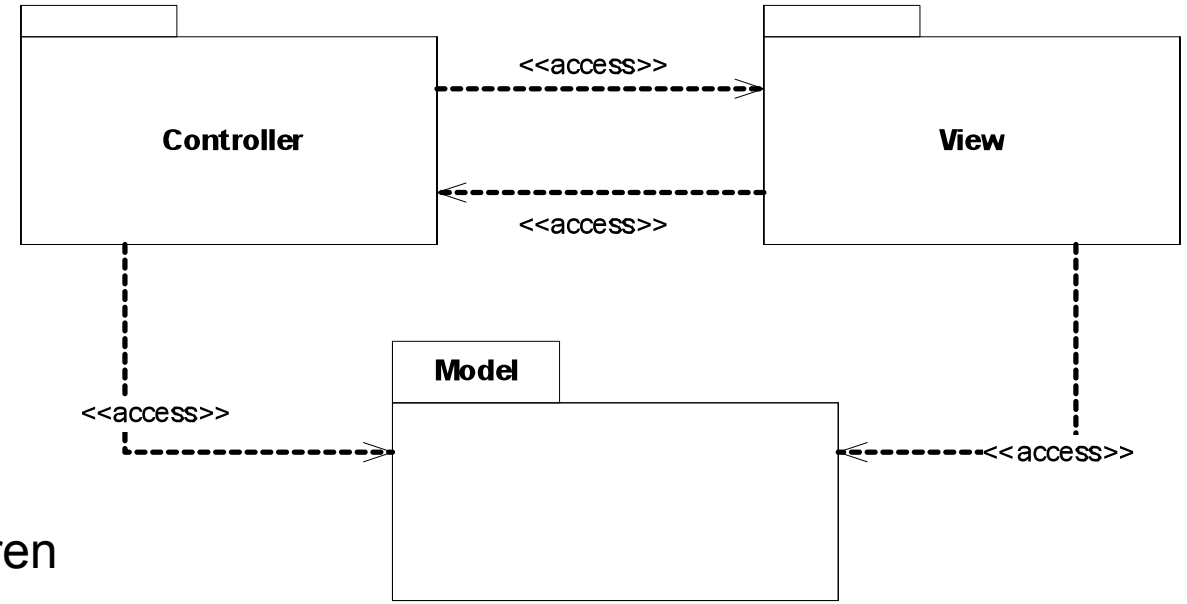
UML: Details als Klassendiagramm

- Diese Darstellung enthält weitaus mehr Informationen als das Box-and-Line Diagramm:
  - Verwendung des „Observer-Patterns“ als Benachrichtigungsmechanismus
  - Strukturelle Einschränkungen (z.B. 1-zu-1 Beziehung zwischen View und Controller)
  - Repräsentation des internen Zustands durch Attribute
  - Festlegung der Signatur
- Dennoch benötigen wir weitere Diagramme, z.B. zur Festlegung erlaubter Abläufe



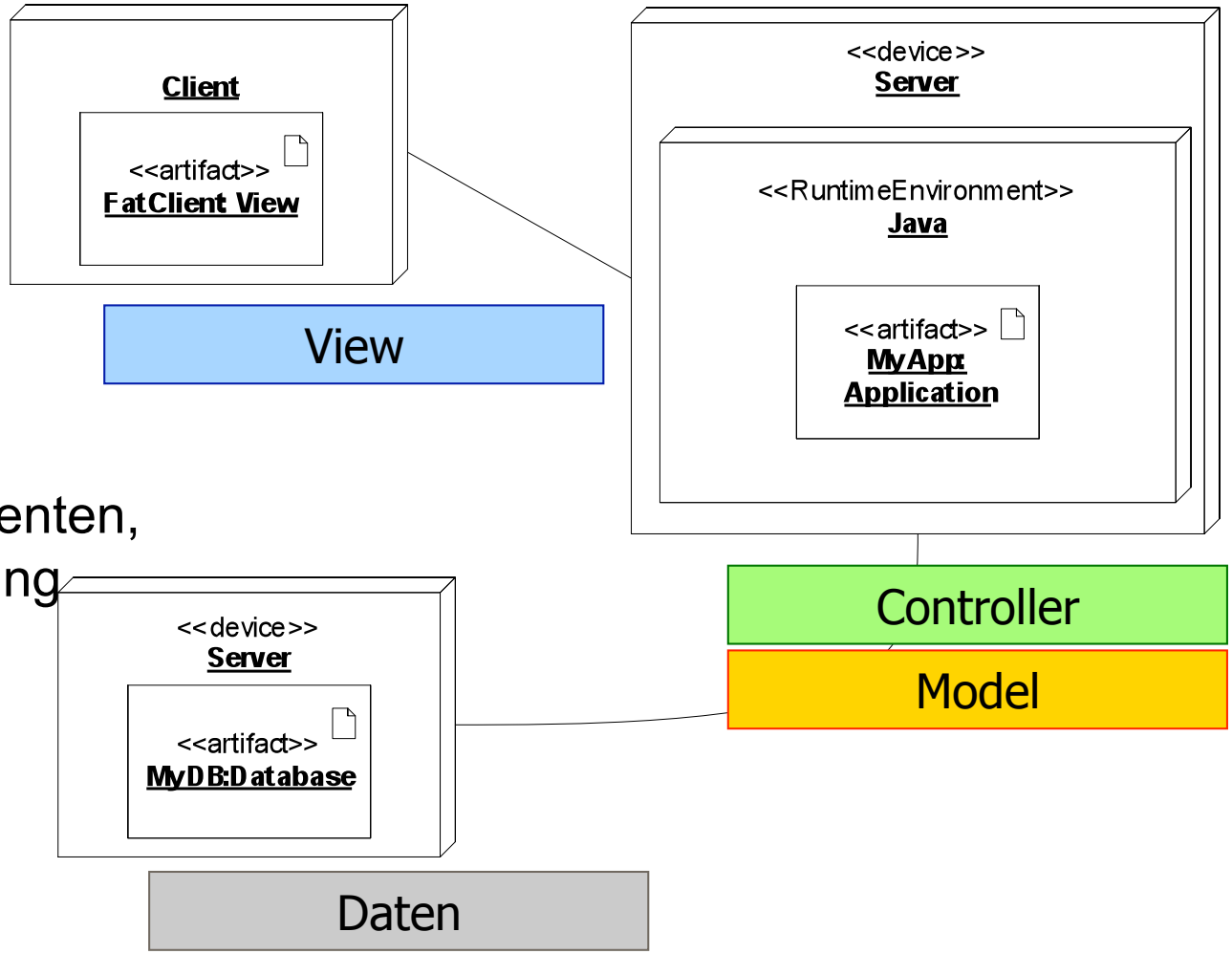
UML: MVC – Gesamtheitliche Sicht

- Paketdiagramm
  - Dienen zur abstrakten Gruppierung mehrerer Elemente (Komponenten, Klassen etc.)
  - Stellen Beziehungen auf grober Ebene dar
  - Können hierarchisch strukturiert werden
  - Lassen sich gut auf Komponentenstrukturen abbilden



UML: MVC – Gesamtheitliche Sicht

- Verteilungsdiagram
- Betrachtet die Verteilung von Software auf Hardwareeinheiten
- Systemsicht auf die (Hardware)komponenten, die an der Ausführung eines Systems beteiligt sind



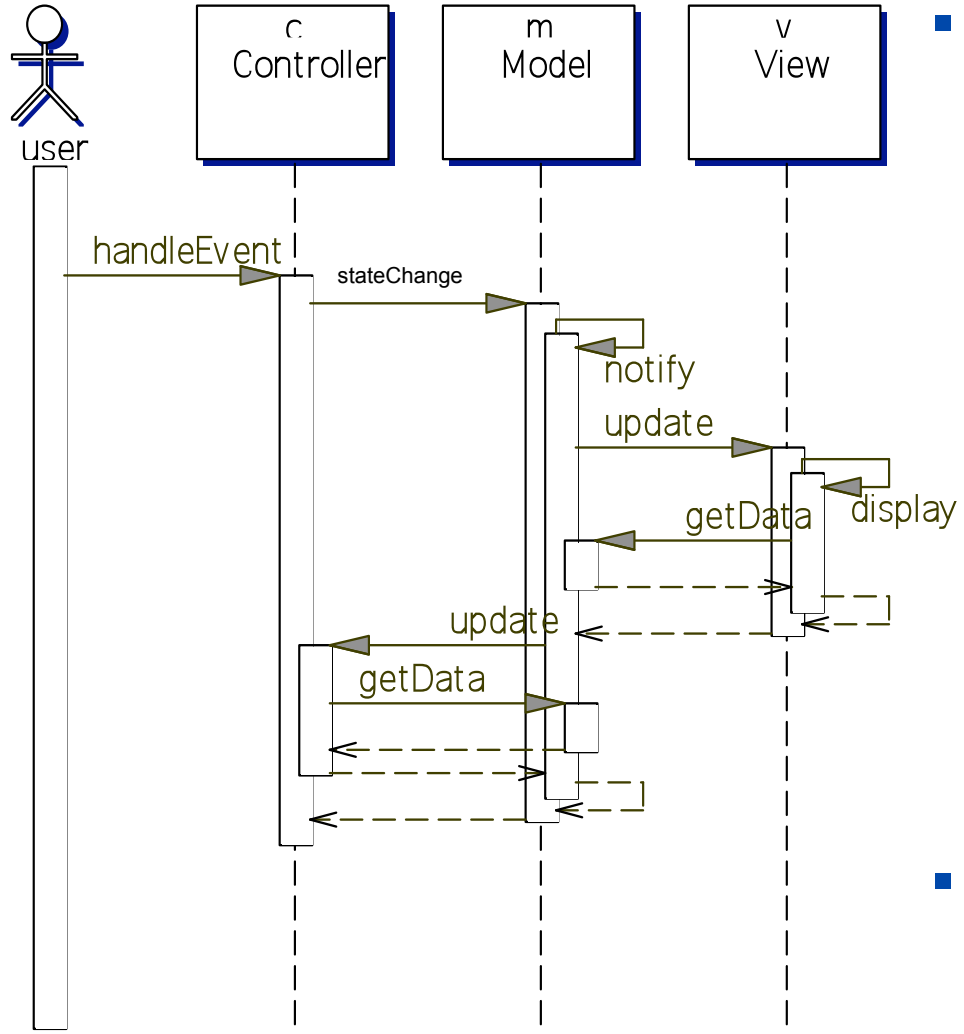
UML: Überblick (Auszug)

- Die UML bietet eine Fülle unterschiedlicher Diagrammtypen. Die Auswahl erfolgt nach Bedarf…
  - Um ausgewählte Zustände oder Zustandsveränderungen als Folge einzelner „Snapshots“ darzustellen, eignen sich Instanzdiagramme
  - Klassen- und Komponentendiagramme zeigen die statische Struktur eines Systems
  - Für eine erste Präsentation der analysierten Funktionalitäten eines Systems bieten sich Anwendungsfalldiagramme an
  - Beispielabläufe ergeben sich aus Sequenz- und Kommunikations-diagrammen als Zusammenspiel mehrerer Komponenten
  - Das interne Verhalten einer Komponente wird durch Zustandsdiagramme abstrahiert

statisch

dynamisch

UML: Beispielhafte Darstellung möglicher Abläufe



- Dieses Diagramm zeigt anhand eines Szenarios einen möglichen Ablauf der Kommunikation zwischen den Komponenten
  - der Controller überträgt eine Benutzereingabe in einen Funktionsaufruf des Models
  - die Zustandsänderung wird vom Model an alle registrierten Observer propagiert
  - Model und View holen sich die aktuellen Daten
- Für die Darstellung aller Einschränkungen war die UML 1.4 nicht immer ausreichend mächtig

UML: Object Constraint Language (OCL) [WK99]

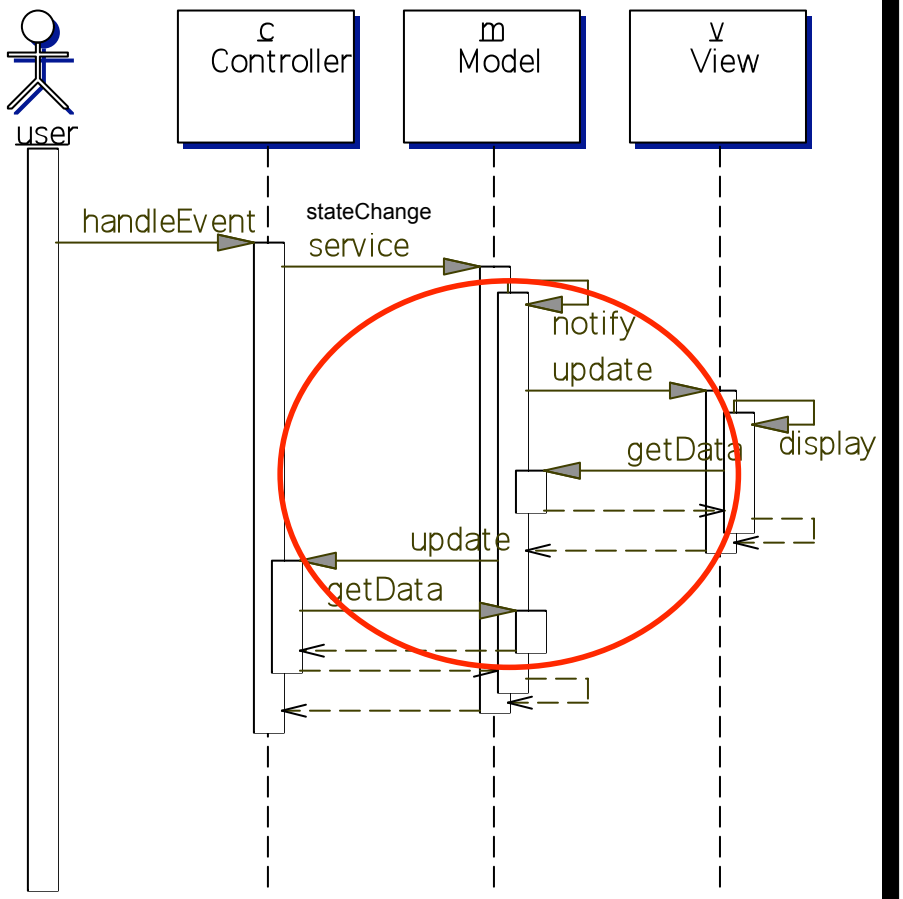
- Schnittstellen stellen Verträge zwischen den Komponenten dar, über die präzise Rechte und Verpflichtungen festgelegt sind („Design by Contract“)
- Die nötige Präzision und damit Eindeutigkeit erreicht die UML oftmals nur in Verbindung mit der OCL, einer textuellen Beschreibungssprache, mit der UML-Diagramme annotiert werden können
- OCL ist deklarativ sowie seiteneffektfrei und ermöglicht die Navigation über das Objektgeflecht
- Schnittstellenspezifikationen können mit Hilfe von OCL-Ausdrücken in Form von Invarianten sowie Vor- und Nachbedingungen zu Verträgen detailliert werden
- OCL-Ausdrücke werden in UML-Diagrammen als Notiz dargestellt und durch geeignete Stereotypen (z.B. <<postcondition>>) markiert.

OCL: Präzisierung der Model-Schnittstelle

- Die Tatsache, dass bei Anwendung des Observer-Patterns die update()-Methode bei genau allen registrierten Observern aufgerufen wird, kann – anstelle von Prosa – in OCL präzise definiert werden

- Eine entsprechende Annotation wäre etwa:

```
{setOfObservers->
 forall.update()}
```



UML: Zusammenfassung

UML Diagramme eignen sich mit Abstrichen für die Spezifikation von Softwarearchitekturen

- Vorteile:
  - Standardisierte Notation mit definierter Semantik
  - Weite Verbreitung und daher schnelles Verständnis
  - Umfassende Werkzeugunterstützung
- Nachteile:
  - Komplex

Einordnung

	UML2.0
Komponenten	Explizit im Meta-Modell - Umsetzung zur Laufzeit „aufgelöst" als Menge von Objekten (heutige Programmiersprachen) oder als „echte" Laufzeitkomponente
Schnittstellen	Explizit im Meta-Modell (Interfaces)
Verbindungen	Explizit im Meta-Modell (Associations)
Konfigurationen	Strukturen über Kompositionsstrukturdiagramme darstellbar
Strukturänderung	über Verhaltensdiagrammen mit operationaler Semantik beschreibbar
Kommunikation	vielfältige Kommunikationsprimitiven im Metamodell (Aufrufe, Signale)
Datenänderung	mit Hilfe von Zustandsdiagrammen darstellbar

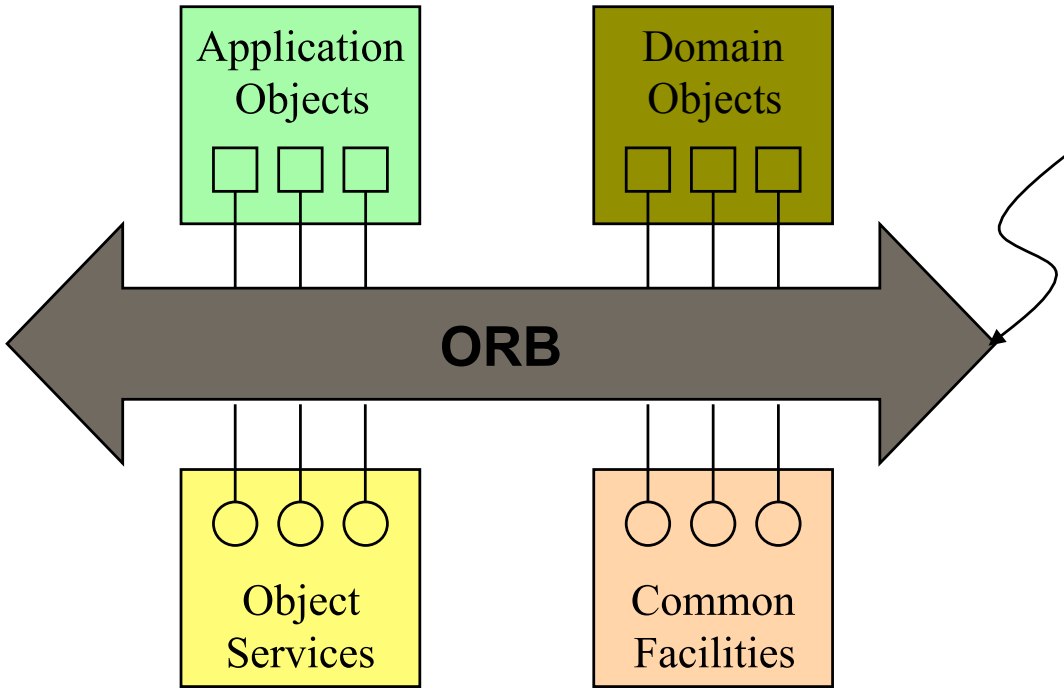
Inhalt

- Motivation und Grundlagen
- Ein Architekturbeispiel
- Graphische Beschreibungstechniken
  - Box-and-Line Diagramme
  - UML
- Beschreibungstechniken (Überblick und Beispiele)
  - Interface Description Language (IDL) und Programmiersprachen
  - Komponentenbasierte Spezifikation von Softwarearchitekturen
  - Architecture Description Languages (ADLs)
  - Formale Spezifikation von Softwarearchitekturen
- Zusammenfassung

Interface Description Language (IDL) und Programmiersprachen im Überblick

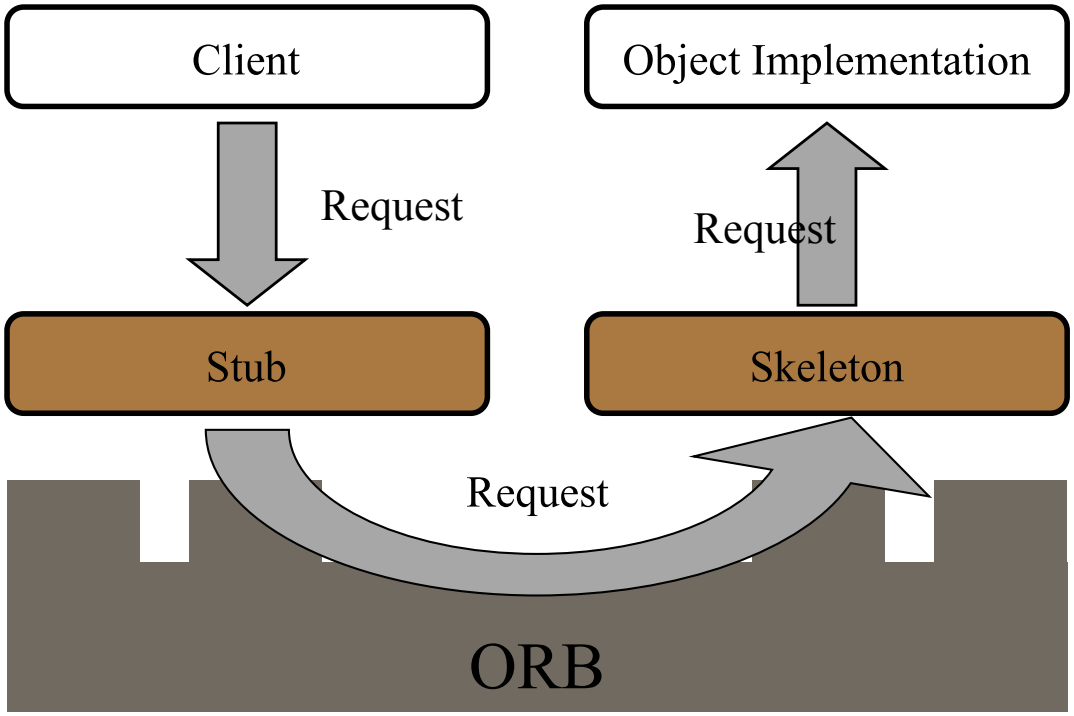
- Interface Description Languages (IDLs)
  - Spezifikationstechnik für Schnittstellen
  - Unabhängig von einer Programmiersprache
  - Abbildungen in die diversen Programmiersprachen
  - Werkzeuge und Infrastruktur verfügbar, z.B. Generatoren, Dienste, etc.
- Grundelemente einer IDL
  - Schnittstellen
  - Methoden
  - Attribute
  - Basis Domänen verfügbar (int, String, Sequenz, etc.)
  - Konstruktion komplexerer Strukturen möglich
- Es existieren eine Reihe von IDLs: RPC IDL, CORBA IDL, COM IDL, ...

MVC Beispiel: CORBA IDL und Java (1)



The ORB serves as the common backbone on which many discrete services can be integrated

MVC Beispiel: CORBA IDL und Java (2)

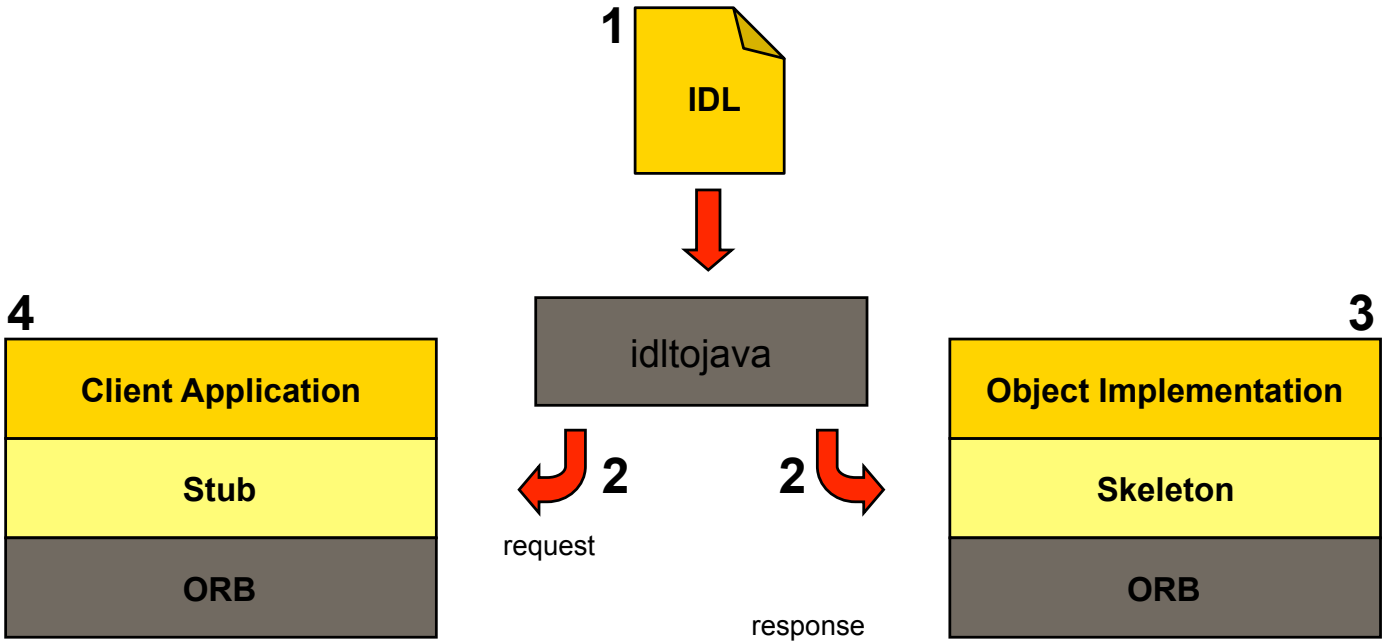


MVC Beispiel: CORBA IDL und Java (3)

```
module MVC {
  interface Observer {
    oneway void update ();
  };

  interface Observable {
    void register (in Observer obs);
    void unregister (in Observer obs);
  };

  interface Controller : Observer {
    void userInput (in String input);
  };
```

```
interface Model : Observable {
    void stateChange (in String newState);
  };

  interface View : Observer {
    void userOutput ();
  };

};
```

MVC Beispiel: CORBA IDL und Java (4)



steps:

1	write the IDL file
2	compile with idltojava (stubs/skeleton generated automatically)
3	write object implementation (servant)
4	write client application

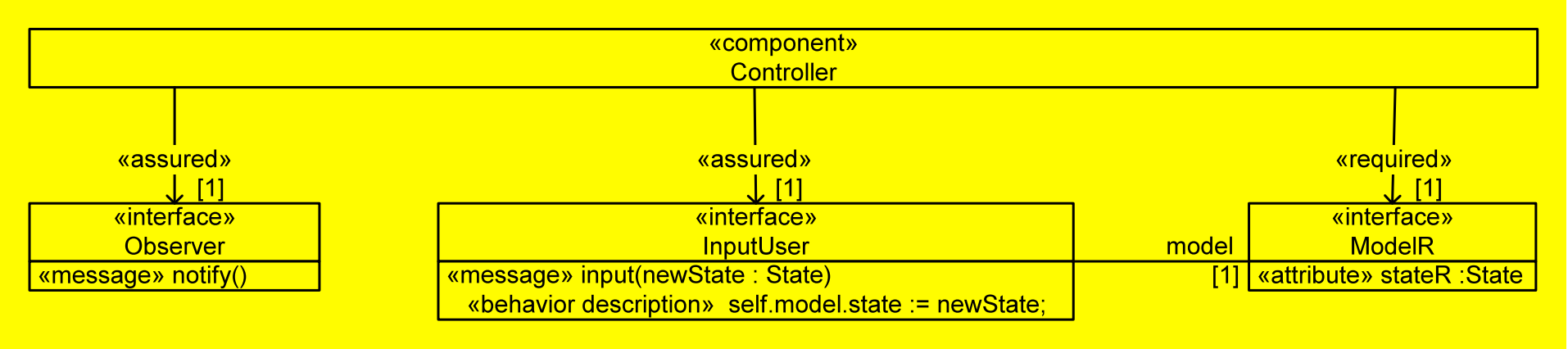
Einordnung

	CORBA IDL
Komponenten	Komponenten als Implementierung der Schnittstellen, somit nur in der Programmiersprache vorhanden
Schnittstellen	CORBA Interfaces
Verbindungen	CORBA Objektreferenzen
Konfigurationen	Nur auf Ebene der Programmiersprache
Strukturänderung	Nur auf Ebene der Programmiersprache
Kommunikation	Synchrone und asynchrone Methodenaufrufe
Datenänderung	Lesende und schreibende synchrone Zugriffe auf primitive und komplexe Datenstrukturen

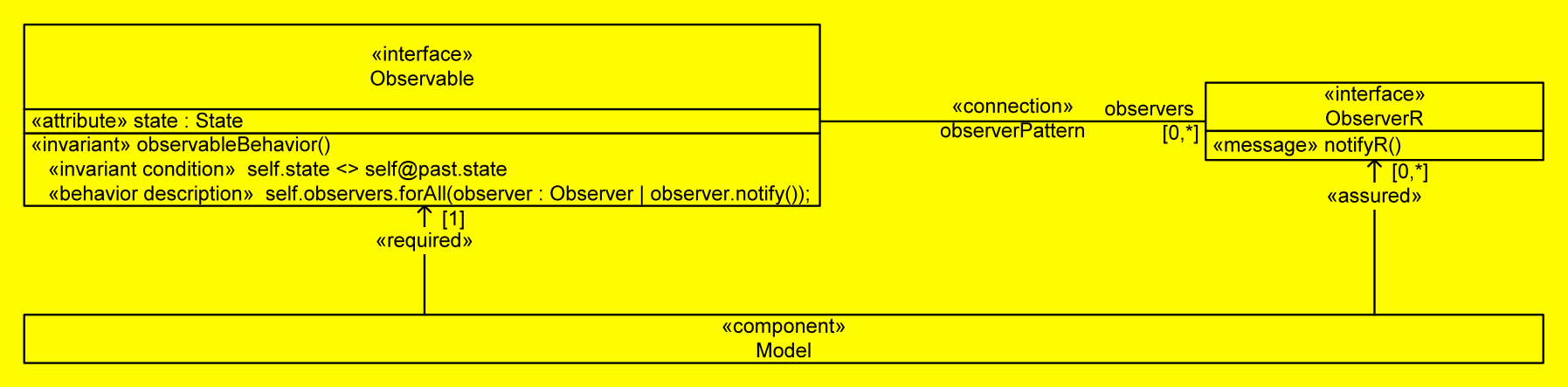
Komponentenbasierte Spezifikation von Softwarearchitekturen im Überblick

- Komponentenbasierte Softwarearchitekturen
  - Unabhängige, in sich abgeschlossene, vollständige und wiederverwendbare Komponenten (Code, Design, Hilfe für Anwender und Programmierer, etc.)
  - Eine Softwarearchitektur wird aus Komponenten komponiert
  - Infrastruktur und Werkzeuge verfügbar, z.B. EJB, .NET, etc.
- Grundelemente komponentenbasierter Softwarearchitekturen
  - Komponenten
  - Schnittstellen bzw. Ports
  - Protokolle und Verbindungen
- Eine spezifische Beschreibungstechnik steht nicht zur Verfügung; häufig werden Erweiterungen von UML verwendet

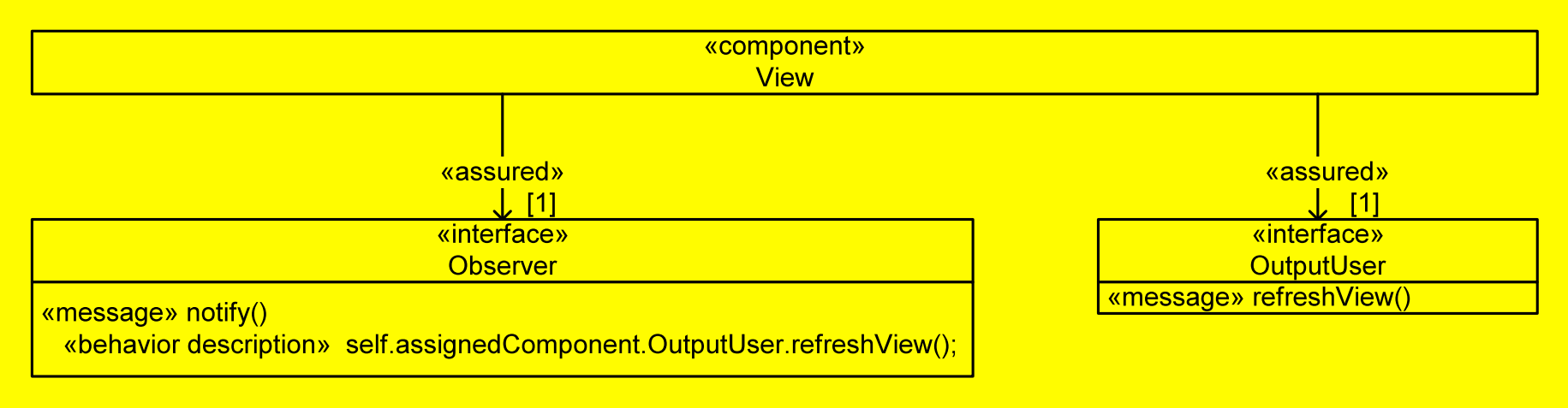
MVC Beispiel: DesignIt (1)



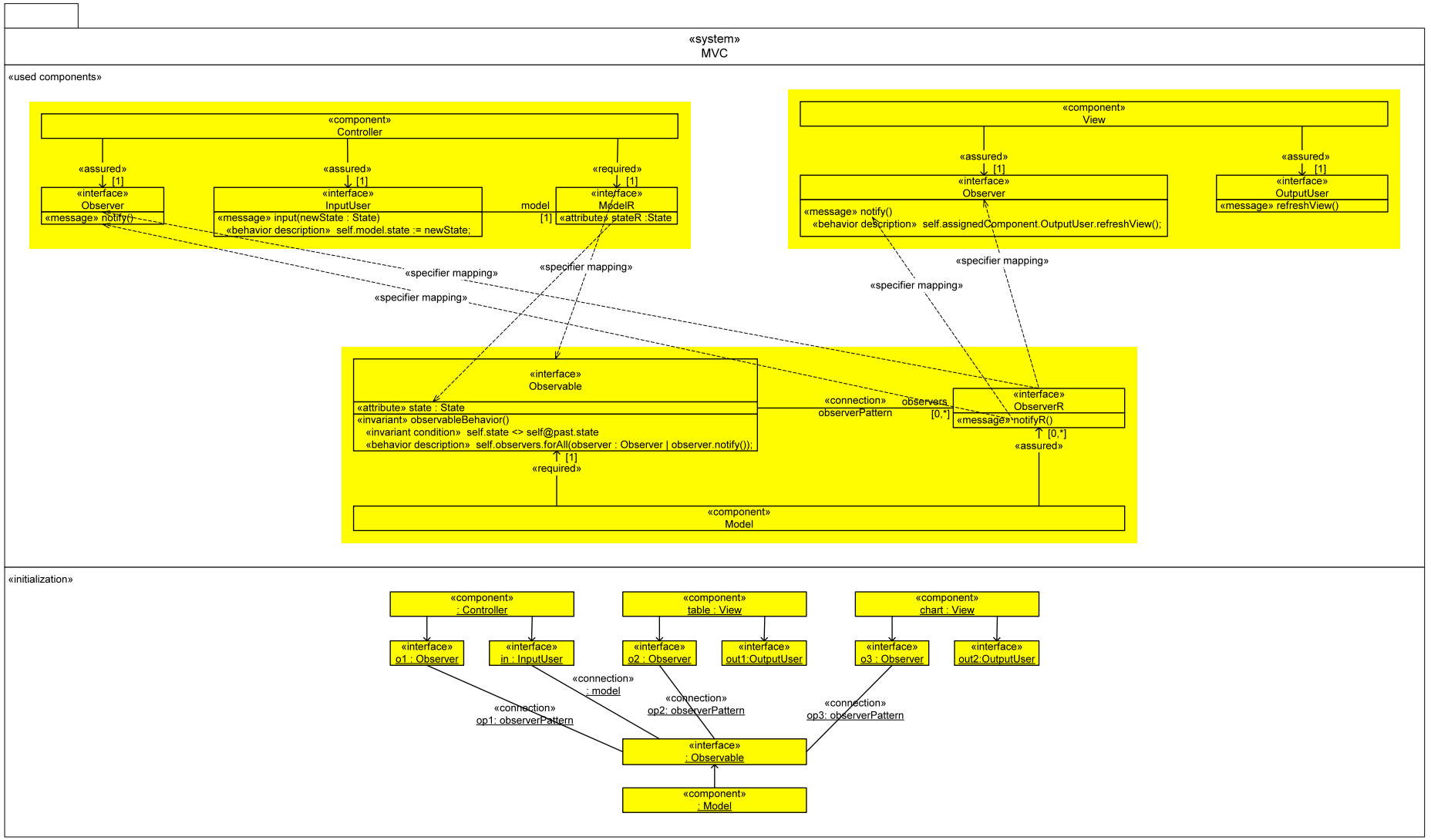
MVC Beispiel: DesignIt (2)



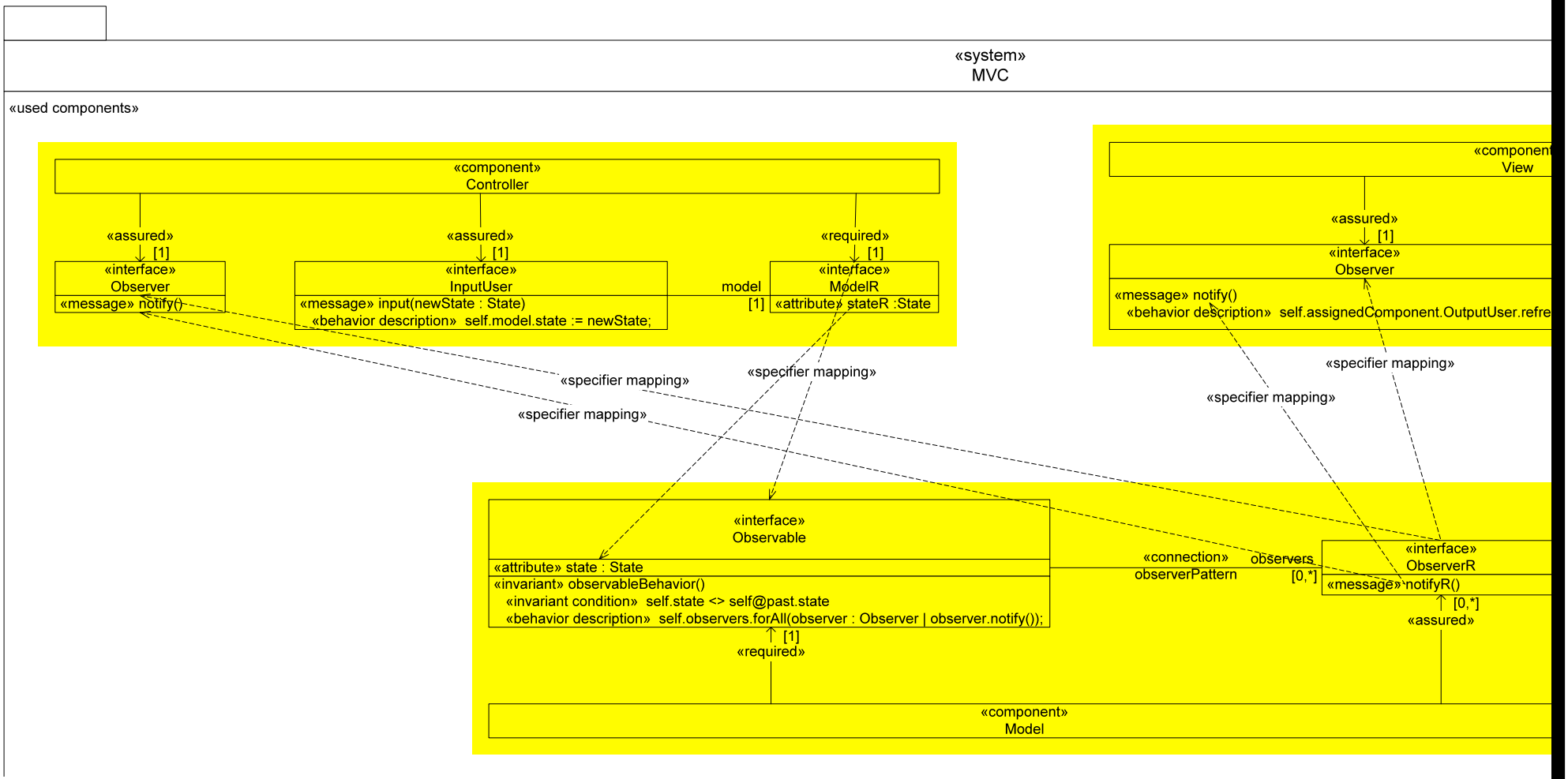
MVC Beispiel: DesignIt (3)



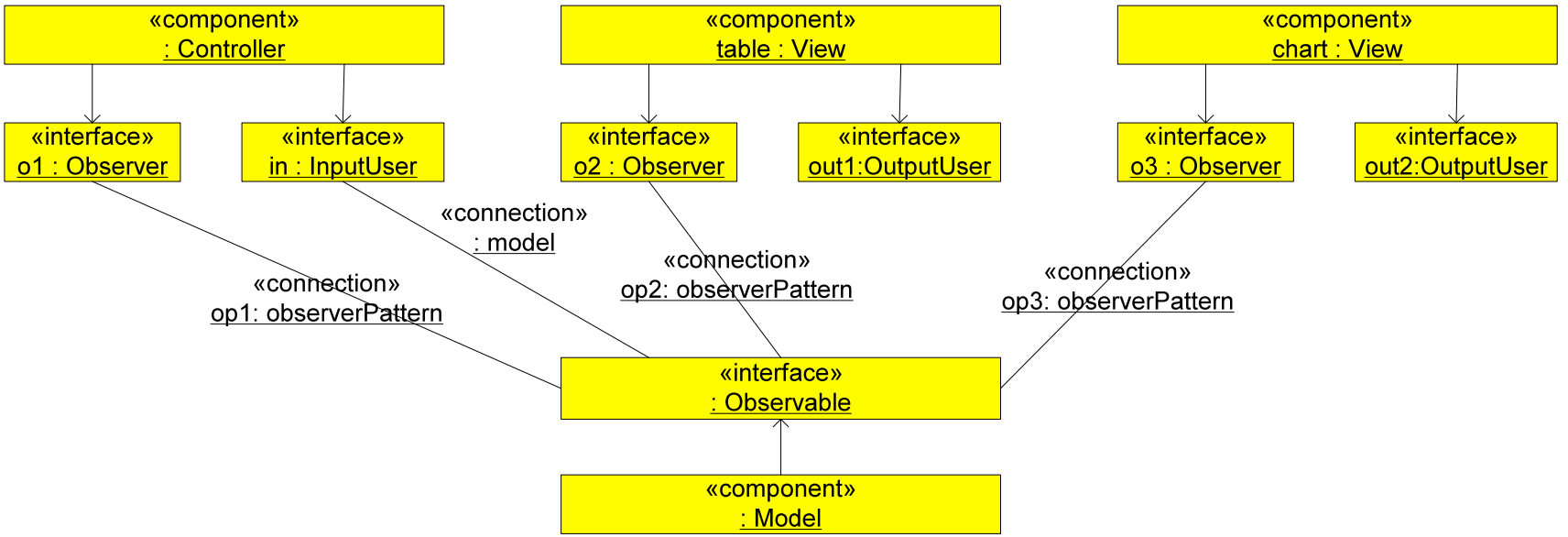
MVC Beispiel: DesignIt (4)



MVC Beispiel: DesignIt (5)



MVC Beispiel: DesignIt (6)



Einordnung

	DesignIt
Komponenten	Komponenten als Beschreibungskonzepte
Schnittstellen	Als Menge von Methoden und Attributen
Verbindungen	Verbindungen zwischen Schnittstellen
Konfigurationen	Auf Ebene der Komponenten und Systeme
Strukturänderung	Auf Ebene der Komponenten und Systeme
Kommunikation	Synchrone und asynchrone Methodenaufrufe
Datenänderung	Lesende und schreibende synchrone Zugriffe auf primitive und komplexe Datenstrukturen

Inhalt

- Motivation und Grundlagen
- Ein Architekturbeispiel
- Graphische Beschreibungstechniken
  - Box-and-Line Diagramme
  - UML
- Beschreibungstechniken (Überblick und Beispiele)
  - Interface Description Language (IDL) und Programmiersprachen
  - Komponentenbasierte Spezifikation von Softwarearchitekturen
  - Architecture Description Languages (ADLs)
  - Formale Spezifikation von Softwarearchitekturen
- Zusammenfassung

Architecture Description Languages (ADLs) im Überblick

- Architecture Description Languages (ADLs)
  - Spezielle Sprache für die Beschreibung von Architekturen
  - Semi-formale Spezifikationstechnik
  - Meist unterstützt durch Analyse- und Entwicklungswerkzeuge
- Grundelemente einer ADL
  - Komponenten (Components)
  - Verbindern (Connectors)
  - Konfigurationen (Configurations)
- Es existieren viele spezifische ADLs: ACME, Aesop, C2, Darwin, Koala, MetaH, Rapide, SADL, UniCon, Weaves, Wright, xADL, ...

ADL	ACME	Aesop	C2	Darwin	MetaH	Rapide	SADL	UniCon	Weaves	Wright
Focus	Architectural interchange, predominantly at the structural level	Specification of architectures in specific styles	Architectures of highly-distributed, evolvable, and dynamic systems	Architectures of highly-distributed systems whose dynamism is guided by strict formal underpinnings	Architectures in the guidance, navigation, and control (GN&C) domain	Modeling and simulation of the dynamic behavior described by an architecture	Formal refinement of architectures across levels of detail	Glue code generation for interconnecting existing components using common interaction protocols	Data-flow architectures, characterized by high-volume of data and real-time requirements on its processing	Modeling and analysis (specifically, deadlock analysis) of the dynamic behavior of concurrent systems

MVC Beispiel: Rapide (1)

```
type Subscriber is interface
     requires function retrieve_info() return Data;
     action in notify_subscr;

behavior
     notify_subscr ⇒ retrieve_info();

constraint
     match (notify_subscr -> retrieve_info())^(*~);

end;

type Publisher is interface
     provides function retrieve_info() return Data;
     action out notify;

end;
```

MVC Beispiel: Rapide (2)

```
type Model is interface
     include Publisher;
     action in f; -- fachliche Aktion

behavior
          f ⇒ notify;

end;

type View is interface
     include Subscriber;
     action in show_view;

end;

type Controller is interface
     include Subscriber;
     action in user_command;
     action out show_view;
     action out f;

end;
```

MVC Beispiel: Rapide (3)

```
architecture MVC is

        M: Model;
        C: Controller;
        V: View;

    connect

        C.show_view to V.show_view;
        C.f to M.f;

        V.retrieve_info to M.retrieve_info;
        C.retrieve_info to M.retrieve_info;

        M.notify to V.notify_subscr;
        M.notify to C.notify_subscr;

end MVC;
```

MVC Beispiel: UniCon (1)

```
COMPONENT Model
      INTERFACE IS
            TYPE Computation
            PLAYER Notify IS RoutineCall
            SIGNATURE(...) END Notify
            PLAYER RetrieveInfo IS RoutineDef
            SIGNATURE(...) END RetrieveInfo
      ...
      END INTERFACE

      IMPLEMENTATION IS
            VARIANT Model_C++ IN 'model.cxx'
            IMPLTYPE(Source) /* C++ */
            END Model_C++
      END IMPLEMENTATION

END Model
```

MVC Beispiel: UniCon (2)

```
COMPONENT MVC
     INTERFACE IS
          TYPE Computation
          PLAYER Start IS RoutineDef
          SIGNATURE(...)
          ...
     END INTERFACE

     IMPLEMENTATION IS

          USES M INTERFACE Model VARIANT(Model_C++)
          USES V INTERFACE View
          USES C INTERFACE Controller

          BIND Start TO Controller.Start

          ESTABLISH Call WITH
          M.Notify AS caller
          V.Notify AS definer
          END Call

          ESTABLISH Call WITH
          M.Notify AS caller
          C.Notify AS definer
          END Call
          ...
     END IMPLEMENTATION

END MVC
```

MVC Beispiel: UniCon (3)

```
CONNECTOR Call
      PROTOCOL IS
            TYPE ProcedureCall
            ROLE definer IS Definer
            END definer
            ROLE caller IS Caller
            END caller
      ...
      END PROTOCOL

      IMPLEMENTATION IS
            BUILTIN
      END IMPLEMENTATION

END Call
```

Einordnung

	Rapide	UniCon
Komponenten	Nur auf Ebene der Programmiersprache	Components
Schnittstellen	Interfaces	Interfaces mit vorgegebenen Typen
Verbindungen	Connections zwischen den Interfaces	Verbindungen und Protokolle
Konfigurationen	Hierarchische Dekomposition von Interfaces	Hierarchische Dekomposition von Components
Strukturänderung	Spezielle Sprachelemente vorhanden	Nur auf Ebene der Programmiersprache
Kommunikation	Synchroner und asynchroner Methodenaufruf möglich	Nur auf Ebene der Programmiersprache
Datenänderung	Primitive Daten können über synchronen Methodenaufruf abgebildet werden	Nur auf Ebene der Programmiersprache

Inhalt

- Motivation und Grundlagen
- Ein Architekturbeispiel
- Graphische Beschreibungstechniken
  - Box-and-Line Diagramme
  - UML
- Beschreibungstechniken (Überblick und Beispiele)
  - Interface Description Language (IDL) und Programmiersprachen
  - Komponentenbasierte Spezifikation von Softwarearchitekturen
  - Architecture Description Languages (ADLs)
  - Formale Spezifikation von Softwarearchitekturen
- Zusammenfassung

Formale Spezifikation von Softwarearchitekturen im Überblick

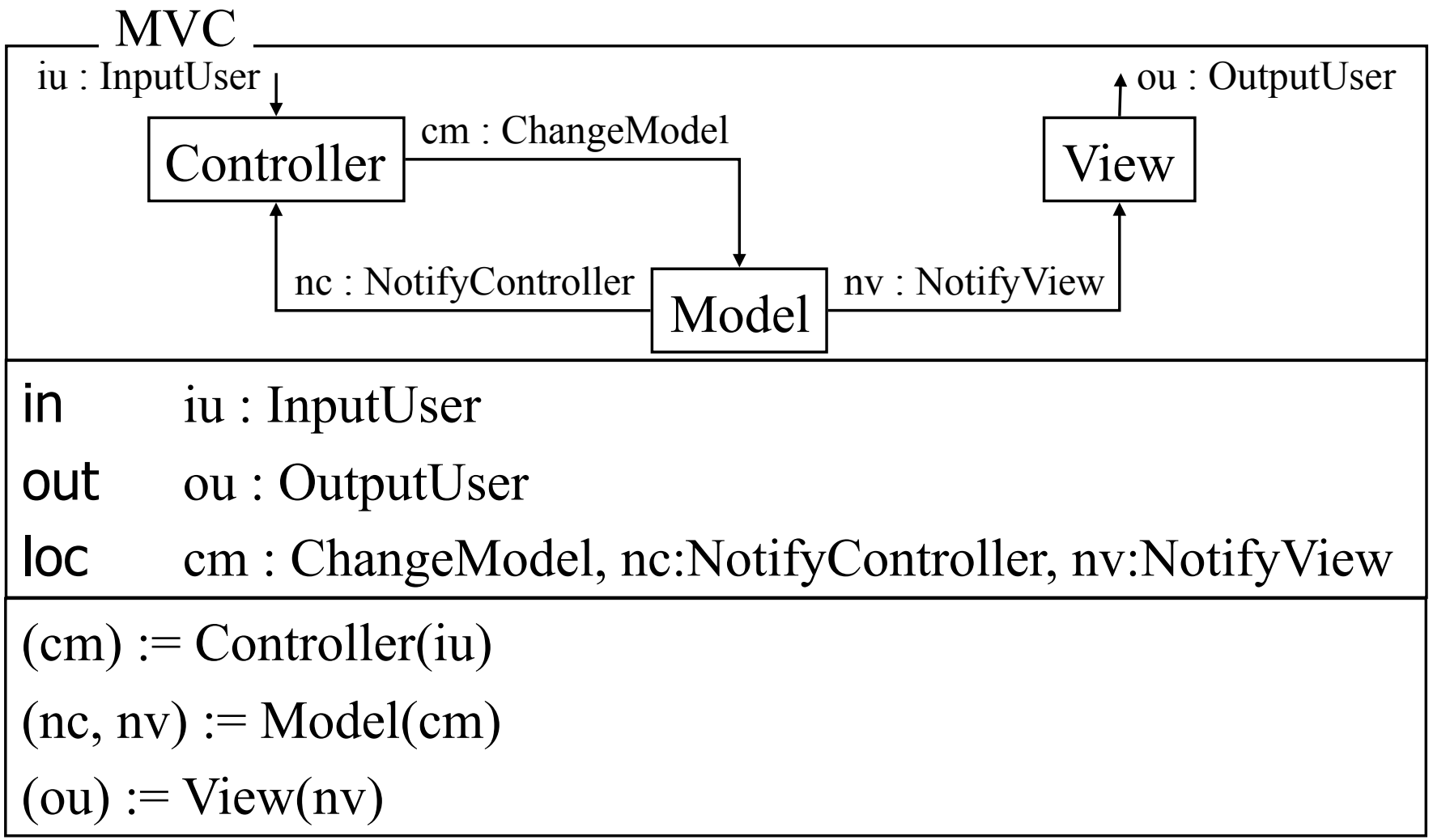
- Formale Spezifikation
  - Mit Hilfe der Mathematik werden präzise Spezifikationen erstellt
  - Aussagen über die Spezifikation lassen sich mathematisch beweisen
  - Verifikation der Korrektheit einer Implementierung ist möglich
  - Vollständige Generierung der Implementierung ist möglich
- Grundelemente einer formalen Spezifikation
  - Definition von Mengen
  - Relationen und Funktionen auf Mengen
- Es existieren eine Reihe von formalen Spezifikations-techniken: Temporal Logic, Π-Kalkül, Z, Focus, Communicating Sequential Processes, ...

Kernproblematik der formalen Spezifikation von Architektur

- Struktur-, Komponenten- und Schnittstellenmodell
- Spezifikation des Komponenten- und Schnittstellen-verhaltens
  - Zustandsmaschinen
  - Ein-/Ausgabefunktion oder -relation

    In Focus: Eine (deterministische) Komponente ist eine Funktion die Ströme von Eingabesignalen/-nachrichten auf Ströme von Ausgabesignalen/-nachrichten abbildet (kann auch durch Zustandsübergangsdiagramm mit Ein-und Ausgabe, eine Moore/Mealy-Maschine dargestellt werden)
- Spezifikation des Architekturverhaltens abgestützt auf den Komponenten- und Schnittstellenbegriff
  - Aus Komponentenmodellen wird Architekturverhalten konstruiert
  - Aus der Zustandsmaschine für Komponenten wird die Zustandsmaschine der Architektur
  - Aus Funktionen über Strömen wird eine Familie von Strömen in der Architektur

MVC Beispiel: Focus



Einordnung

	Focus
Komponenten	Components
Schnittstellen	Ports
Verbindungen	Connections zwischen Ports
Konfigurationen	Hierarchische Dekomposition von Components
Strukturänderung	Keine
Kommunikation	Asynchroner Nachrichtenaus-tausch zwischen Components
Datenänderung	Daten und deren Änderung können implizit über die Kommunikationshistorie beschreiben werden

Inhalt

- Motivation und Grundlagen
- Ein Architekturbeispiel
- Graphische Beschreibungstechniken
  - Box-and-Line Diagramme
  - UML
- Beschreibungstechniken (Überblick und Beispiele)
  - Interface Description Language (IDL) und Programmiersprachen
  - Komponentenbasierte Spezifikation von Softwarearchitekturen
  - Architecture Description Languages (ADLs)
  - Formale Spezifikation von Softwarearchitekturen
- Zusammenfassung

Zusammenfassung (1)

- Beschreibungen dienen dem Architekten in der Entwicklersicht als Mittel, Aussagen bzw. Einschränkungen über das Anwendungssystem zu formulieren
- Für die Dokumentation und die Kommunikation im Entwicklerteam haben sich graphische Beschreibungstechniken bewährt
- Die UML erfreut sich auch hier großer Beliebheit
- In der Vorlesung nutzen wir eine an der UML angelehnte Notation, die wir beispielhaft mit einer Semantik versehen haben

Zusammenfassung (2)

- Instanzen repräsentieren die Laufzeitsicht
- Beschreibungen repräsentieren Entwicklungssicht
- Ein Typ repräsentiert eine Menge von Instanzen mit gleichen Eigenschaften
- Beschreibungen (bzw. Spezifikationen) beschreiben Instanzen bzw. Typen
- Architekturbeschreibungen bzw. –Spezifikationen dienen dem Architekten für den Entwurf (Spielwiese), für die Kommunikation und für die Dokumentation
- Eine Standardbeschreibungstechnik für Softwarearchitekturen existiert (noch) nicht
- Beschreibungstechniken sind Kommunikationsmittel
- Wähle je nach Zielgruppe und Architekturausprägung die „passende“ Auswahl an Beschreibungselementen

Literatur

- [BRS97] Klaus Bergner, Andreas Rausch, Marc Sihling: Using UML for Modeling a Distributed Java Application, Technischer Bericht der Universität München, TUM-I9735, http://wwwbib.informatik.tu-muenchen.de/infberichte/1997/TUM-I9735.ps.gz, 1997
- [BCK+98] Len Bass, Paul Clements, Rick Kazman, Ken Bass: *Software Architecture in Practice (Sei Series in Software Engineering)*, Addison-Wesley Publishing, 1998
- [DSB99] Desmond Francis D'Souza, Aarnod Sane, Alan Birchenough: First Class Extensibility for UML – Packaging of Profiles, Stereotypes, Patterns, in: <<UML>>'99 – The Unified Modeling Language, LNCS 1723, 1999
- [DW98] Desmond Francis D'Souza, Alan Cameron Wills. *Objects, Components, and Frameworks With UML: The Catalysis Approach*, Addison Wesley Publishing Company, 1998
- [HNS99] Christine Hofmeister, Robert Nord, Dilip Soni: *Applied Software Architecture*, Addison Wesley – Object Technology Series, 1999
- [WK99] Jos Warmer und Anneke Kleppe, *The Object Constraint Language: Precise Modeling with UML,* Addison-Wesley, 1999

Literatur

- [SGW94] Bran Selic, Garth Gullekson, Paul T. Ward. Real-Time Object-Oriented Modeling. John Wiley & Sons. 1994
- [Rau01] Andreas Rausch. Towards a Software Architecture Specification Language based on UML and OCL. Proceedings of the Workshop on Describing Software Architecture with UML, 23rd International Conference on Software Engineering. 2001
- [BDR+97] Manfred Broy, Ernst Denert, Klaus Renzel, Monika Schmidt. Software Architectures and Design Patterns in Business Applications, Technischer Bericht der Technischen Universität München, TUM-I9746, 1997
- [BS01] Manfred Broy, Ketil Stølen. Specification and Development of Interactive Systems. Springer Verlag. 2001
- [SG96] Mary Shaw, David Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall. 1996