

# 32. Strukturelle Analyse (Systemanalyse für Kontextmodell und Top-Level-Architektur)

## Mit UML-Komponenten

1

Prof. Dr. rer. nat. Uwe Aßmann  
Institut für Software- und Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
Version 15-0.5, 22.06.15

- 1) UML-Komponenten
  - 2) Kontextmodell
  - 3) Top-level Architektur (TLA)
  - 4) Asynchrone Systemmerkmale
  - 5) Anhänge
- 1) Adapter in der TLA

- ▶ ST für Einsteiger, Kap. Klassendiagramme
- ▶ Zuser, Kap. 7-9
- ▶ Störrle Kap. 6 (!!)
- ▶ Balzert Kap. 6-7, 9-10

# Überblick Teil III:

## Objektorientierte Analyse (OOA)

3

1. Überblick Objektorientierte Analyse
  1. Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodelldrierte Modellierung mit UML
  1. Analyse des Domänenmodells: Strukturelle metamodelldrierte Modellierung
    1. Modellierung von komplexen Objekten
  2. Systemanalyse: Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen (Verhaltensmodell)
  1. Funktionale Verfeinerung: Dynamische Modellierung von Lebenszyklen mit Aktionsdiagrammen
  2. Funktionale querschneidende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen
4. Beispiel Fallstudie EU-Rent

4

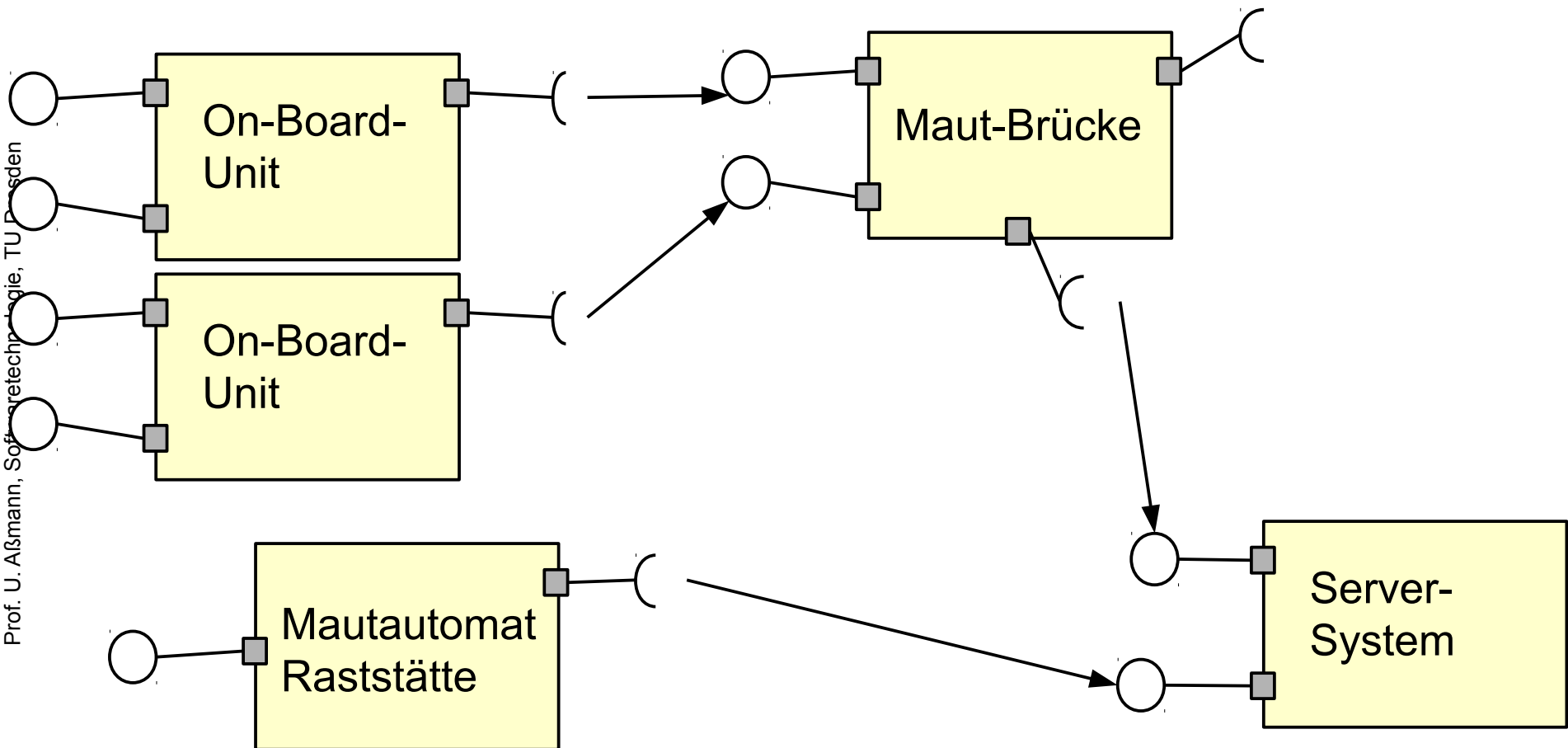
- ▶ Auftragnehmer Telekom und Daimler Financial Services
- ▶ Schätzung der Einnahmen auf 3 Mrd jährlich
- ▶ Inbetriebnahme geplant 31.8.2003, dann 2.11. 2003
  - Tägliche Vertragsstrafe 250k€
- ▶ Realisiert 1.1. 2005, vollständig 1.1.2006
- ▶ Vertrag mit allen Anlagen und Nebenvereinbarungen aus 17.000 Seiten.
  - Der Kernvertrag, in dem Fragen der Haftung, Vertragsstrafen und Kündigungsfristen geregelt sind, umfasst 190 Seiten.



# Warum war TollCollect verspätet? (Beobachtungen)

5

- ▶ Die Integrationsarbeiten auf oberster Ebene wurden unterschätzt
- ▶ Die Interaktionen zwischen den Systemen (ihren Kontextmodellen) wurden zu spät auf Skalierbarkeit getestet



Notation: UML Komponenten

# Schritte der strukturellen, metamodelgetriebenen Analyse

6

- ▶ gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur

strukturelle OOA

Klassen identifizieren

Klassen nach Profilen klassifizieren

Merkmale identifizieren

Attribute identifizieren

Operationen identifizieren

Ereignisse und Ausnahmen identifizieren

Ströme und Kanäle identifizieren

Klassenbeziehungen identifizieren

Assoziationen ident.

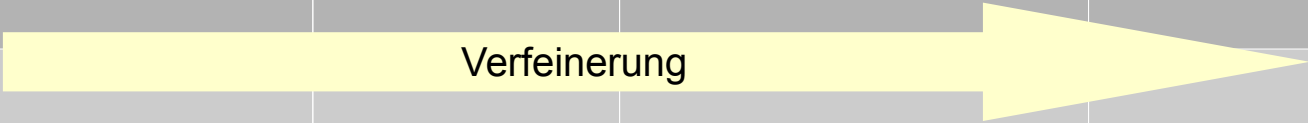
Assoziationsklassen id.

Vererbungen ident.

Komplexe Klassen ident.

Kanäle ident.

# Q5: Schritte der Modellierung in Bezug auf Schichten des Systems

	Schichten	Analyse	Entwurf	Feinentwurf	Implementierung
Benutzungs-schnittstelle (Boundary)	GUI				
	Controller				
Anwendungs-logik (Control)	Kontextmodell	Aufstellung aus Domänenmodell; Erarbeitung System-schnittstellen	stabil	Umsetzen auf jUML	stabil
	Top-Level-Architektur	Verfeinerung des Kontextmodells	stabil	Umsetzen auf jUML	stabil
	Architektur		Ausarbeitung Architektur (PSM)	Einziehen von Plattformabhängigkeiten (PSM); Umsetzen auf jUML; Serialisierung	Details ausfüllen, Methoden ausprogrammieren
	Tools		Ausarbeitung Tools	Einziehen von Plattformabhängigkeiten (PSM); Umsetzen auf jUML; Serialisierung	Details ausfüllen, Methoden ausprogrammieren
Datenhaltung (Database)	Material	Aufstellung aus Domänenmodell		Einziehen von Plattformabhängigkeiten (PSM); Umsetzen auf jUML;	Details ausfüllen, Methoden ausprogrammieren

# Die Fragen der Systemanalyse im Kundengespräch

8

Die obersten Schichten eines Systems sind meist hierarchisch gestaltet.

- ▶ Welche Komponenten hat das System?
- ▶ Welche Dienste soll das System liefern? (Schnittstellen)
- ▶ Welche Daten sollen in das System fließen? (Streams, Senken)
  - Datentypen, die in und aus dem System fließen, gehören ins Domänenmodell
- ▶ Wie kommuniziert das System mit anderen Systemen? (Kanäle, Konnektoren)
- ▶ Welche Ereignisse treten außerhalb des Systems auf und wie fließen sie in das System? (Ereigniskanäle)
- ▶ Welche Ereignisse treten innerhalb des Systems auf und wie reagiert das System? (Ausnahmen, exceptions)



# 32.1 UML-Komponentenklassen für Großobjekte (Hierarchische komplexe Klassen)

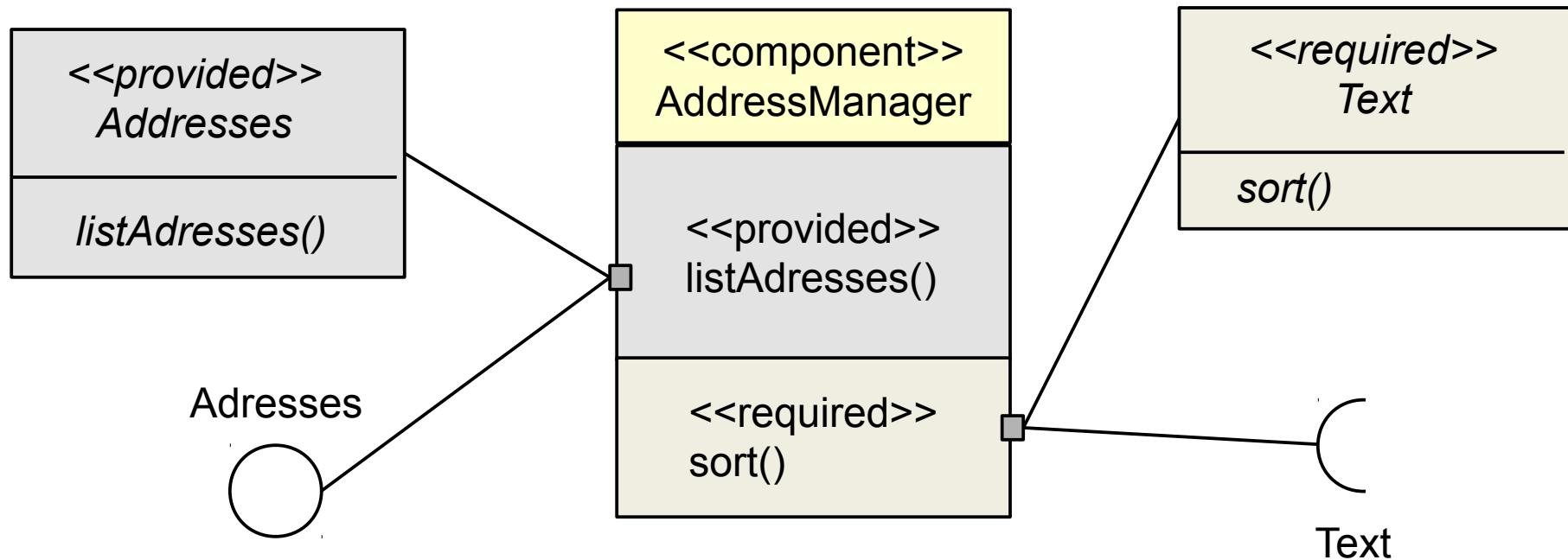
9

- UML-Komponenten sind hierarchisch aggregierte komplexe Klassen mit *angebotenen* und *benötigten* Schnittstellen

# Lollipops and Plugs (Balls and Sockets)

10

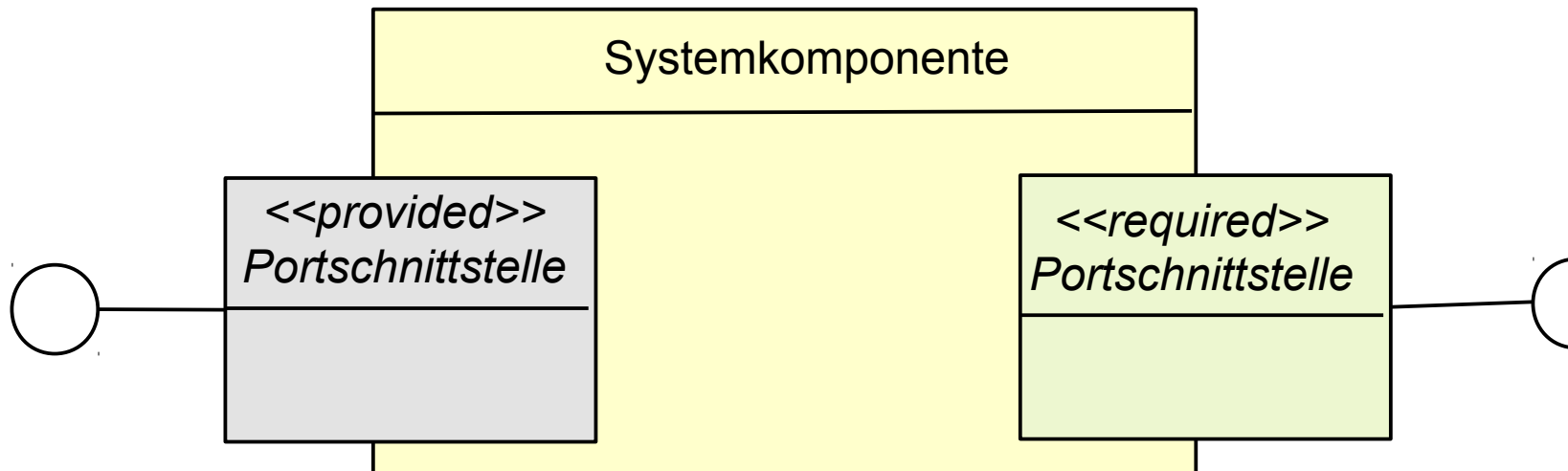
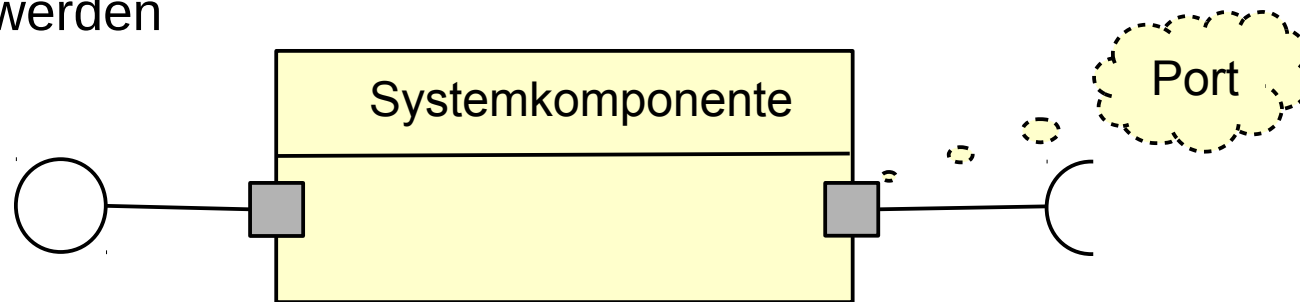
- ▶ Klassen, die *angebotene (provided)* von *benötigten (required)* Schnittstellen unterscheiden, heißen **Komponenten**
  - Eine benötigte Schnittstelle spezifiziert, welche Partnerklasse die Klasse zum Ausführen benötigt (separates Compartment)
    - In Java wird die benötigte Schnittstelle *nicht spezifiziert*, sondern vom Übersetzer herausgefunden
  - Klassen mit angebotener und benötigter Schnittstelle sind einfacher wiederzuverwenden, da alle Aufrufabhängigkeiten explizit sind



# Anschlüsse (Ports)

11

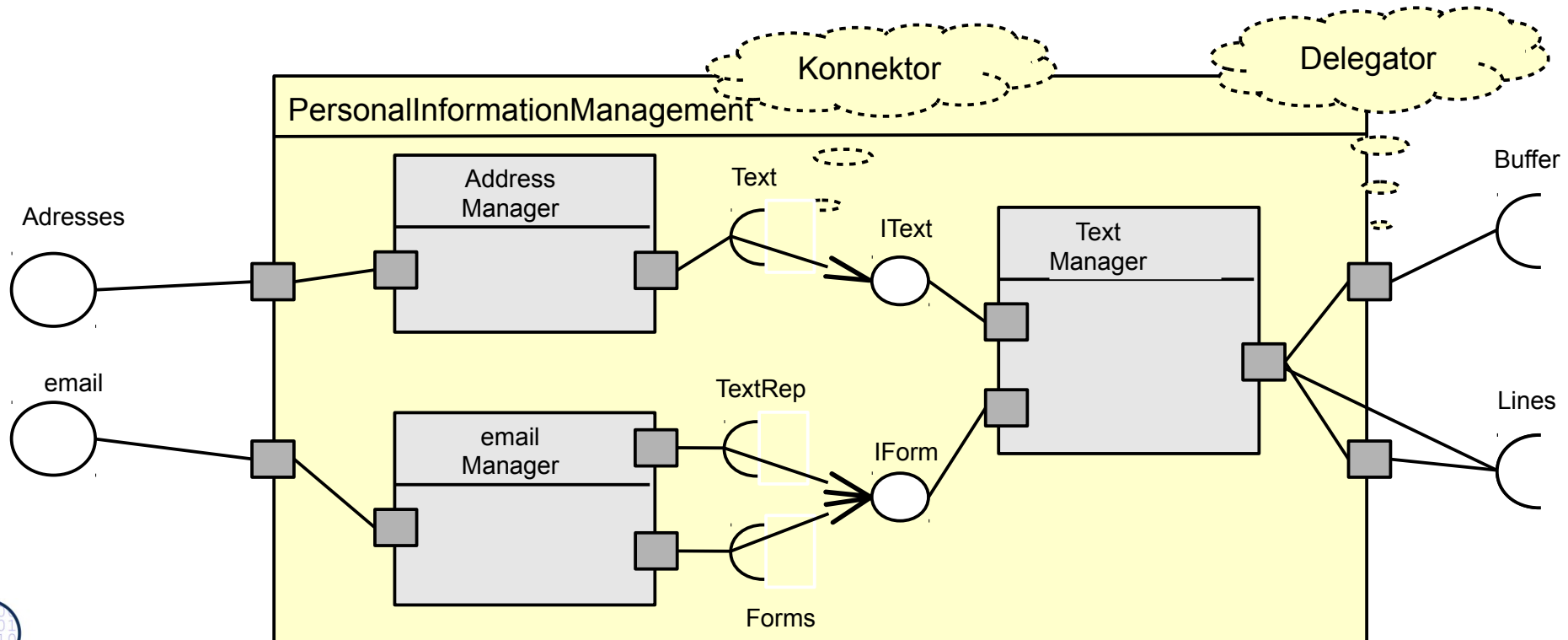
- ▶ **Anschlüsse (ports)** sind spezielle Compartments, bestehend aus verkapselten Port-Klassen mit Port-Schnittstellen
  - **provided**: normale, *angebotene* Schnittstelle
  - **required**: benutzte, *benötigte* Schnittstelle
- Ports können statt als Compartments auch als aufliegende Portklassen notiert werden



# Schachtelung von Klassen zu UML-Komponenten

12

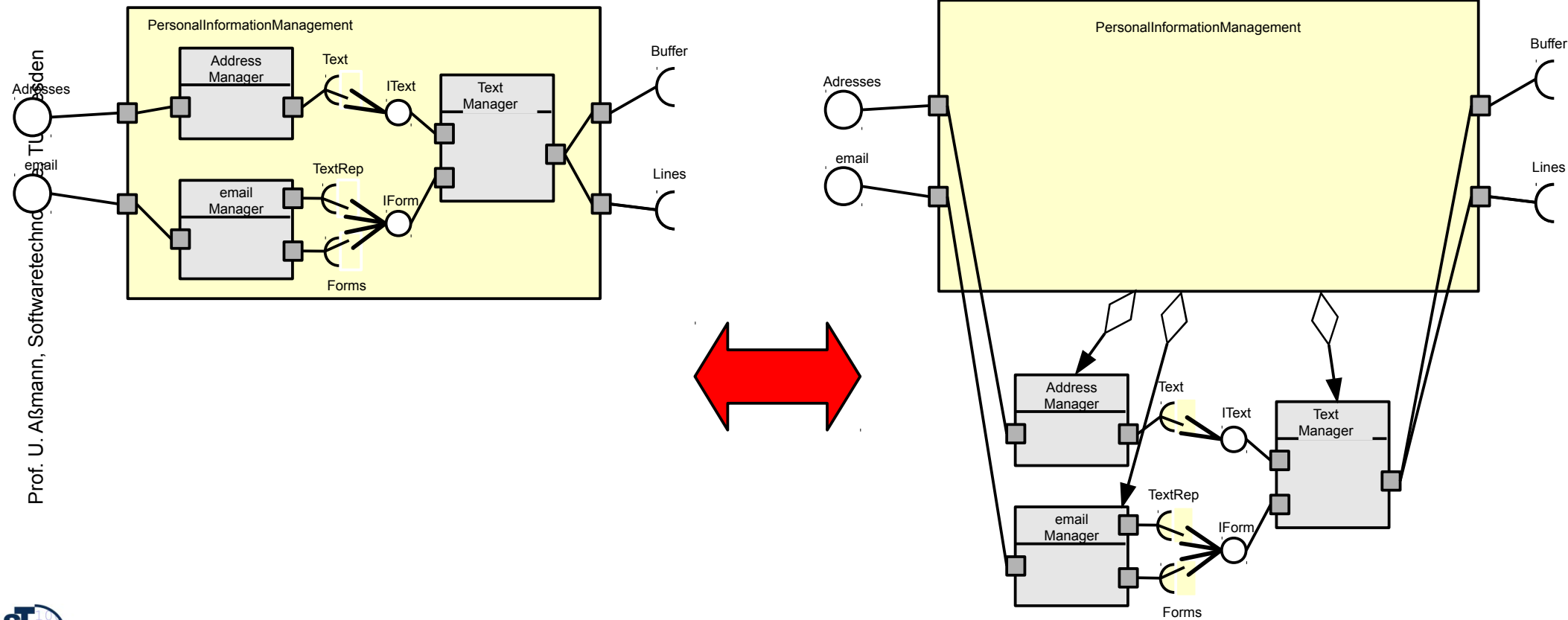
- ▶ Eine **UML-Komponente (Komponentenklasse)** ist eine Klasse mit angebotenen und benötigten Schnittstellen, mit Ports (Portklassen) und inneren Klassen
  - UML-Komponenten bilden hierarchische Klassen für hierarchische Objekte
  - Ports werden mit *Verbindern (Links, einfachen Konnektoren)* verbunden
  - *Delegator*: Link von äußerem zu innerem Port
  - Achtung: jeder Port kann eine Schnittstelle oder einer Klasse entsprechen
- Implementierung mit **Facade Pattern**: Komponente spielt eine Facade für die Unterkomponenten

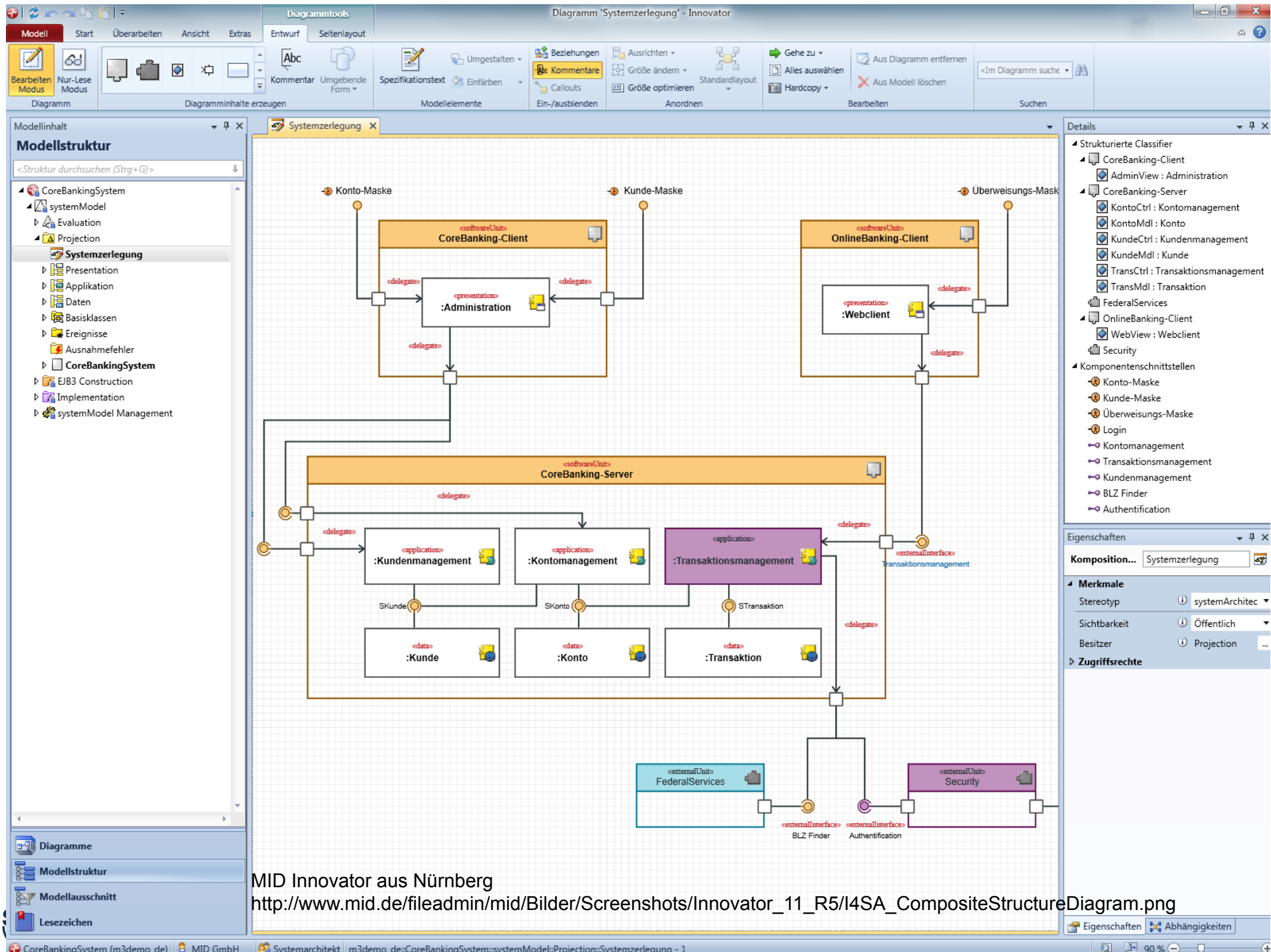


# Schachtelung bedeutet Aggregation

13

- ▶ *UML-Komponenten* sind hierarchische Klassen, die ihre Unterkomponenten aggregieren und ihre Sichtbarkeit begrenzen
  - Eine Komponente ist gleichzeitig ein *Paket* und eine *Facade* für alle Unterkomponenten
  - I.d.R. wird eine UML-Komponente mit einem Facade oder mehreren Bridge Pattern realisiert





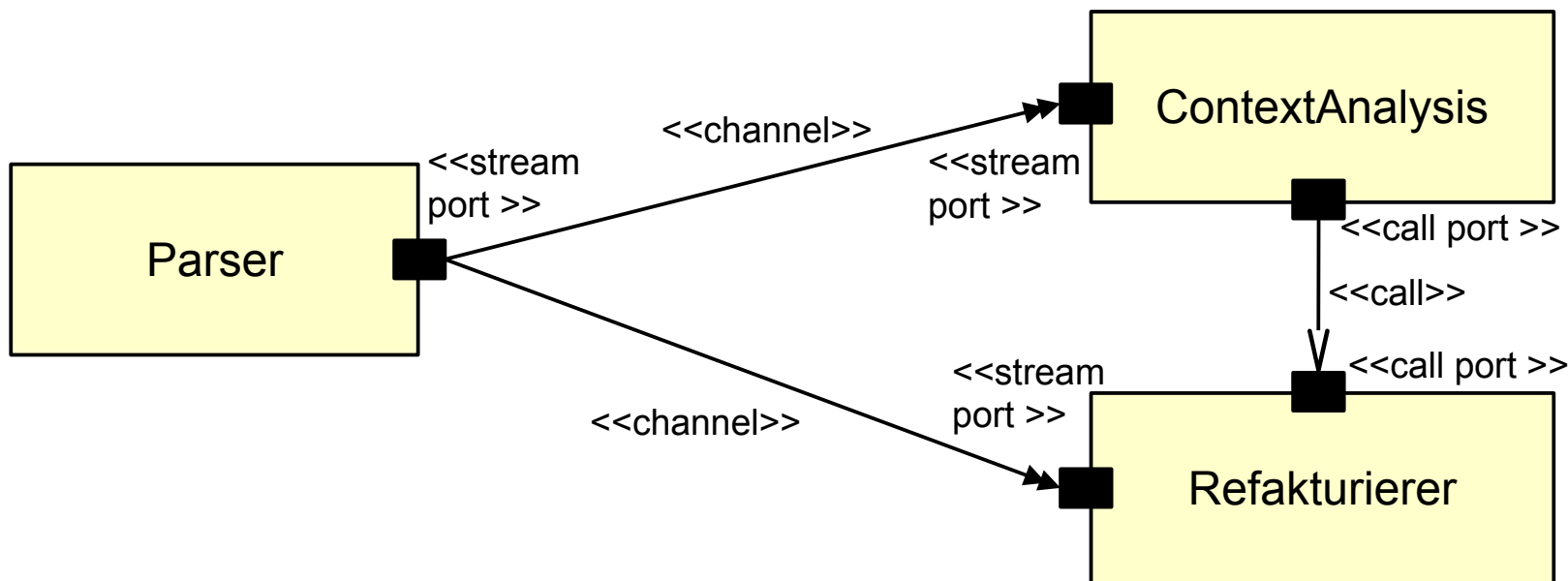
MID Innovator aus Nürnberg

[http://www.mid.de/fileadmin/mid/Bilder/Screenshots/Innovator\\_11\\_R5/I4SA\\_CompositeStructureDiagram.png](http://www.mid.de/fileadmin/mid/Bilder/Screenshots/Innovator_11_R5/I4SA_CompositeStructureDiagram.png)

# Strom-Anschlüsse von Komponentenklassen und Kanal-Konnektoren

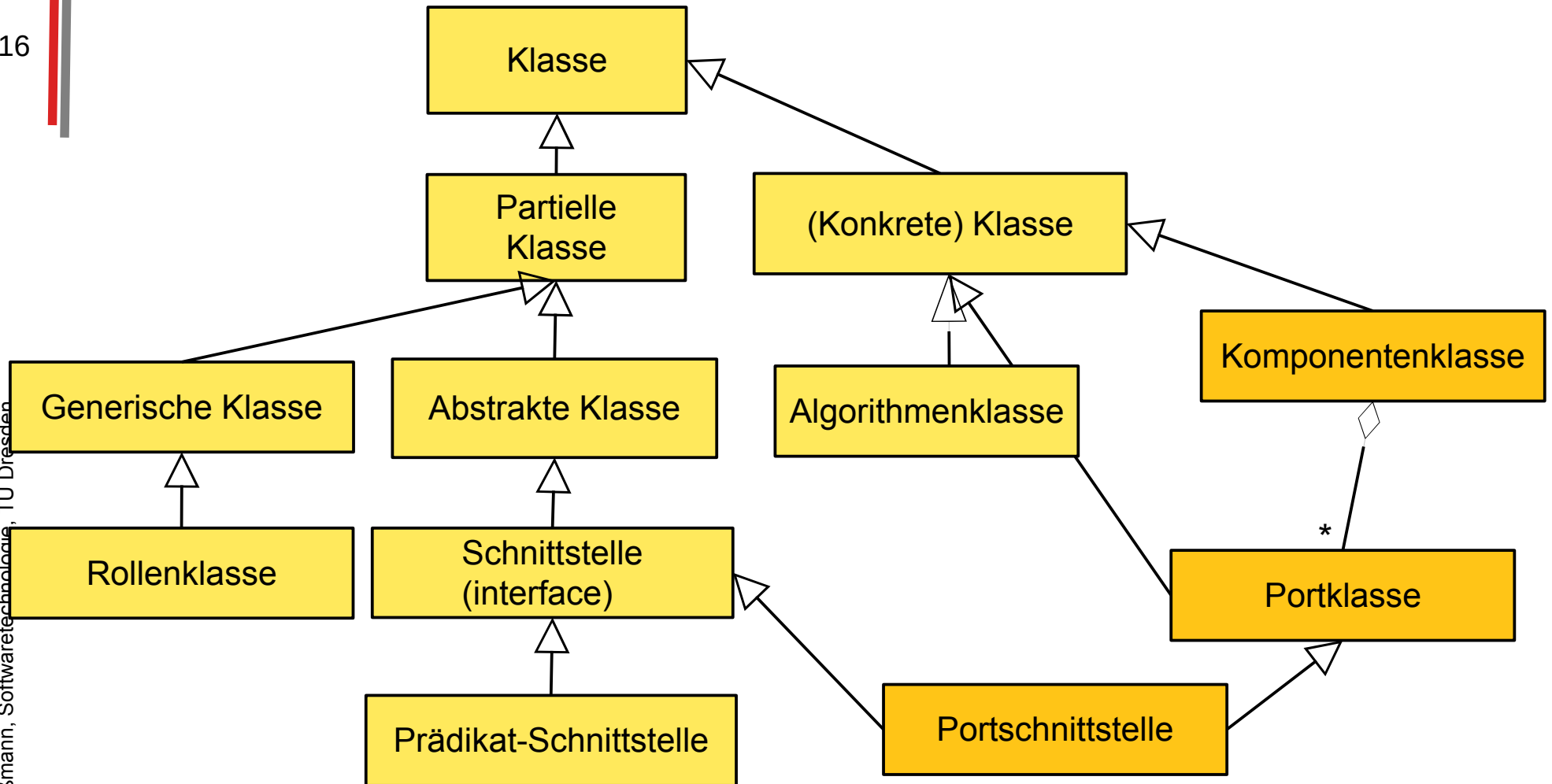
15

- ▶ Ein **Aufrufsanschluß (call port)** ist eine Portschnittstelle, die angebotene oder benötigte Operationen enthält, über die es synchron aufgerufen wird oder synchron aufruft
- ▶ Ein **Stromanschluß (stream port)** ist eine Portschnittstelle, die einen Ein- oder Ausgabestrom enthält, über den Daten ein und aus fließen (Iterator-Muster)
  - Ein Stromanschluß läßt auf ein aktives Objekt (Prozess) schließen, das den Strom bedient.
  - Daten können einfach oder strukturiert sein (große Objekte, Werte, Formulare, Webseiten)
  - Datentypen, die über den Stromanschluß fließen, stammen aus dem Domänenmodell
- Strom-Ports werden durch **Strom-Kanäle (channels, pipes)** zu Strömen verbunden
  - Es entstehen Pipe-und-Filter-Architekturen (Datenflussarchitekturen)
- ▶ Entwurfs- und Implementierungsmodell: Portschnittstellenklasse wird mit Muster Iterator (Stream, Sink) realisiert; Kanal mit Muster Channel



# Q2: Begriffshierarchie von Klassen (Erweiterung)

16





# 32.2 Kontextmodell

17



# Kontextmodell (Schnittstellenmodell)

18

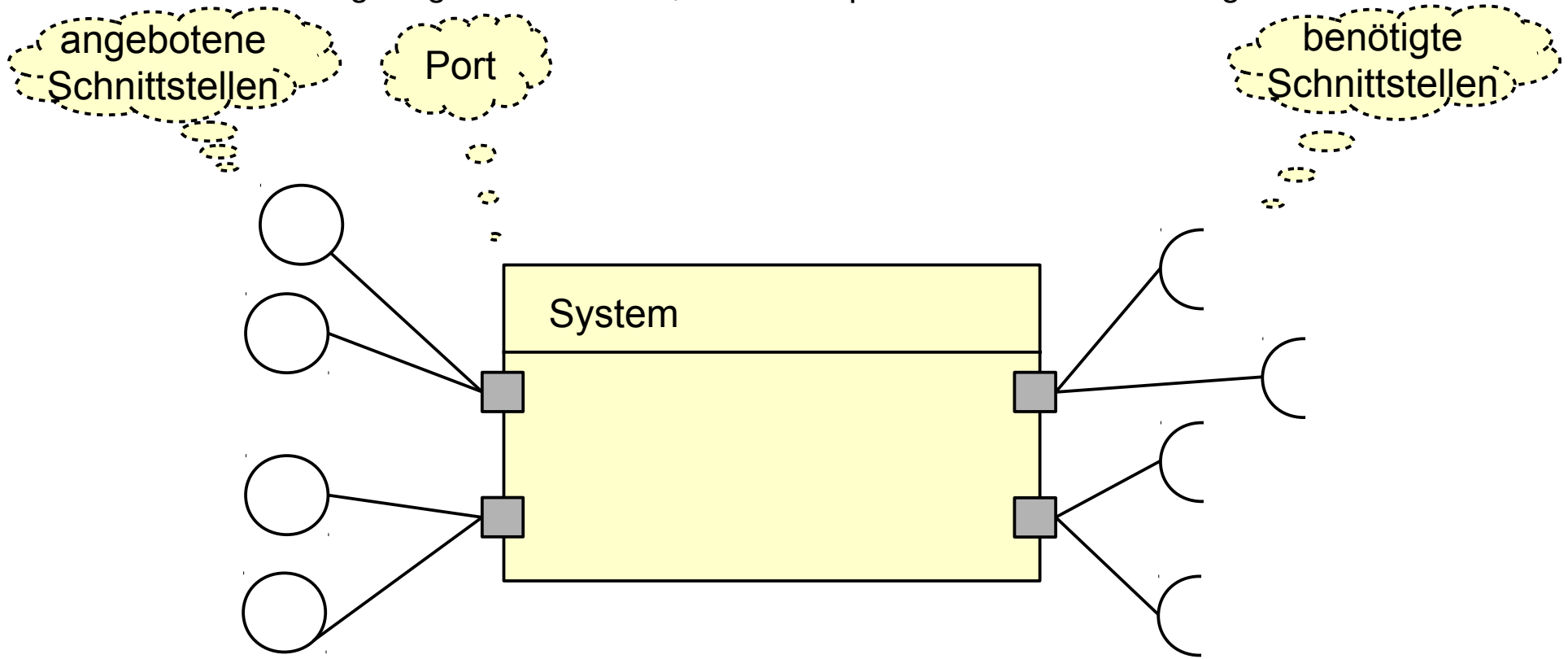
- ▶ Das **Kontextmodell (Schnittstellenmodell)** enthält die äusseren Schnittstellen des Systems (Portschnittstellen)
  - Funktionen für alle Anwendungsfälle (oberste Schicht der Anwendungslogik)
  - Ein- und Ausgabekanäle (ports, streams)
  - Benutzerschnittstelle (Eingabeformulare, GUI mit Masken und Abfragen)
- ▶ Sowie Daten, die in und aus dem System fließen (Typen aus dem Domänenmodell),
  - Ereignisse, Ausnahmen
- ▶ Es wird meist mit hierarchischen Komponentenklassen (UML-Komponenten) notiert
  - Mit angebotenen und benötigten Schnittstellen, mit ports
  - Hierarchisch durch die Ganz/Teile-Relation verfeinert

Das System wird oft als ein großes komplexes Objekt gesehen, dessen Schnittstellen vom Kontextmodell beschrieben werden.

# UML-Komponenten im Kontextmodell

19

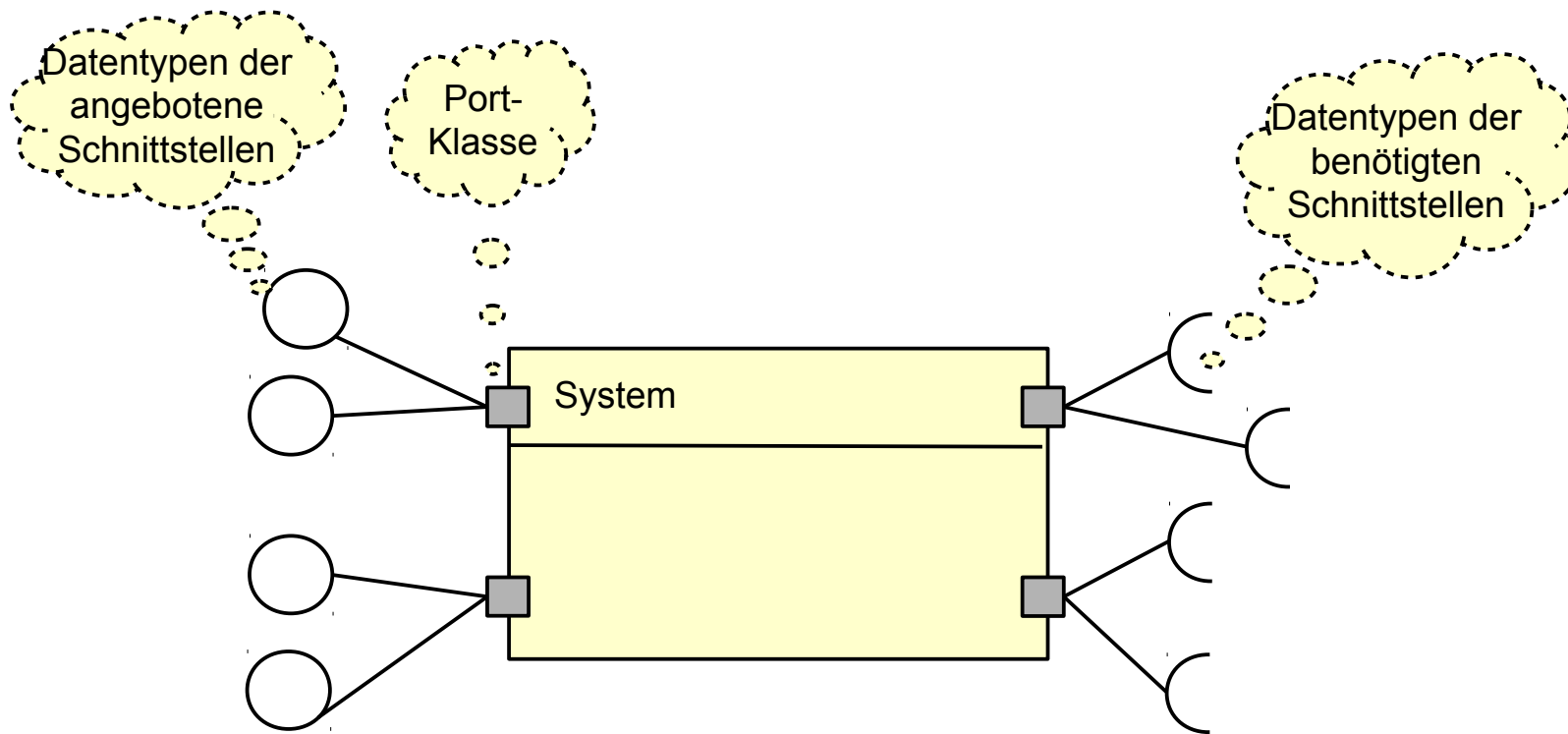
- ▶ UML-Komponenten werden im **Kontextmodell** des Softwaresystems eingesetzt (Störle: Montagediagramm)
  - Das System ist selbst eine Komponente
  - Die oberen (äußeren) Schichten stellen geschachtelte Komponenten dar
- ▶ Das Kontextmodells enthält *angebotene* und *benötigte* Schnittstellen:
  - Funktionale und strombasierte Schnittstellen (call und stream ports)
  - GUI-Bildschirme, Masken, Formulare
- Reine Klassen genügen nicht immer, denn sie spezifizieren keine benötigten Schnittstellen



# Datentypen des Kontextmodells stammen aus dem Domänenmodell

20

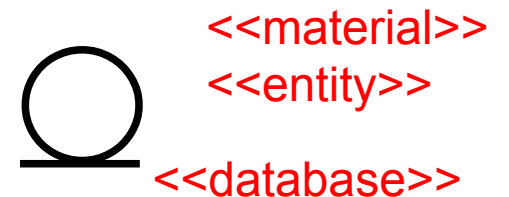
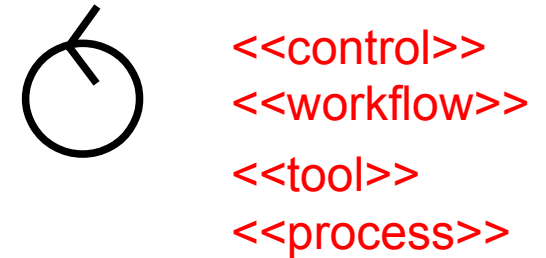
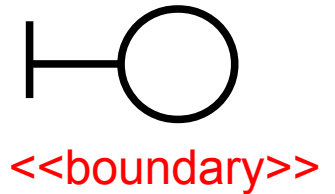
- ▶ Das Domänenmodell stellt die Typen für die technischen Objekte, Prozesse und Funktionalitäten des Kontextmodells zur Verfügung
  - Anschlüsse: Funktionale und strombasierte Schnittstellen (call und stream ports)
  - GUI-Bildschirme, Masken, Formulare
  - Port-Klassen und Schnittstellen sind Grenzklassen (boundary)



# Unterscheidung von Systemschnittstellenklassen von anderen Klassen mit dem BCD-Profil

21

- ▶ **Schnittstellen-Klassen** (Portschnittstellen, Portklassen, GUI-Klassen, Grenzklassen, boundary classes)
  - Teil einer Schnittstelle oder Benutzerschnittstelle
- ▶ **Steuerungsklassen** (control classes)
  - Aktive Klasse der Anwendungslogik, die die Ausführung eines Prozesses steuert
  - Mit oder ohne Zustand
- ▶ **Material**: Passive Klasse, beschreibt Daten
  - Entitätsklassen (Entity): Beschreibt komposite, persistente Datenobjekte der Domäne
  - Datenklasse (Database): Adapterklasse für eine Entity in der Datenbank
    - Oft sind Entity and Database vereinigt
  - MaterialContainer: Container für Material
- ▶ BCD-Architektur: 3-Schichten-Architektur (3-tier architecture)
- ▶ BCED-Architektur: 4-Schichten-Architektur (4-tier architecture)

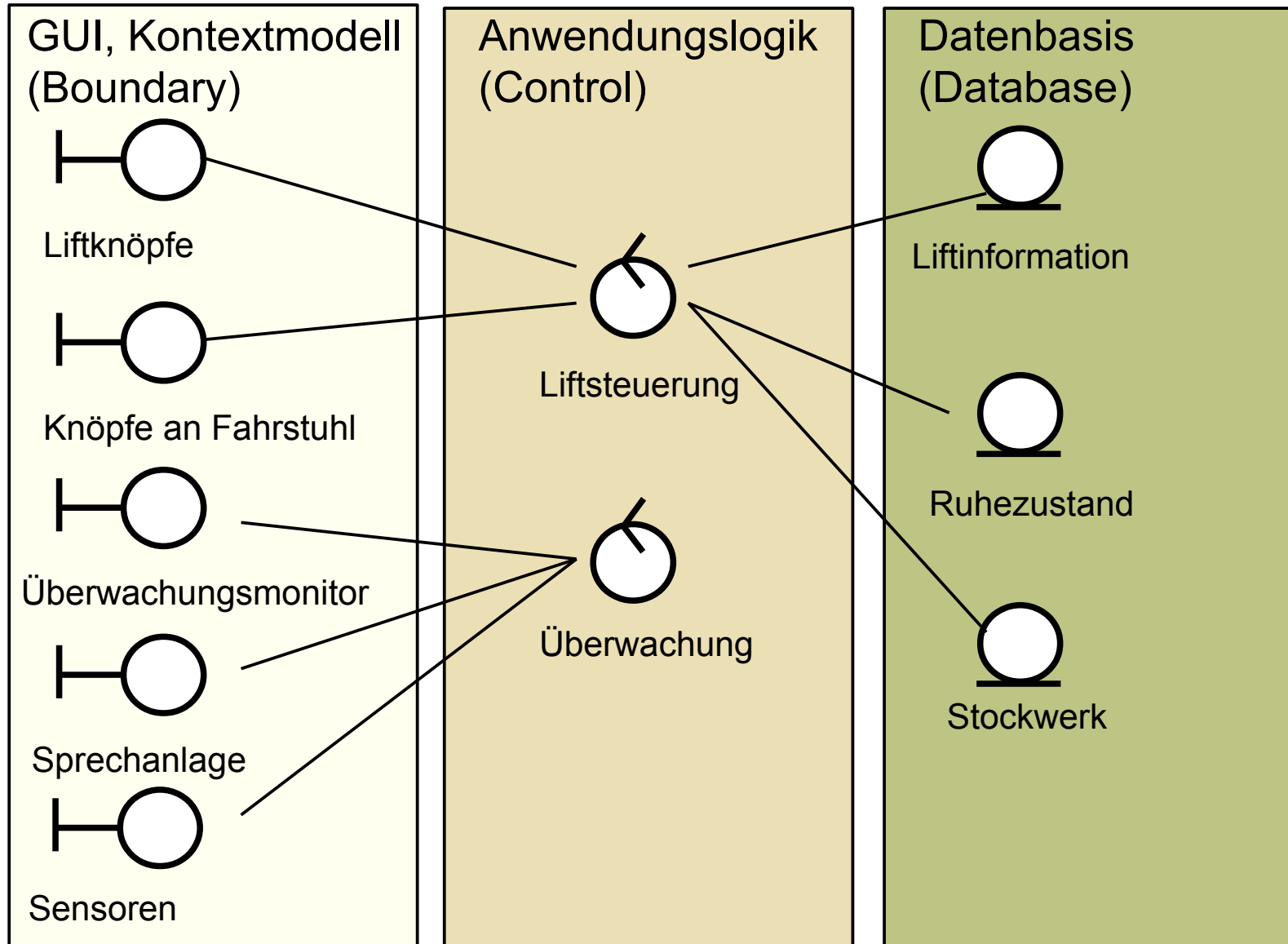


<<container>>

# Bsp: Analyse-Klassenmodell Liftsteuerung im BCD-Stil

22

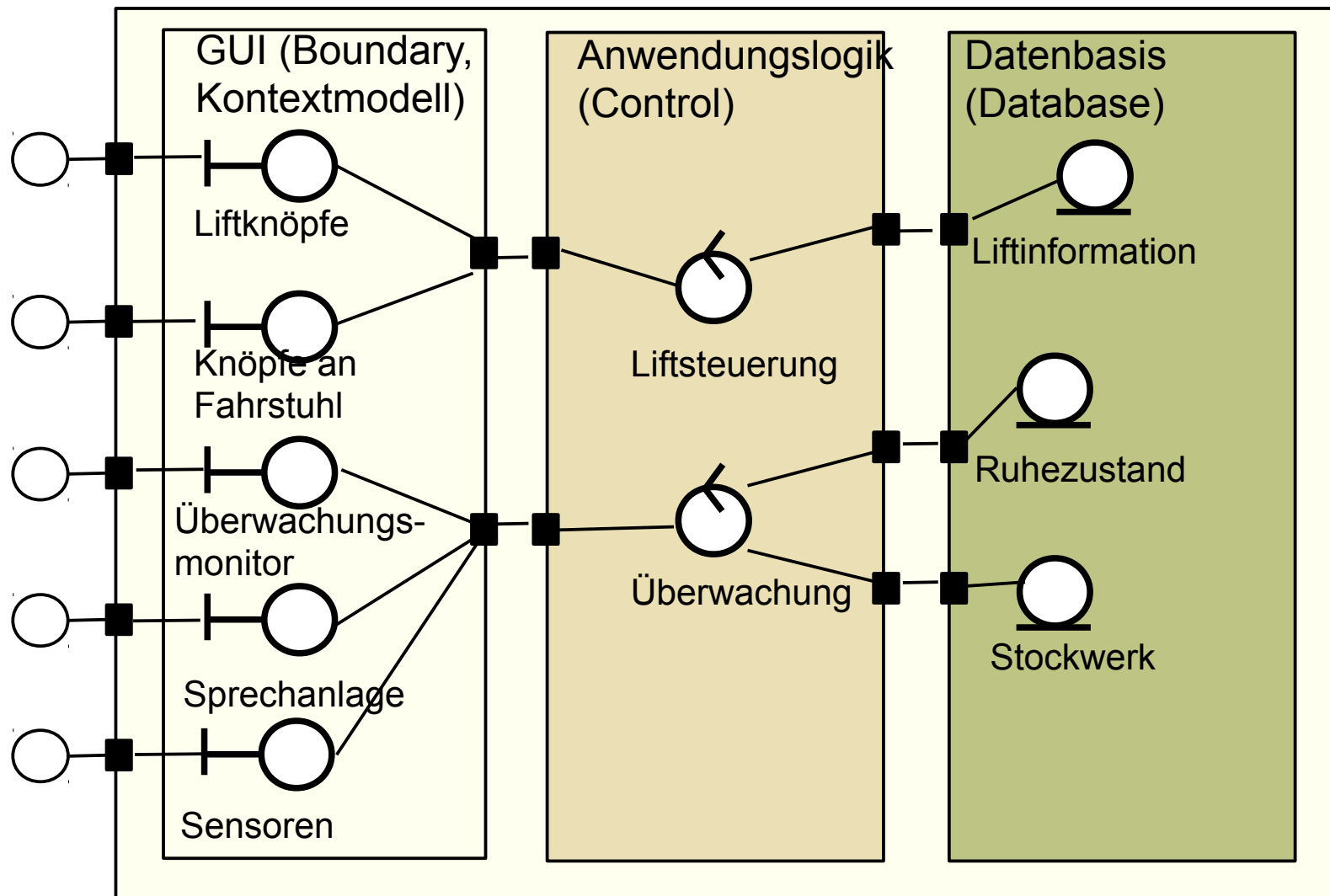
- ▶ als 3-Schichten-Architektur (3-tier architecture) von **Klassen**. Schichten sind nicht voll gekapselt



# Bsp: Analyse-Modell Liftsteuerung mit Kontextmodell als UML-Komponentenklasse

23

- ▶ Als 3-Schichtenarchitektur von **Komponenten** → Schichten sind voll gekapselt und besser vorbereitet auf Austausch von Komponenten und Verteilung





# 32.3 Top-Level-Architektur

---

---

24

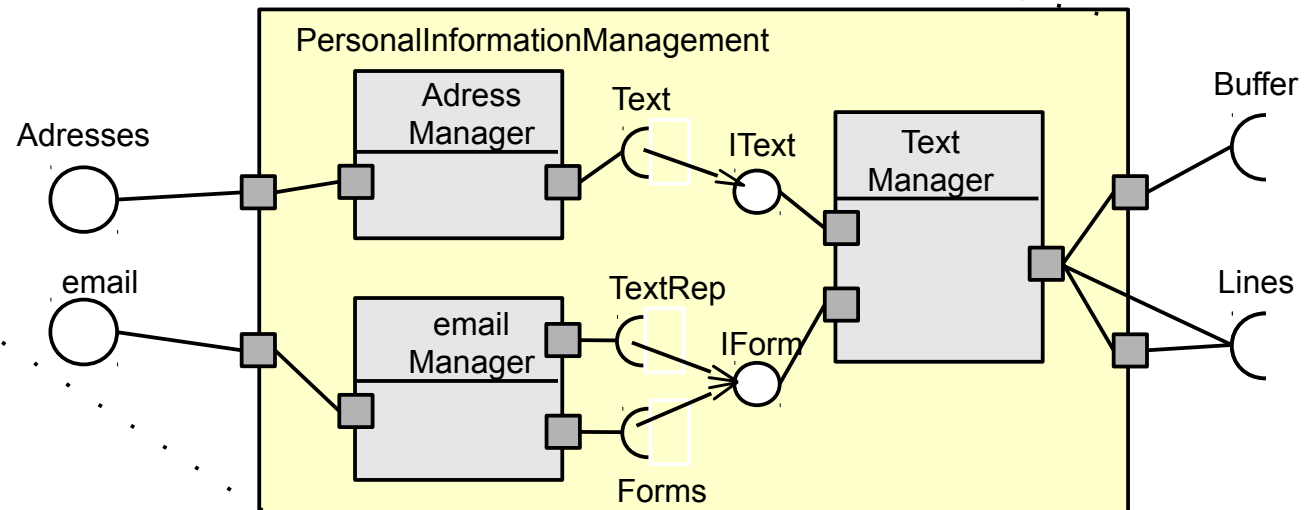
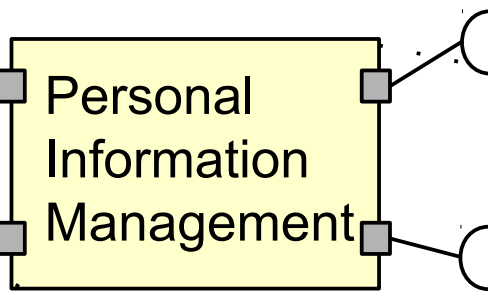
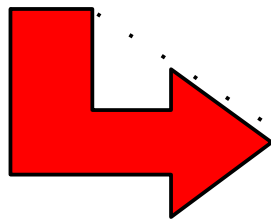
Das System ist ein großes komplexes Objekt, dessen Schnittstellen vom Kontextmodell beschrieben werden und das hierarchisch organisiert ist (Top-Level-Architektur).



# Top-Level-Architektur

25

- ▶ Wird das System als Großobjekt mit angebotenen und benötigten Schnittstellen betrachtet, kann man es hierarchisch mit part-of verfeinern
  - Das Kontextmodell wird als UML-Komponente, d.h., hierarchische Klasse, spezifiziert
  - Die **Top-Level-Architektur** entsteht durch die erste Verfeinerung des Kontextmodells
  - Die inneren Komponentenklassen werden sichtbar



# Warum wird die Top-Level-Architektur in der Anforderungsanalyse ermittelt?

26

- ▶ Man sieht die Antwort an TollCollect: *Die Top-Level-Architektur der Anwendung gehört mit zur Anforderungsspezifikation, weil*
  - die Komponenten der ersten Schicht dem Benutzer sichtbar sind (was sind die Top-Level-Komponenten der TollCollect-Software?)
  - sie oft Aufgaben direkt zugeordnet werden können, die der Benutzer kennt (was sind die Aufgaben und Top-Level-Komponenten einer Groupware?)
  - sie zur Kostenplanung und Abrechnung für das Projekt verwendet werden (Produktstrukturplan, siehe Vorlesung Softwaremanagement)
  - sie dem Manager eine Einteilung von Projektmitarbeitern vereinfachen
- ▶ Wenn sie nicht zur Anforderungsspezifikation hinzukommt, ist sie als erstes Dokument im Entwurf auszuarbeiten
  - um die Planung zu vereinfachen

# 32.3 Asynchrone Systemmerkmale im Kontextmodell

27

Ereignisse und Ausnahmen (Exceptions)

# Schritte der strukturellen, datengetriebenen Analyse

28

- ▶ gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur

strukturelle OOA

Klassen identifizieren

Klassen nach Profilen klassifizieren

Merkmale identifizieren

Attribute identifizieren

Operationen identifizieren

Ereignisse und Ausnahmen identifizieren

Ströme und Kanäle identifizieren

Klassenbeziehungen identifizieren

Assoziationen ident.

Assoziationsklassen id.

Vererbungen ident.

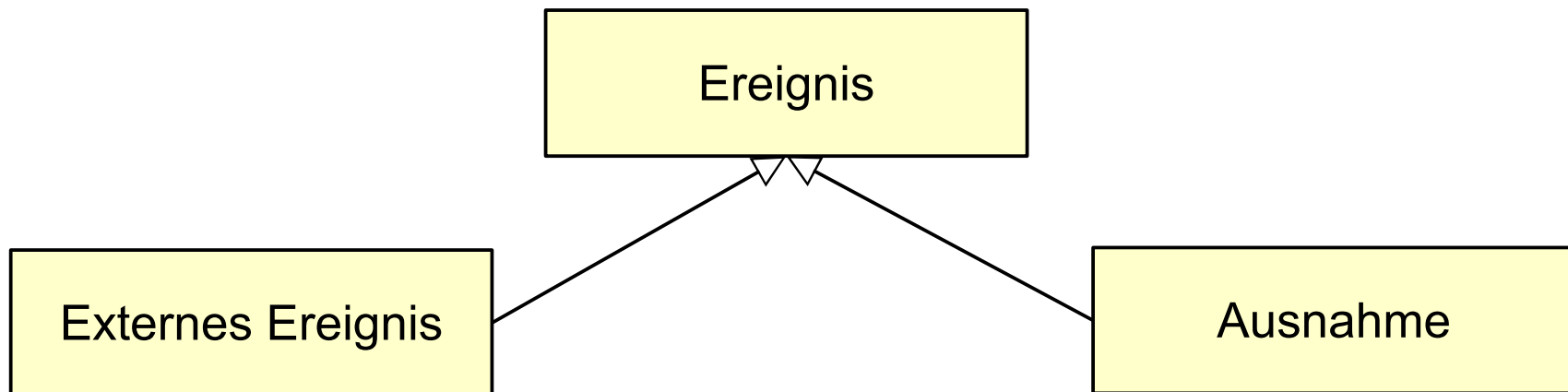
Komplexe Klassen ident.

Kanäle ident.

# Asynchrones Verhalten des Systems

29

- ▶ Im Kontextmodell muß zusätzlich das *asynchrone Verhalten des Systems* spezifiziert werden
- ▶ **Externe Ereignisse** – und wie soll das System darauf reagieren?
  - Benutzererzeugte Ereignisse
  - Plattform-erzeugte Ereignisse (Platte voll, Power out, ...)
  - Sensorwerte
  - Externe Ereignisse können durch Ströme (stream ports) in das System fließen

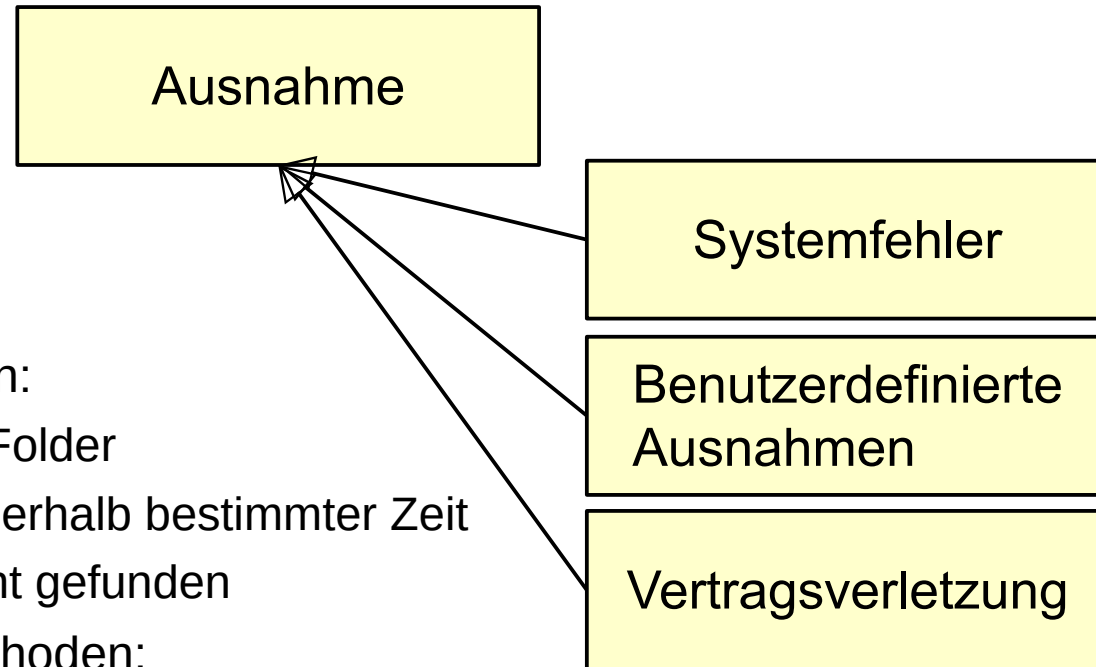


# Ausnahmen im Kontextmodell

30

- ▶ Eine **Ausnahme (exception)** ist ein internes Ereignis, das vom System selbst ausgelöst wird

- Systemfehler:
  - Division durch 0
  - Zugriff durch null-Zeiger
  - Platte voll
- Benutzerdefinierte Ausnahmen:
  - Zu viele Dateien in einem Folder
  - Benutzer reagiert nicht innerhalb bestimmter Zeit
  - Material in Datenbank nicht gefunden
- Vertragsverletzungen von Methoden:
  - Zeiger null
  - integer-Wert zu gross

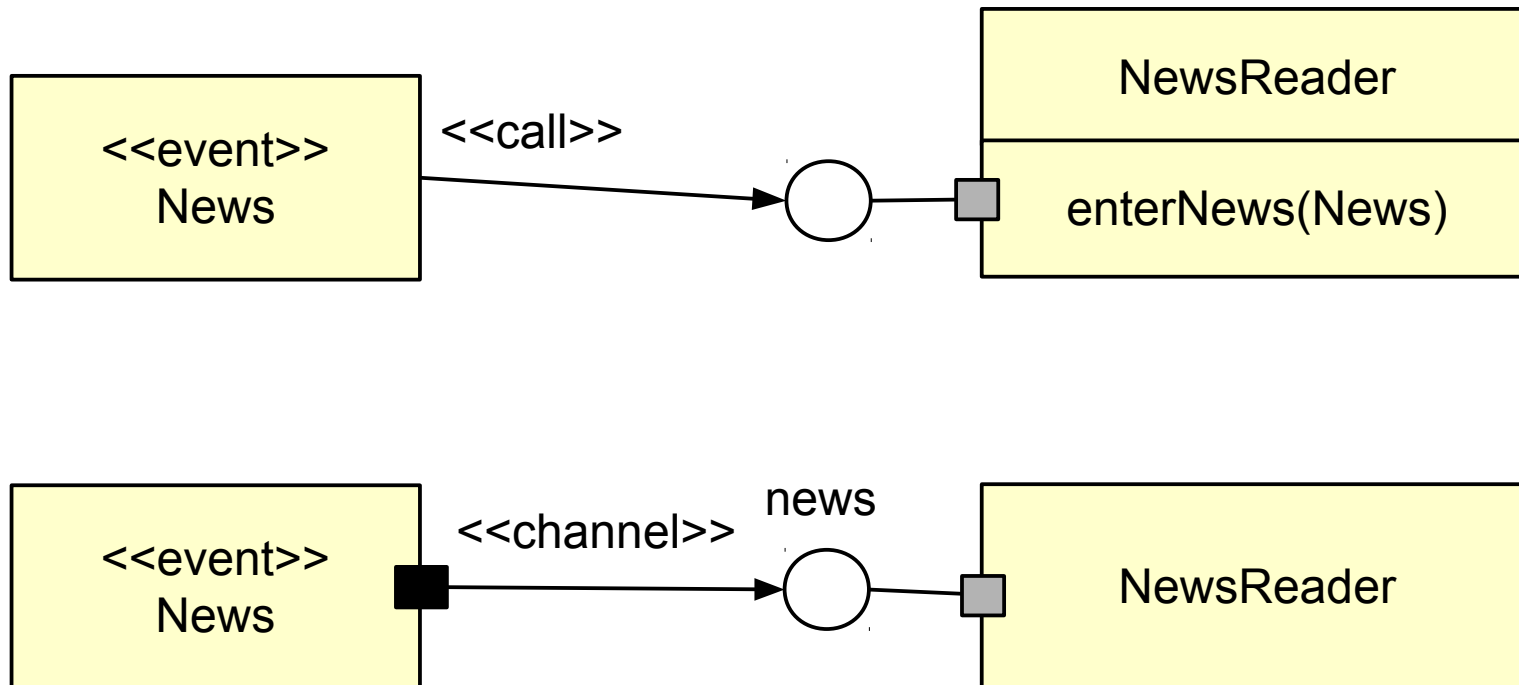


- ▶ Ausnahmen des Systems und die Reaktion des Kontextes auf diese gehören ins Kontextmodell!
- ▶ Implementierung: Ausnahmen sind als Konzept in Java vorhanden (exception)

# Externe Ereignisse gelangen ins System über das Kontextmodell

31

- ▶ Ein Ereignis ist also ein asynchron auftretendes Geschehen, das als Ereignisobjekt repräsentiert und schliesslich in eine Klasse, Komponente, oder das System hineingereicht wird
  - als Parameter einer Methode (call port)
  - als Objekt in einem *Stromanschluss* über einen *Kanal* (Entwurfsmuster Channel)



# Unterschied Domänen- vs. Kontextmodell

32

## **Domänen-Modell .-** ***Das Abbild der realen Welt***

Notation: UML

Objekte: Fachgegenstände

Klassen: Fachbegriffe

Attribute ohne Typen

Operationen: ohne Typen,  
Parameter und Rückgabewerte

Assoziationen: partiell, bidirektional  
i. Allg. ohne Datentypen

Aggregationen, Kompositionen  
Leserichtung, partielle Multiplizitäten

Vererbung: Begriffsstruktur

Annahme perfekter Technologie

Funktionale Essenz

Grobe Strukturskizze

## **Kontext-Modell - Teile der** ***Technik, die der Kunde sehen muss***

Notation: UML

Objekte: Softwareeinheiten

Klassen: Komponenten, Grenzklassen,  
Ports, Interface,  
Stereotypen aus Profilen

Attribute: Klassenattribute,  
Ports, Ströme

Unidirektionale Assoziationen mit  
voller Multiplizität, Navigation

Vererbung: Programmableitung

Asynchrones Verhalten: Ereignisse, Ausnahmen

Genauere Strukturdefinition in der  
Top-Level-Architektur: Adapter, Konnektoren



# The End - Anhang mit optionalem Material

33

- ▶ Einige Folien sind eine überarbeitete Version von Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. Used by permission.

Fragen:

- ▶ Was unterscheidet Komponenten und Klassen?
- ▶ Warum modelliert man auf den äußeren Schichten des Systems mit Komponenten statt mit Klassen?
- ▶ Welche Schnittstellen besitzt ein System neben einer graphischen Benutzerschnittstelle noch?
- ▶ Welches “required interface” hat eine Anwendung gegenüber dem Betriebssystem?

# 32.A.1 Adapter zum Brücken von Schnittstellen in der Top-Level-Architektur

34

# Aufgabe des Kontextmodells: Verbindung nach außen

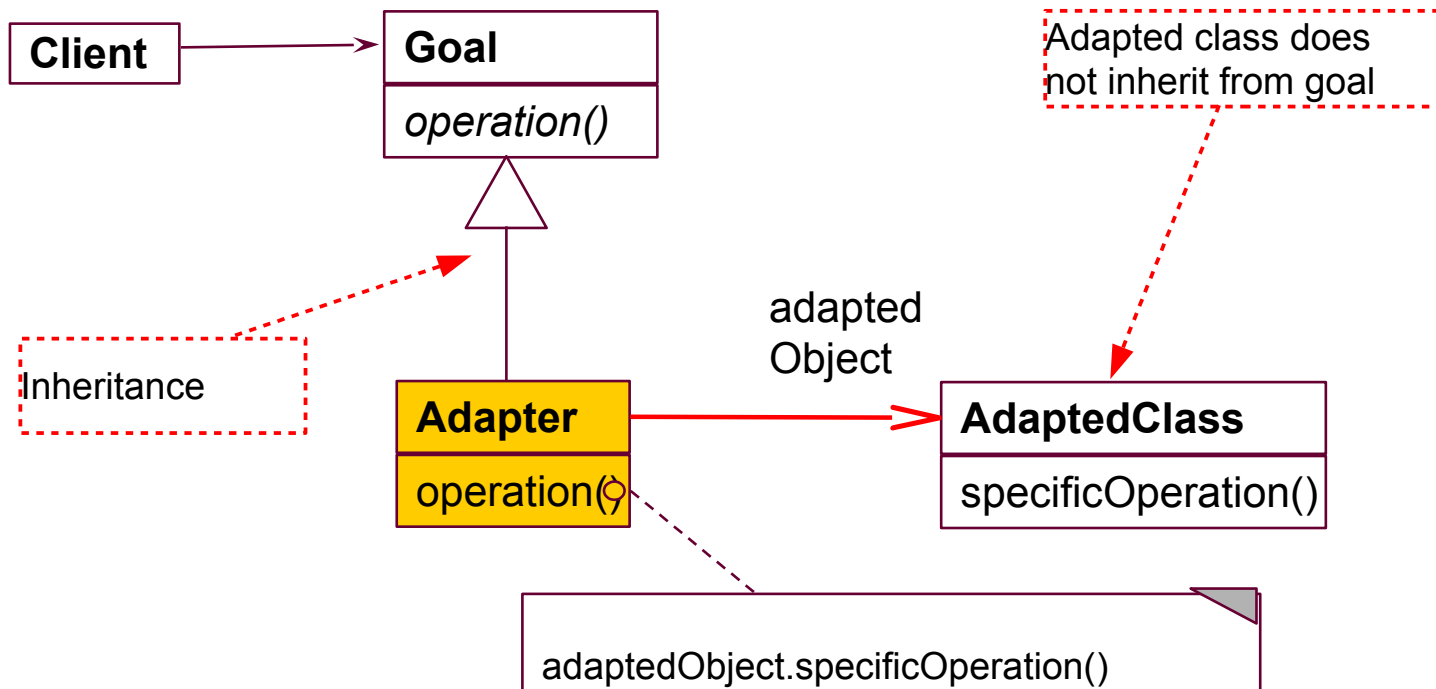
35

- ▶ Eine wesentliche Aufgabe des Kontextmodells ist die Verbindung des Systems mit dem Kontext, d.h. der restlichen Welt
- ▶ Die Top-Level-Architektur detailliert die Verantwortlichkeiten für die Verbindungen
- ▶ Jede neue Verbindung erzeugt eine neue Rolle des Systems (neuer Kontext)
- ▶ Da die Schnittstellen der externen Systeme mit denen der internen Komponenten oft nicht zusammenpassen (*architectural mismatch*), werden Adapter konzipiert, die die Rollen implementieren
  - Dazu kann man das Entwurfsmuster *Adapter* verwenden

# Entwurfsmuster Adapter (Objektadapter) (Wdh.)

36

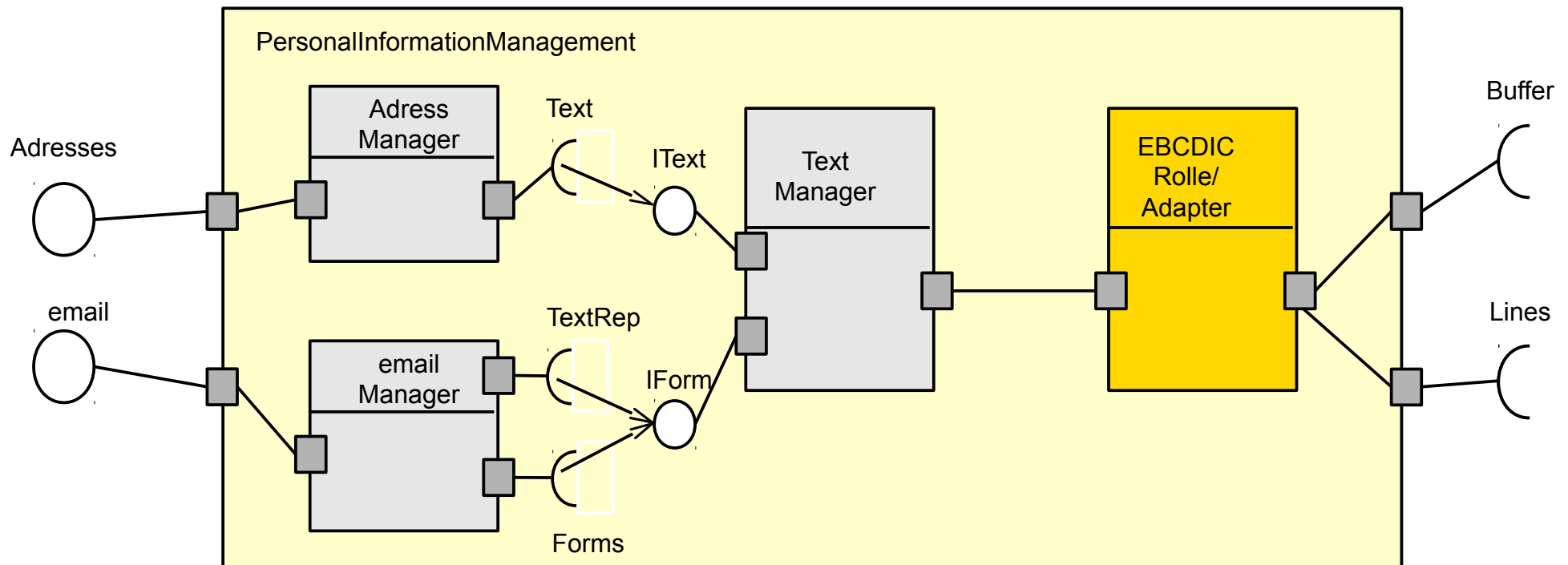
- ▶ Ein Adapter (Objektadapter) ist ein Objekt, das
  - eine Schnittstelle auf eine andere abbildet
  - ein Protokoll auf ein anderes abbildet
  - Datenformate auf einander abbildet
  - Delegation verwendet



# Top-Level-Architektur mit Adaptern

37

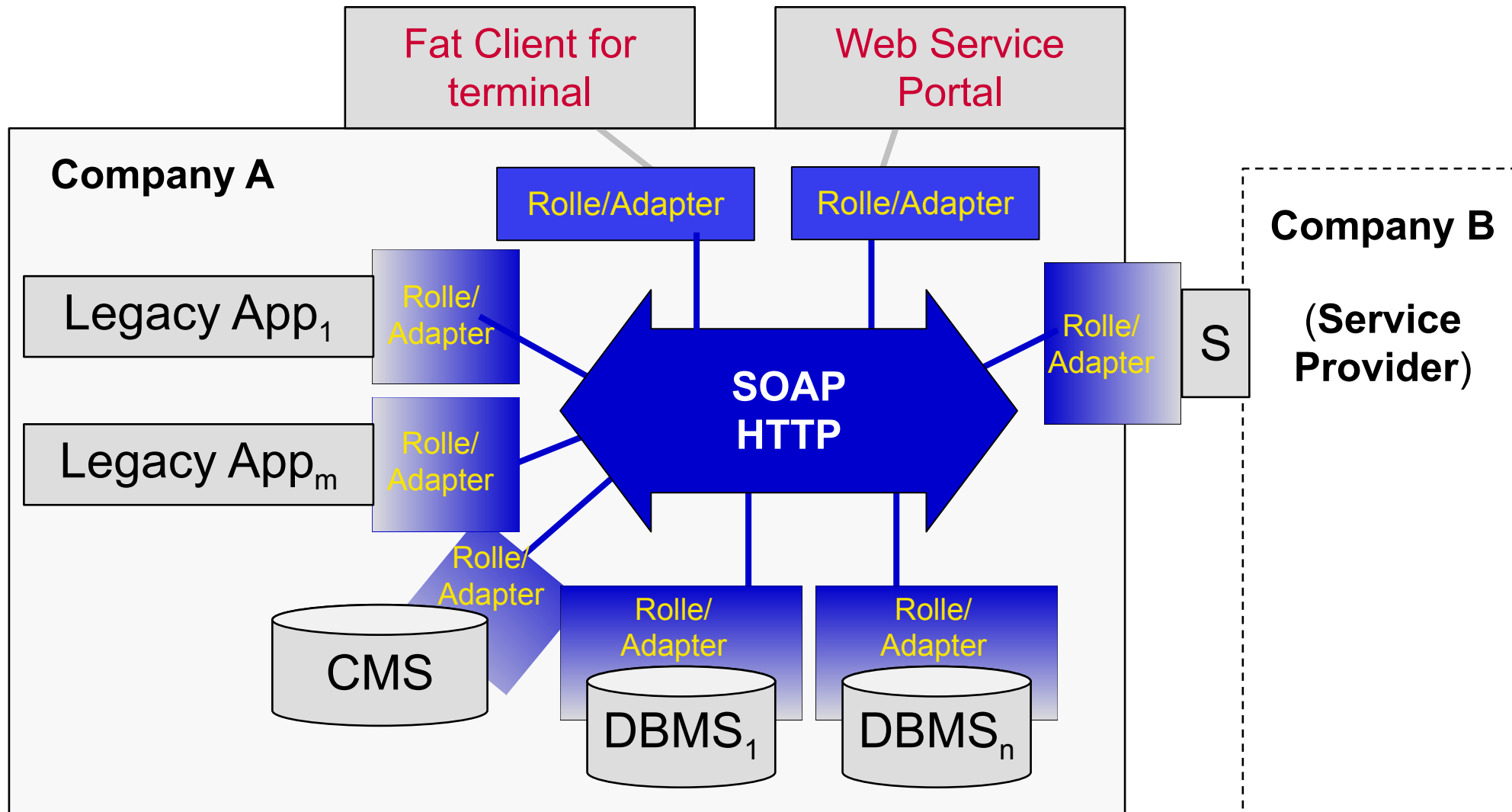
- ▶ Die Rollen bzw. Adapter werden nun zwischen den Top-Level-Komponenten und Komponenten der Umgebung eingesetzt
  - z.B. als Daten-Adapter für die unterschiedlichen Character-Codes der unterschiedlichen Maschinen



# Beispiel: Web Services mit komplexen Konnektoren SOAP/HTTP

38

- ▶ **Enterprise Application Integration (EAI)** steckt Systeme aus Komponenten mit Hilfe von Rollen bzw. Adaptern zusammen



# Profil "Aktive und Passive Klassen"

39

- ▶ Zum Kontextmodell gehört die Unterscheidung von *aktiven Klassen (Prozessen)* und *passiven Klassen*

- Im einfachsten Fall existiert *eine* aktive Klasse

- ▶ Aktive Objekte

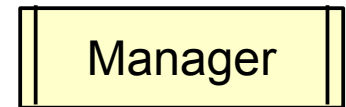
- Aktoren

- Prozesse `<<actor>>`

- Workflow (interaktiver Prozess)

- Werkzeuge (Tools) `<<tool>>`

`<<active class>>`



`<<process>>`

- ▶ Passive Objekte (Materials, entities, data objects)

- Aktive Objekte arbeiten auf Materialien

`<<datatype>>`

`<<material>>`

`<<passive class>>`

`<<entity>>`

- ▶ Kanäle (channels, pipes): Beschreiben, wie aktive Objekte miteinander kommunizieren

- Über Kanäle fließen Daten, sie werden mit Senken geschrieben und mit Streams gelesen

`<<channel>>`

`<<call>>`

`<<stream>>`

# Beispiel: Bahrami-Profil für Domänenmodell

40

- ▶ Das Bahrami Profil wird hauptsächlich im Domänenmodell eingesetzt
  - Und damit als Typen für die Schnittstellen im Kontextmodell
- ▶ Konzept, Begriff (concept) <<concept>>
  - Etwas, worauf sich viele Leute eines Anwendungsbereiches geeinigt haben. Oft angeordnet in Taxonomien oder Ontologien
  - Benutzt im Domänenmodell
- ▶ Ereignisklasse (Event) <<event>>
  - Ereignis in der Zeit, extern zum Objekt
- ▶ Organisation <<organization>>
  - Verkörpert Wissen über eine Organisationseinheit
- ▶ Menschen (People) <<people>>
- ▶ Plätze (Places class) <<place>>



# Beispiel: Rumbaugh-Profil

41

- ▶ Domänenmodell:
  - Physical class (e.g., Boat) <<physical>>
  - Business class (e.g., Bill) <<business>>
- ▶ Anwendungslogik in Kontextmodell und Toplevel-Architektur:
  - Logical class (e.g., Timetable) <<logical>>
  - Application class (e.g., BillingTransaction) <<application>>
  - Behavioral class (e.g., Cancellation) <<behavior>>
- ▶ Plattform im Implementierungsmodell:
  - Computer class (e.g., Network) <<computer>>

# 32.A.2 Weitere Arten von Schnittstellen und Klassen

42

(zur Information)

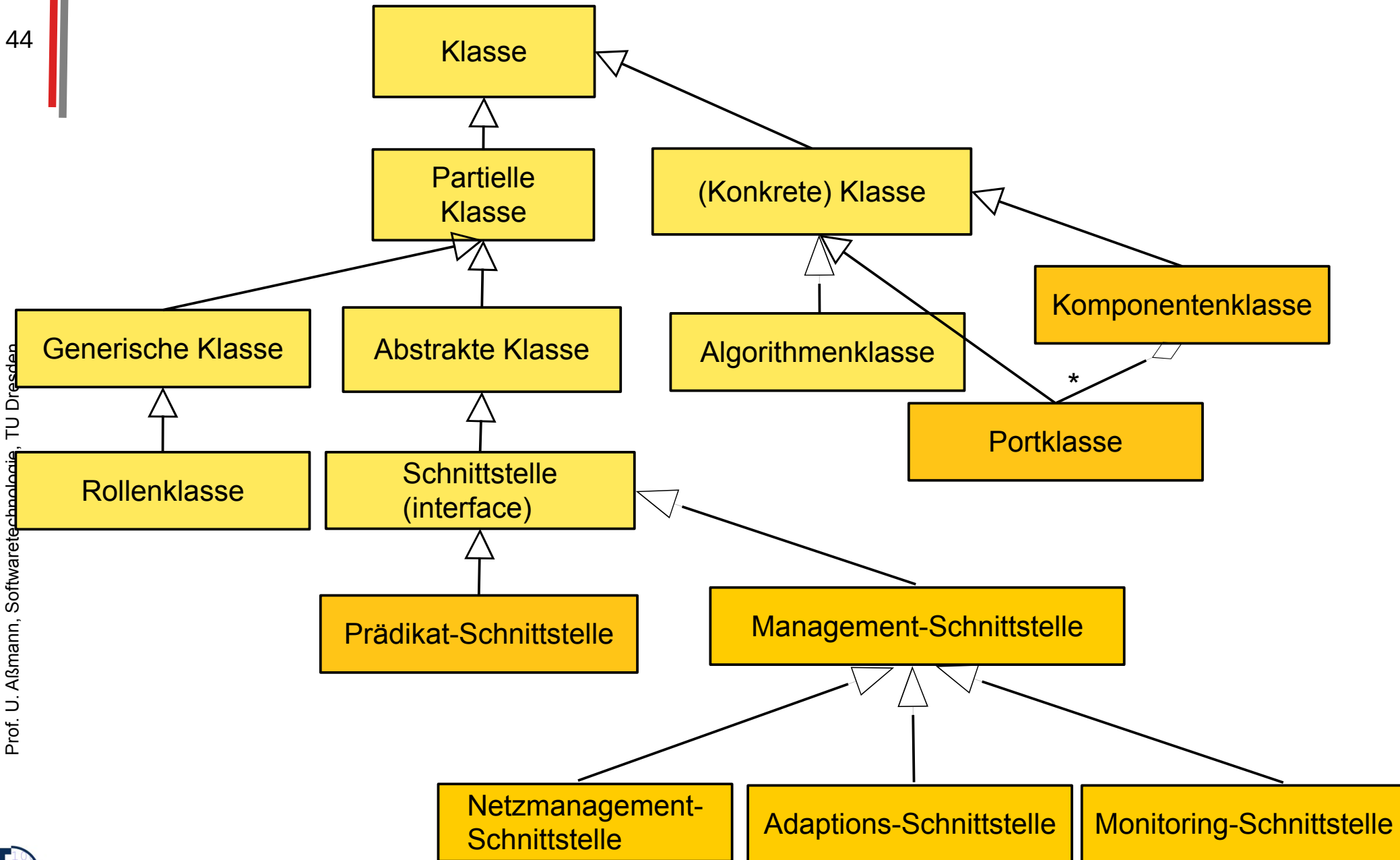
# Weitere Arten von Schnittstellen in komplexen Objekten

43

- ▶ **Funktionale Schnittstellen** enthalten Funktionen, die direkt auf den Zustand des Objekts zugreifen
  - **Prädikat-Schnittstellen**, die Prädikate auswerten
- ▶ **Managementschnittstellen** enthalten Funktionen, die das Objekt und seine Nachbarn verwalten:
  - **Netzmanagement-Funktionen** verändern das Netz
  - **Adaptions-Funktionen** verändern Parameter des Objekts, passen das Objektverhalten auf den Kontext an, optimieren das Objekt, verändern seinen Lebenszyklus
  - **Monitoring-Funktionen** messen bestimmte Parameter des Objekts

# Q2: Begriffshierarchie von Klassen (Erweiterung)

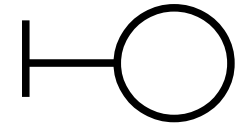
44



# Identifikation von Systemschnittstellenklassen für das Kontextmodell

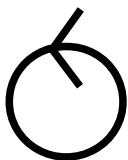
45

- ▶ **Schnittstellenklassen (boundary, Grenzklassen)** bestehen oft aus
  - Formularen, die dem Benutzer präsentiert werden
  - html-Seiten
  - Abfragen, Meldungen, Sichten auf Daten
  - Schnittstellenklassen zu anderen Systemen, inklusive Adaptern für andere Systeme
  - Strom-Anschlüsse für Datenströme, die ein und aus fließen
- ▶ Oft können Grenzklassen als Portklassen der System-Komponentenklasse dargestellt werden
- ▶ Bsp: Terminanwendung:
  - Tabelle der gebuchten Besprechungen
  - Formular, um eine neue Buchung eines Raumes einzutragen
  - Tabelle der Besprechungen pro Mitarbeiter
  - Report über die Auslastung der Besprechungsräume



# Identifikation von Steuerungs- und Entitätsklassen

46

- ▶ **Steuerungsklassen (control)** enthalten *Anwendungslogik*, d.h. die anwendungsspezifische Funktionalität 
  - Steuerungsklassen treten oft in der Top-Level-Architektur auf, wo sie aus Schnittstellenklassen im Kontextmodell entwickelt werden
  - Im Entwurf werden die Steuerungsklassen der Top-Level-Architektur weiter verfeinert
  
- ▶ **Entitätsklassen (data, Datenklassen, Materialien)** werden aus den passiven Klassen eines Domänenmodells bestimmt
  - Entitätsobjekte können physikalisch aus sehr vielen einzelnen Objekten bestehen (physikalische Splitterung)
  - --> Aggregations- und Kompositionsoperation in UML

