

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

Wie man ein System auf grobkörniger Ebene strukturiert

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 19-0.1, 15.06.19

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



Literatur

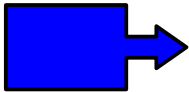
2

Softwaretechnologie (ST)

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

Überblick Teil III: Objektorientierte Analyse (OOA)

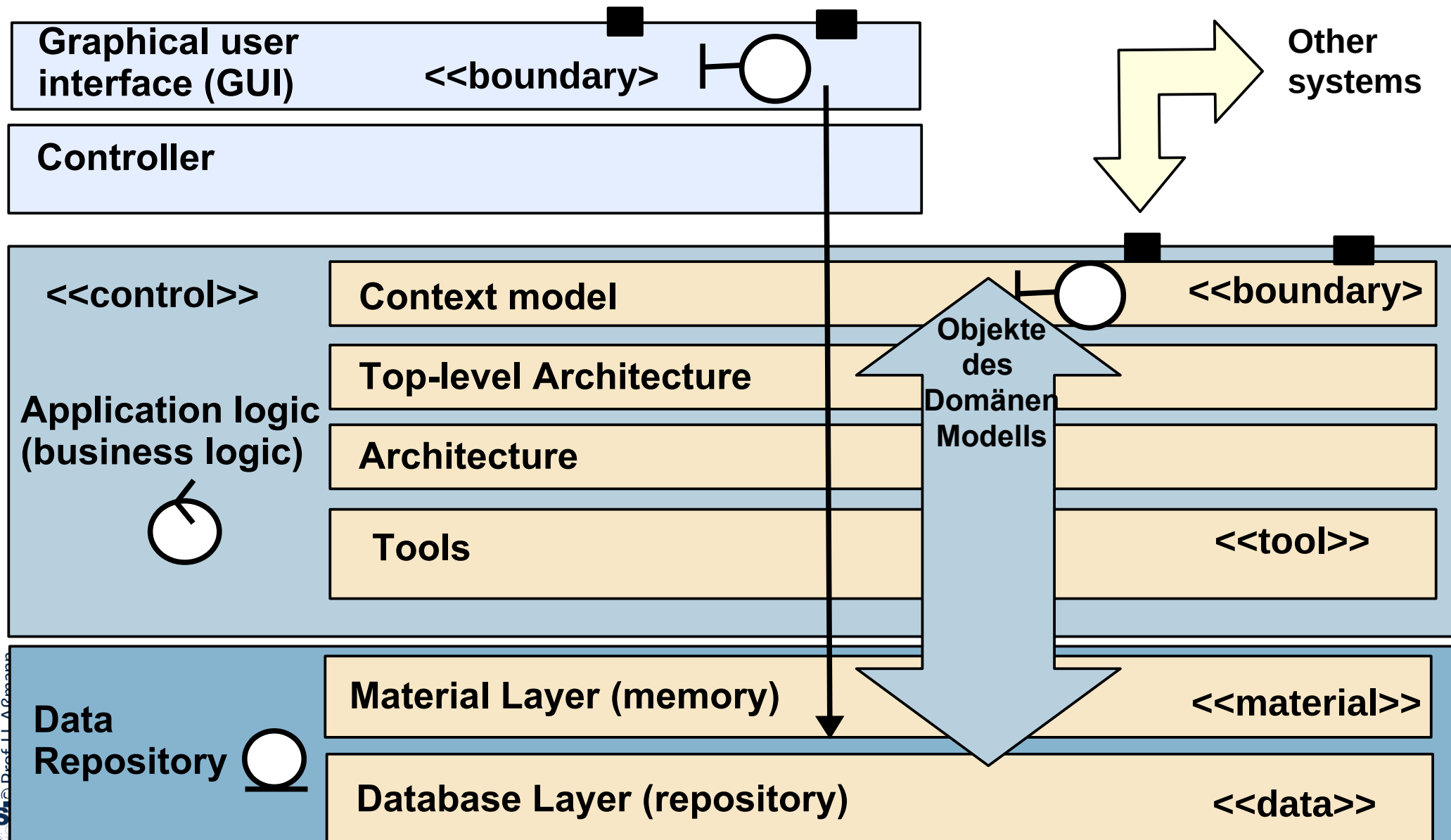
1. Überblick Objektorientierte Analyse
 1. (schon gehabt:) Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodellgetriebene Modellierung mit UML
 1. Strukturelle metamodellgetriebene Modellierung für das Domänenmodell
 2. Strukturelle Modellierung von komplexen Objekten
 3. Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen (Verhaltensanalyse)
 1. Funktionale Verfeinerung: Dynamische Modellierung und Szenarienanalyse mit Aktionsdiagrammen
 2. Funktionale querschneidende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen
 3. (Funktionale querschneidende Verfeinerung für komplexe Objekte)
4. Beispiel Fallstudie EU-Rent



Warum ist dieses Kapitel wichtig?

- ▶ Kontextmodelle gehören ins Pflichtenheft und bilden einen wesentlichen Teil des Vertrags zwischen Softwarefirma und Kunde.
- ▶ Ihre präzise Spezifikation ist essentiell für das Projekt.

Warum komplexe Objekte des Domänenmodells alle Schichten kreuzen, auch CM und TLA



Q5: Schritte der Modellierung in Bezug auf Schichten des Systems

Schichten	Analyse	Entwurf	Feinentwurf	Implementierung	
Benutzungsschnittstelle (Boundary)	Controller		Verfeinerung		
	Kontextmodell	Aufstellung aus Domänenmodell; Erarbeitung System-schnittstellen	stabil	Umsetzen auf jUML	stabil
Anwendungsgik (Control)	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

Toll Collect

http://de.wikipedia.org/wiki/Lkw-Maut_in_Deutschland

7

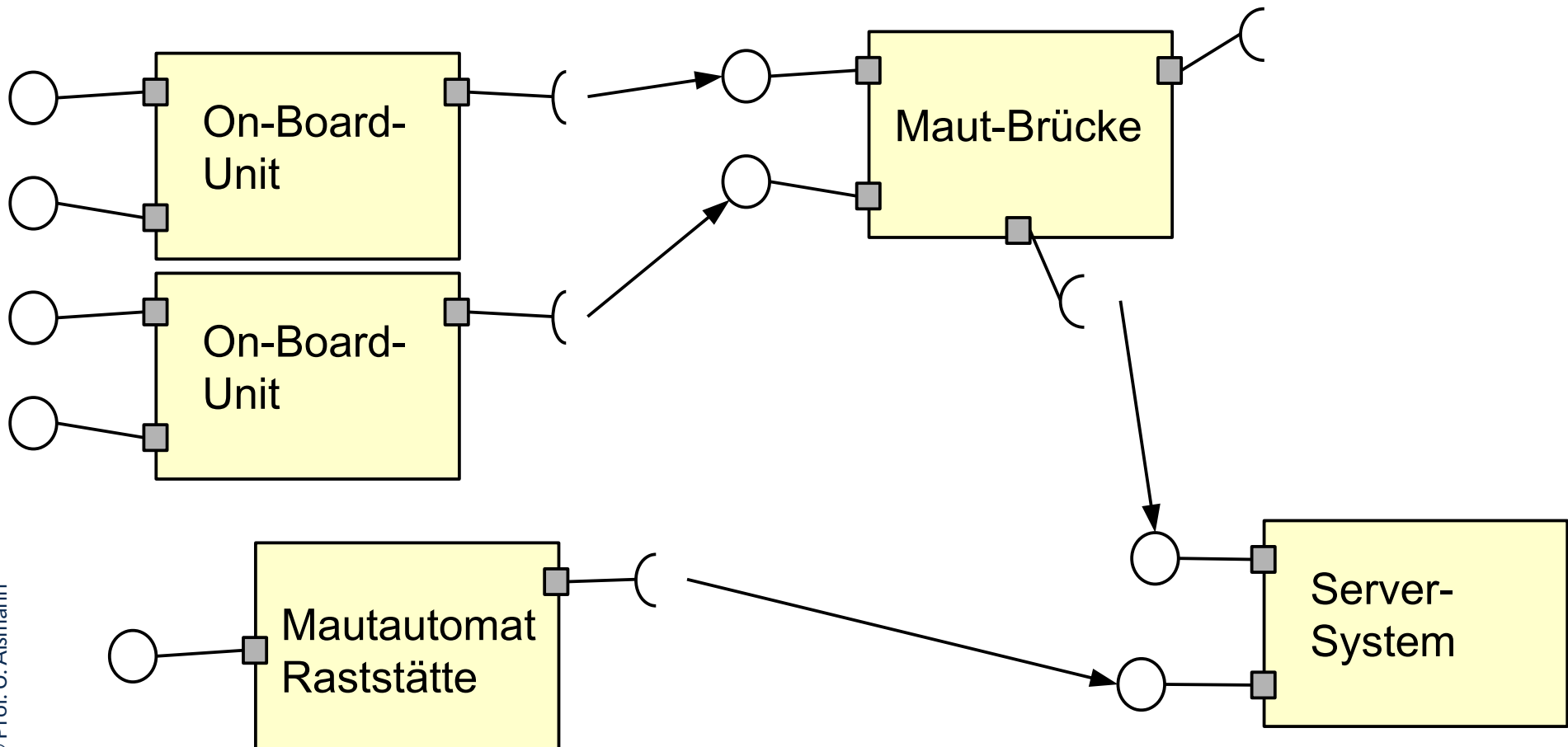
Softwaretechnologie (ST)

- ▶ 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)

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



Notation: UML Komponenten

Die Fragen der Systemanalyse im Kundengespräch

- ▶ Def.: Die **Systemanalyse** fragt nach den Schnittstellen und der Struktur der obersten Schichten eines Systems.

Die obersten Schichten eines Systems sind meist hierarchisch gestaltet. Typische Fragen der Systemanalyse:

- ▶ *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)*

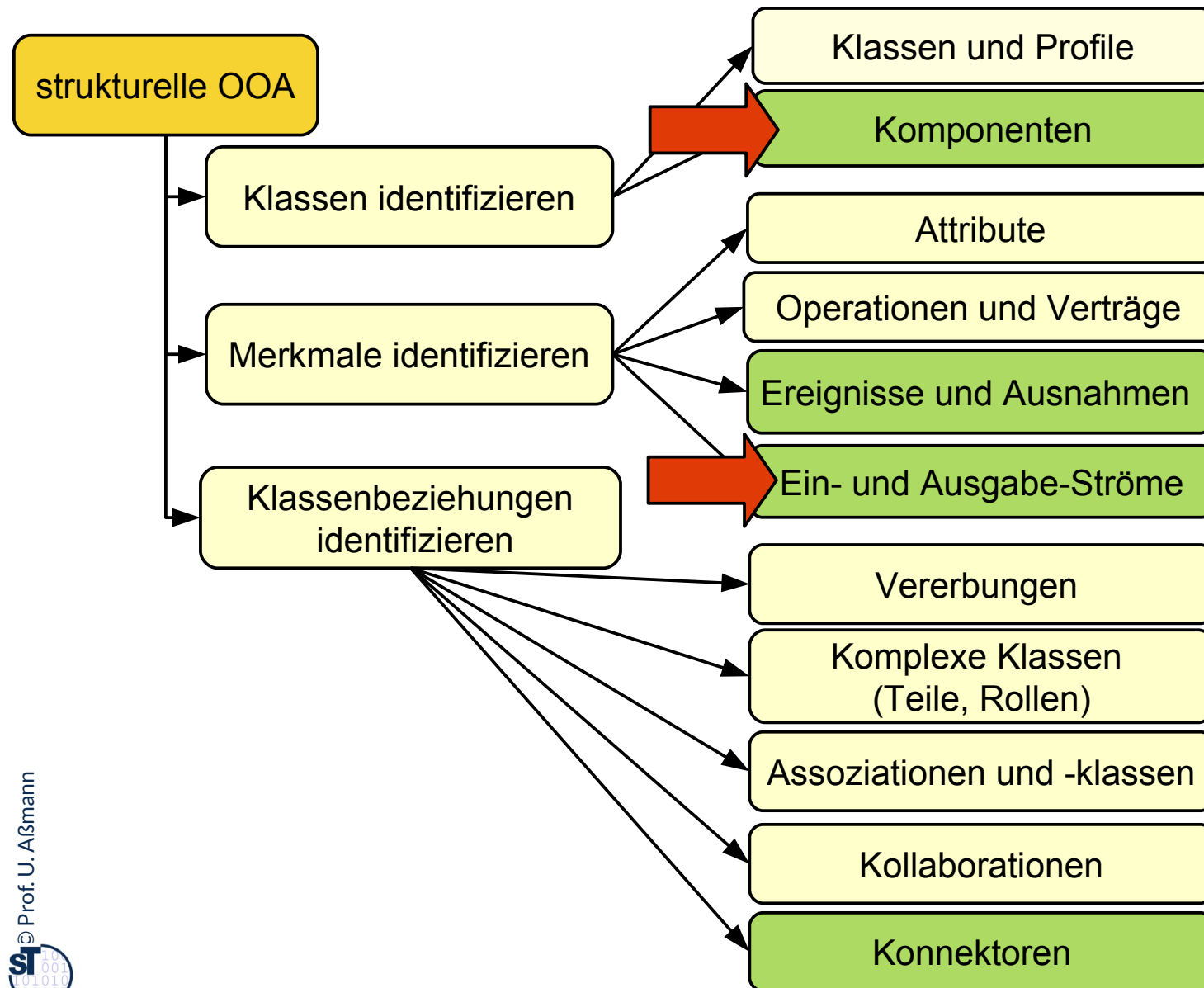
33.1 UML-Komponentenklassen für Verbundobjekte (Hierarchische komplexe Klassen)

- ▶ In der objektorientierten Systemanalyse wird das System als Großobjekt (komplexes Objekt) betrachtet, das
 - an die Umgebung Dienste anbietet
 - von der Umgebung Dienste beansprucht
- ▶ UML-Komponenten sind hierarchisch aggregierte komplexe Klassen mit *angebotenen* und *benötigten* Schnittstellen



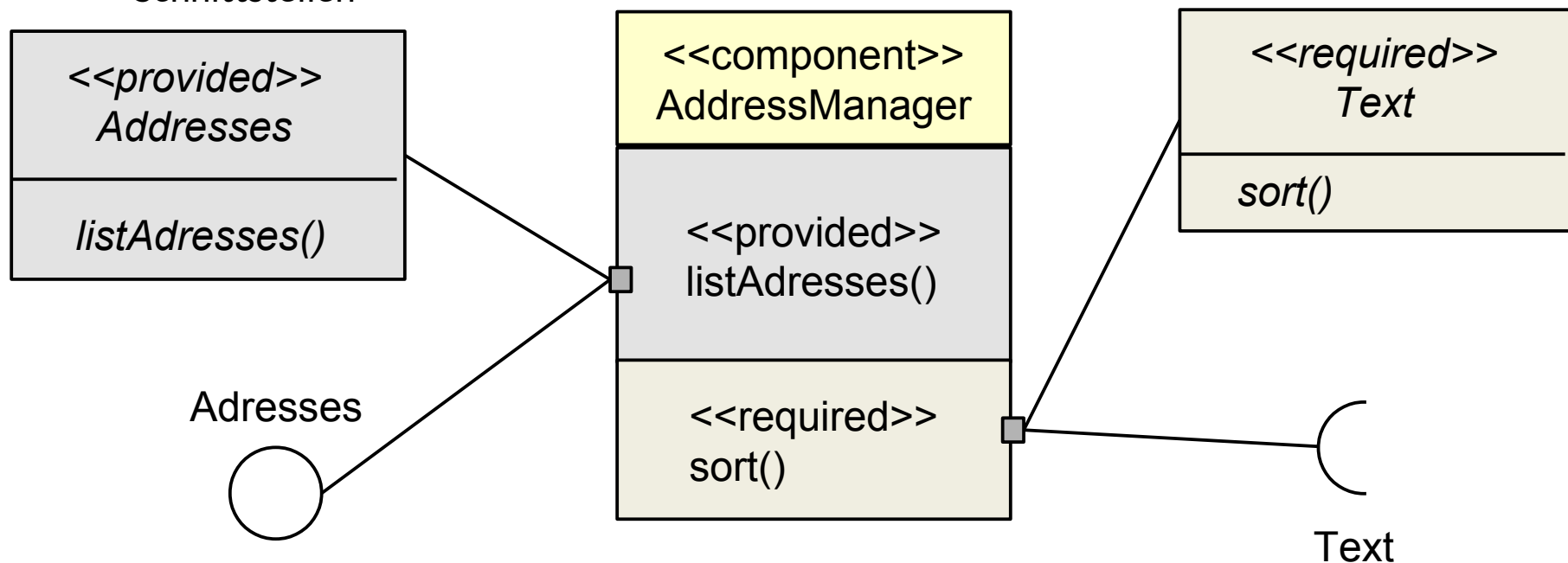
Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

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



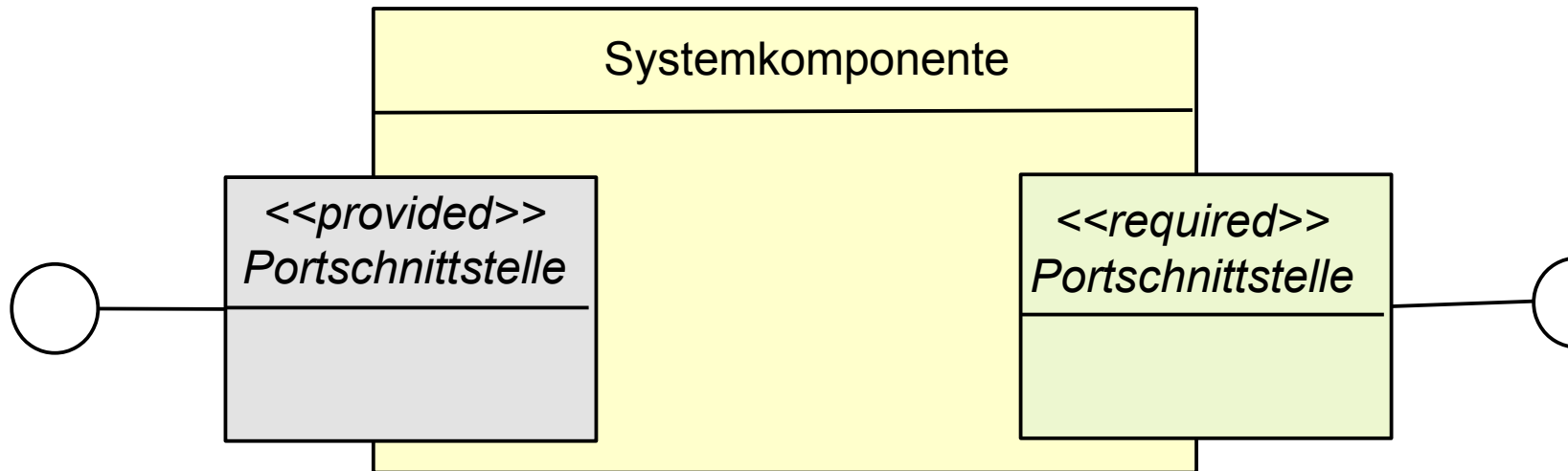
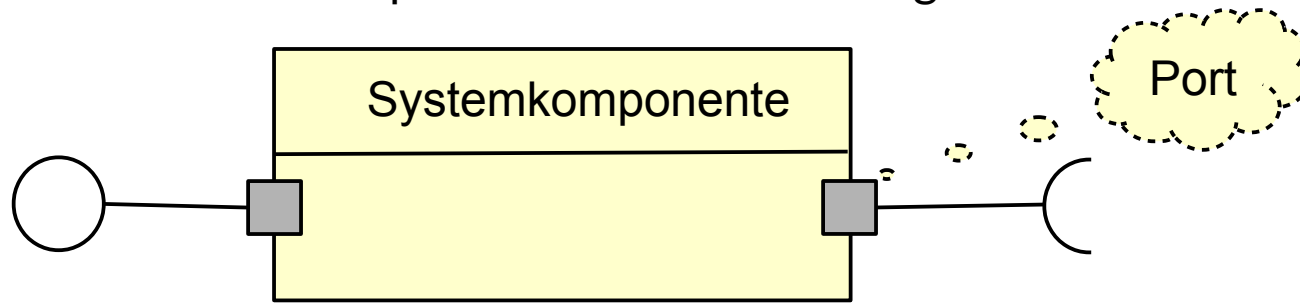
Komponenten mit Lollipops und Plugs (Balls and Sockets)

- ▶ Klassen, die *angebotene (provided)* von *benötigten (required)* Schnittstellen unterscheiden, heißen (Klassen-)Komponenten
 - Eine benötigte Schnittstelle spezifiziert, welche Partnerklasse die Klasse zum Ausführen benötigt (wird in separatem Compartment notiert)
 - Java: wird *nicht spezifiziert*, sondern vom Übersetzer herausgefunden
 - I.d.R. Sind UML-Komponenten komplexe Objekte mit vielen Unterobjekten
 - Komponenten sind einfacher wiederzuverwenden, da alle Aufrufabhängigkeiten explizit sind
 - Das **Kontextmodell** einer Komponente besteht aus der Menge ihrer angebotenen und benötigten Schnittstellen



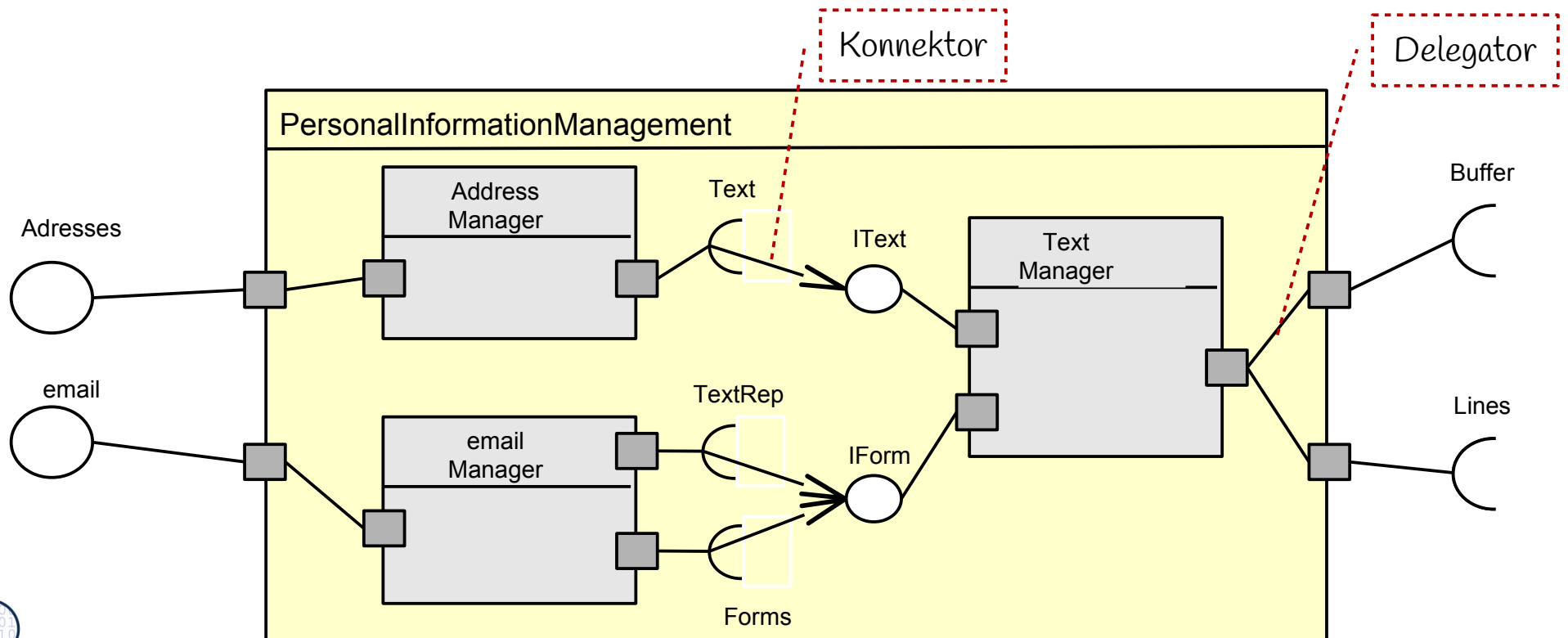
Anschlüsse (Ports) gruppieren Schnittstellen

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



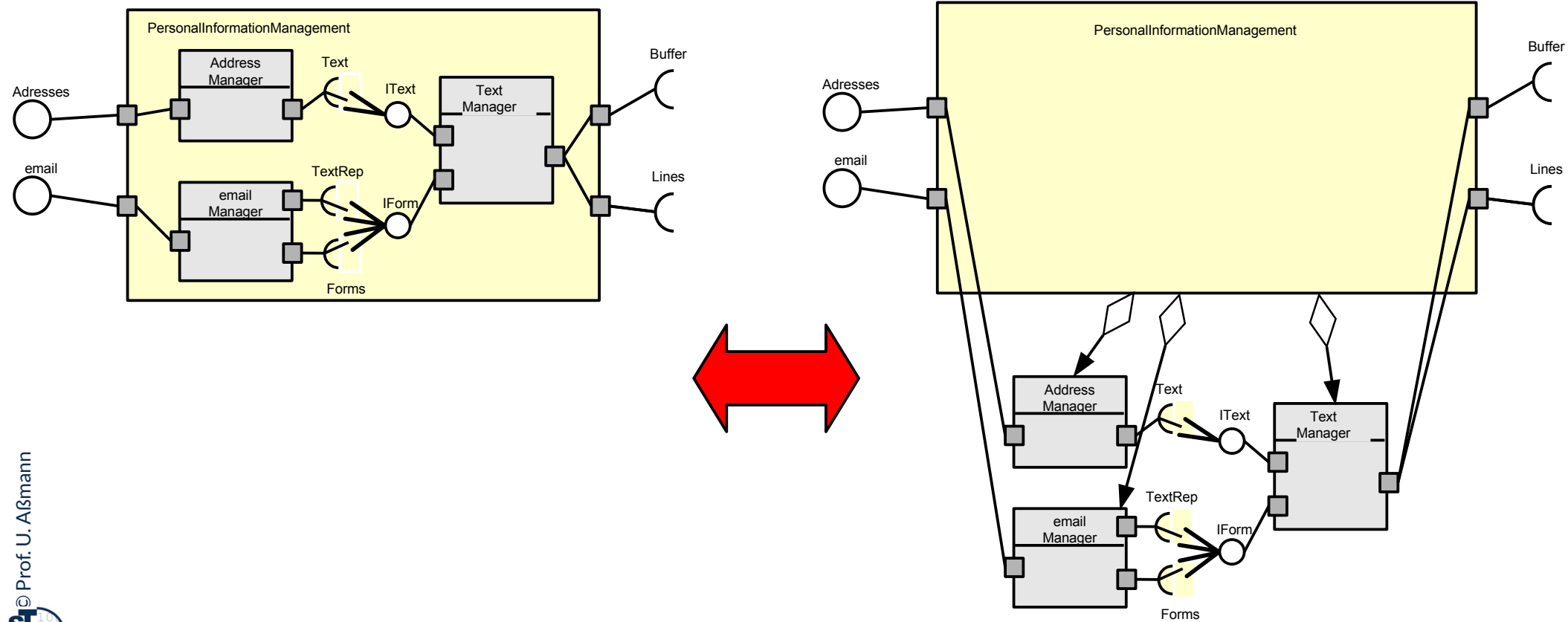
Schachtelung von Klassen zu UML-Komponenten

- ▶ Eine **UML-Komponente** 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

- ▶ UML-Komponenten sind komplexe Großobjekte, 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



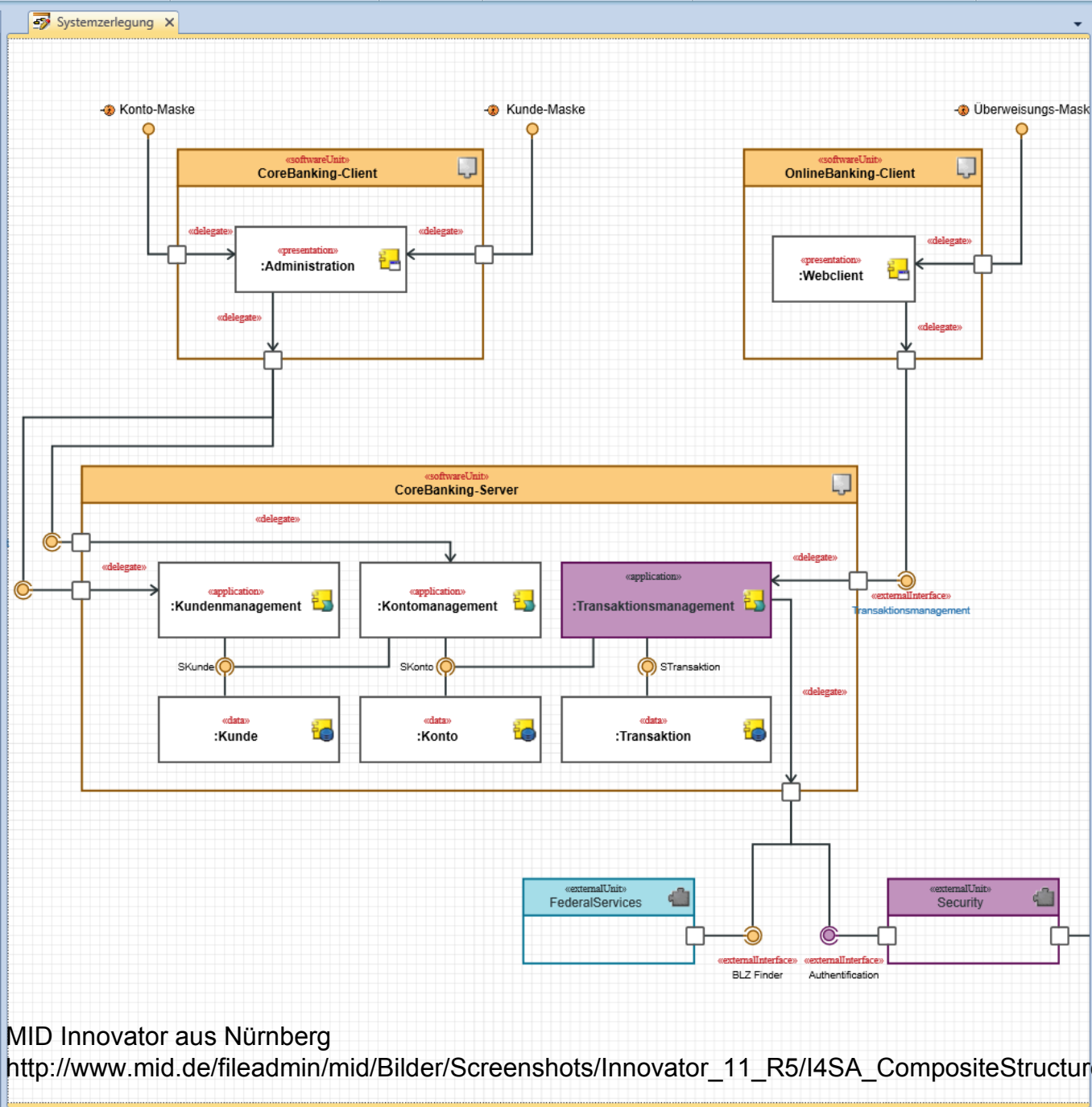
Modellinhalt

Modellstruktur

<Struktur durchsuchen (Strg+G)>

- CoreBankingSystem
 - systemModel
 - Evaluation
 - Projection
 - Systemzerlegung**
 - Presentation
 - Applikation
 - Daten
 - Basisklassen
 - Ereignisse
 - Ausnahmefehler
 - CoreBankingSystem
 - EJB3 Construction
 - Implementation
 - systemModel Management

Diagramme
Modellstruktur
Modellausschnitt
Lesezeichen



Details

- Strukturierte Classifier
 - CoreBanking-Client
 - AdminView : Administration
 - CoreBanking-Server
 - KontoCtrl : Kontomanagement
 - KontoMdl : Konto
 - KundeCtrl : Kundenmanagement
 - KundeMdl : Kunde
 - TransCtrl : Transaktionsmanagement
 - TransMdl : Transaktion
 - FederalServices
 - OnlineBanking-Client
 - WebView : Webclient
 - Security
 - Komponentenschnittstellen
 - Konto-Maske
 - Kunde-Maske
 - Überweisungs-Maske
 - Login
 - Kontomanagement
 - Transaktionsmanagement
 - Kundenmanagement
 - BLZ Finder
 - Authentication

Eigenschaften

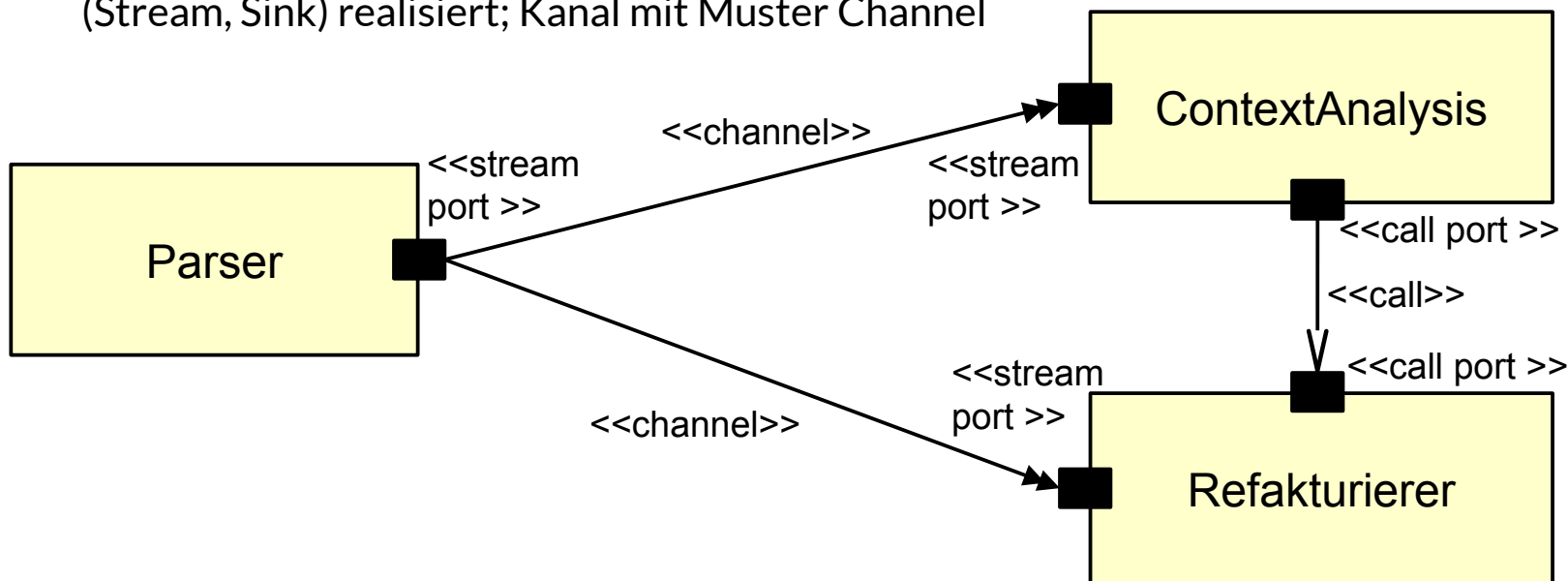
Komposition... Systemzerlegung

- Merkmale
 - Stereotyp systemArchitec
 - Sichtbarkeit Öffentlich
 - Besitzer Projection
- Zugriffsrechte

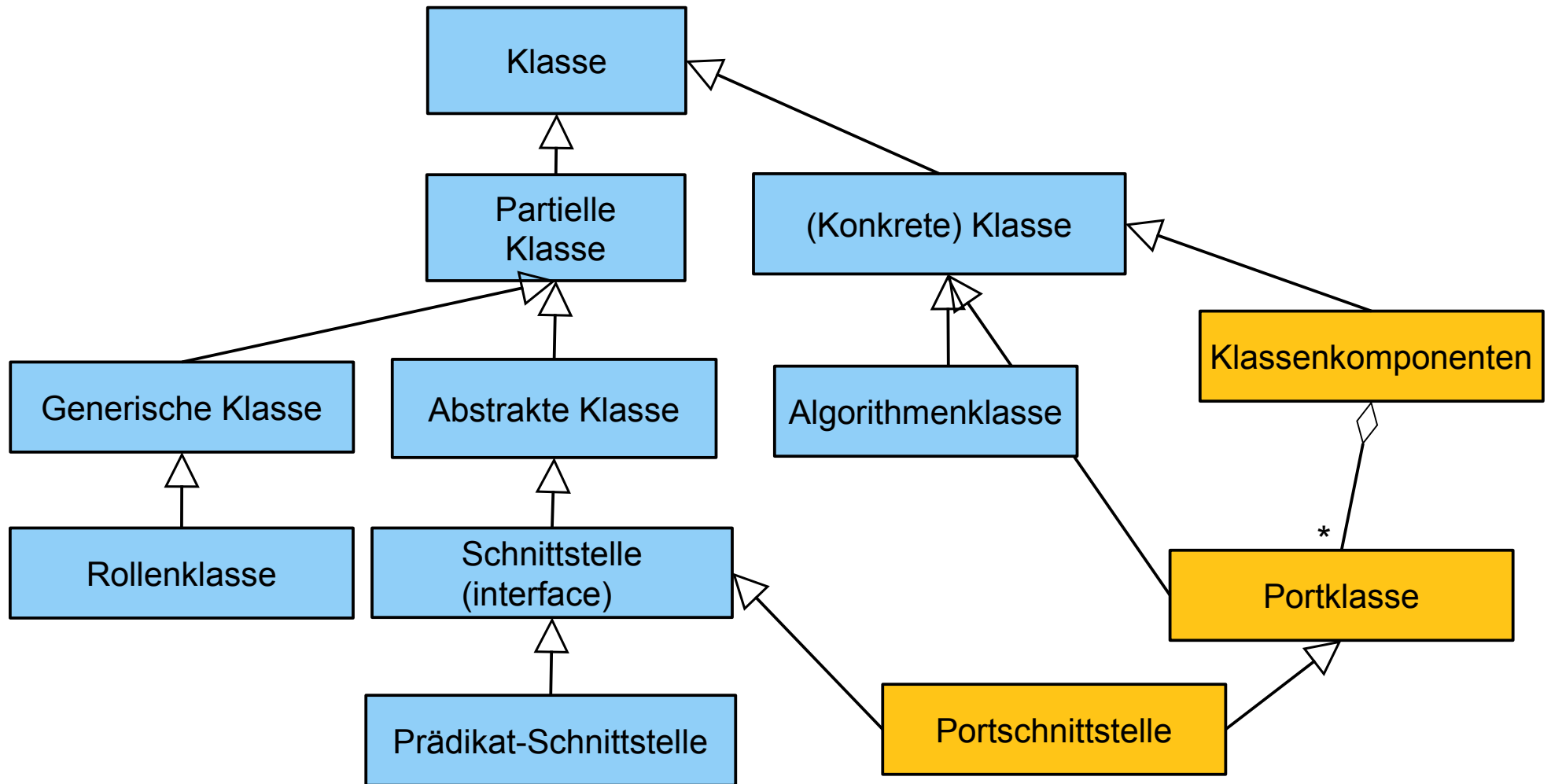
Eigenschaften Abhängigkeiten

Strom-Anschlüsse von Komponenteklassen und Kanal-Konnektoren

- ▶ 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 kontinuierlich 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 **Konnektoren (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)



33.2 Das Kontextmodell

- ▶ Neben der GUI (graphischen Oberfläche, interaktive Schnittstelle) beinhaltet das Kontextmodell alle Schnittstellen eines Systems.



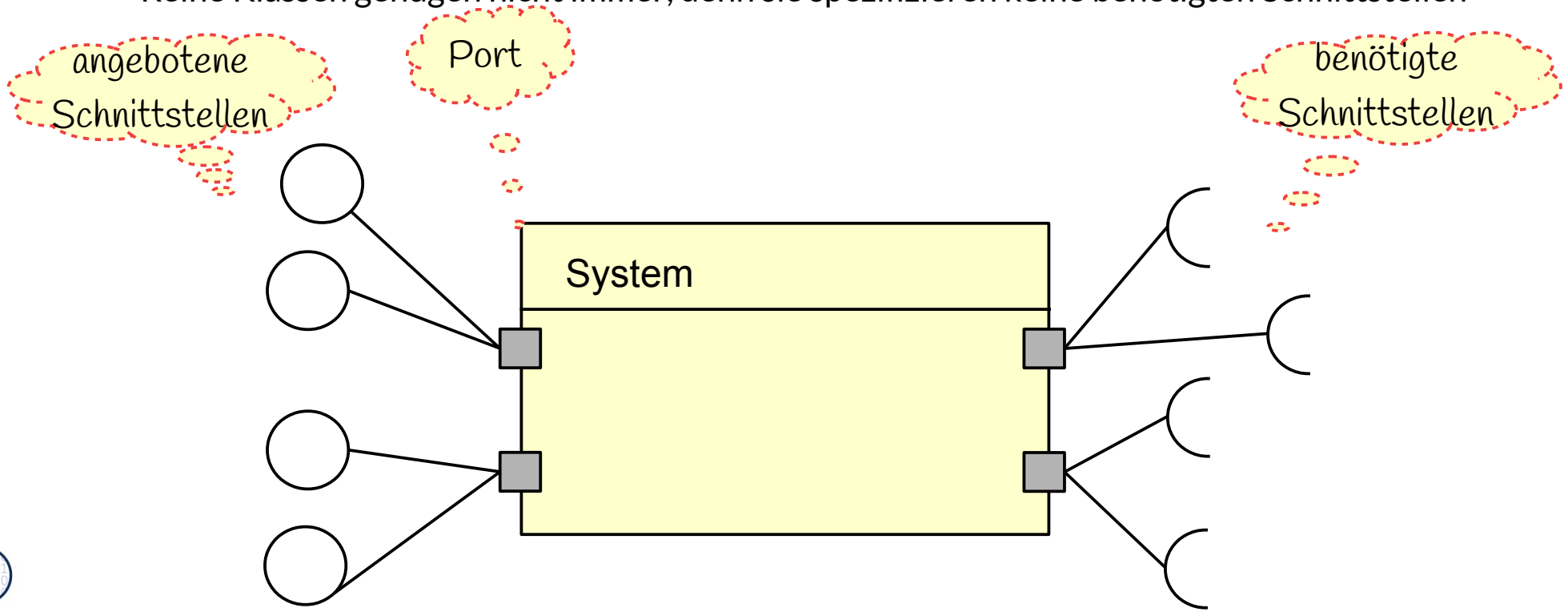
Kontextmodell (Schnittstellenmodell)

- ▶ Das System und seine obersten Schichten wird meist als *Klassenkomponente* verstanden
 - Mit angebotenen und benötigten Schnittstellen, mit ports
 - Hierarchisch durch die Ganz/Teile-Relation verfeinert
- ▶ Das **Kontextmodell (Schnittstellenmodell) eines Systems** enthält alle 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

Das System wird oft als eine große komplexe Komponente gesehen, dessen Schnittstellen vom Kontextmodell beschrieben werden.

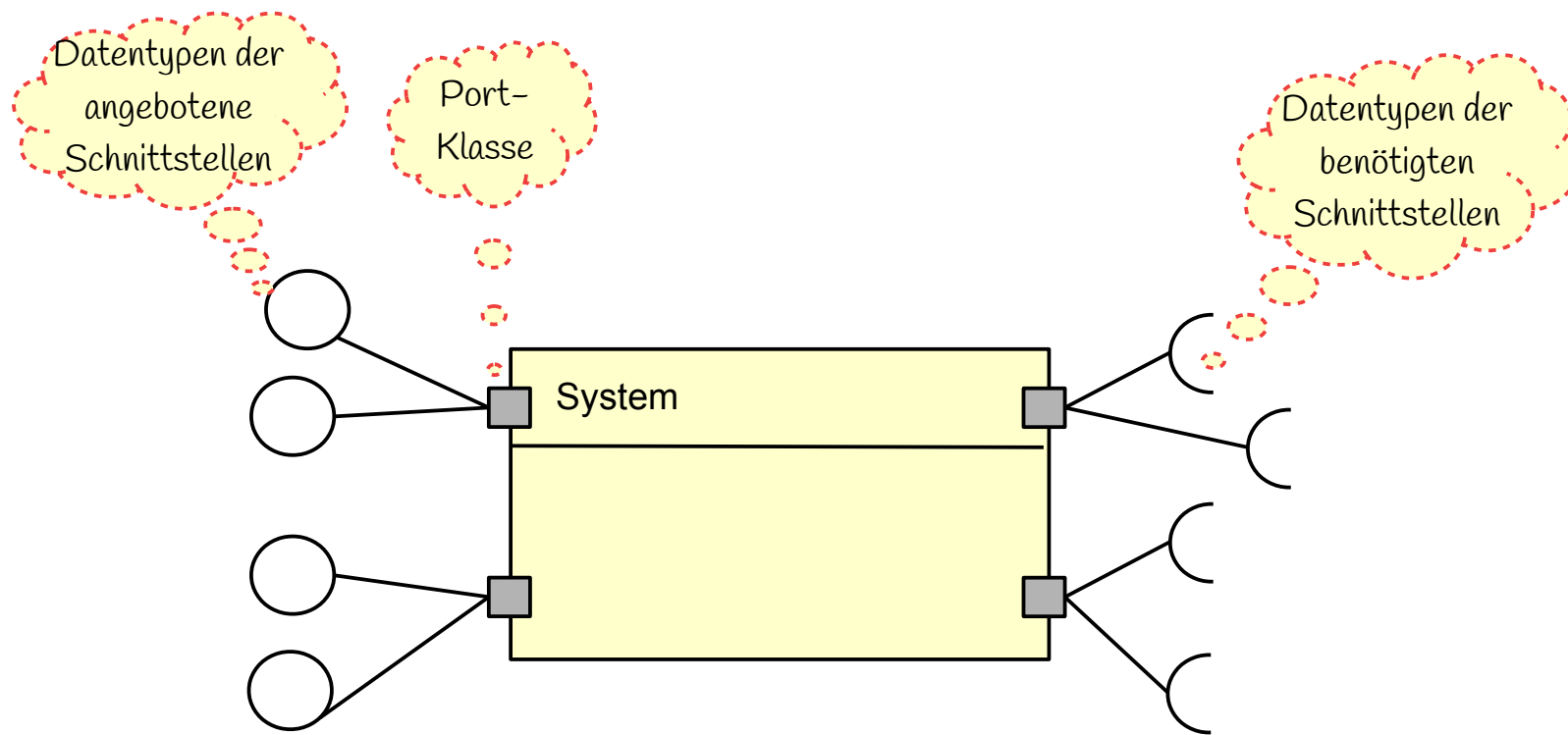
UML-Komponenten im Kontextmodell

- ▶ 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 Kontextmodell enthält *angebotene* und *benötigte* Schnittstellen:
 - Funktionale und Strom-Schnittstellen (call und stream ports), gerade zu anderen Systemen
 - GUI-Bildschirme, Masken, Formulare
- Reine Klassen genügen nicht immer, denn sie spezifizieren keine benötigten Schnittstellen



Datentypen des Kontextmodells stammen aus dem Domänenmodell

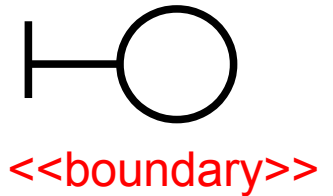
- ▶ Das Domänenmodell stellt die Typen für die technischen Objekte, Prozesse und Funktionalitäten des Kontextmodells zur Verfügung
 - Anschlüsse: Funktionale und strombasierte Schnittstellen (call und stream ports)
 - GUI-Bildschirme, Masken, Formulare
 - Port-Klassen und Schnittstellen sind Grenzklassen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



Unterscheidung von Systemschnittstellenklassen von anderen Klassen mit dem BCD-Profil

- ▶ **Schnittstellen-Klassen** (Portschnittstellen, Portklassen, GUI-Klassen, Grenzklassen, boundary classes)

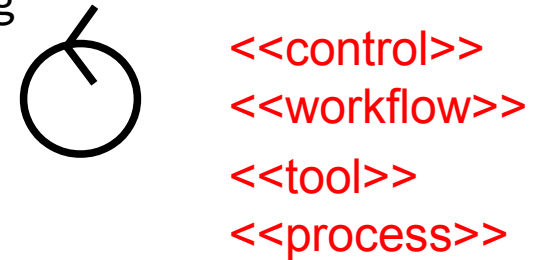
- Teil einer Schnittstelle oder Benutzerschnittstelle



- ▶ **Steuerungsklassen** (control classes)

- Aktive Klasse der Anwendungslogik, die die Ausführung eines Prozesses steuert

- Mit oder ohne Zustand



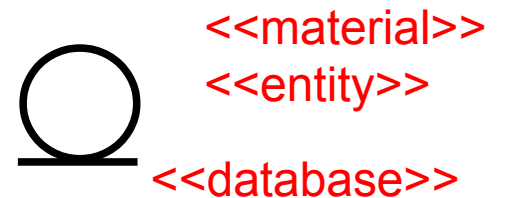
- ▶ **Material:** Passive Klasse, beschreibt Daten

- 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



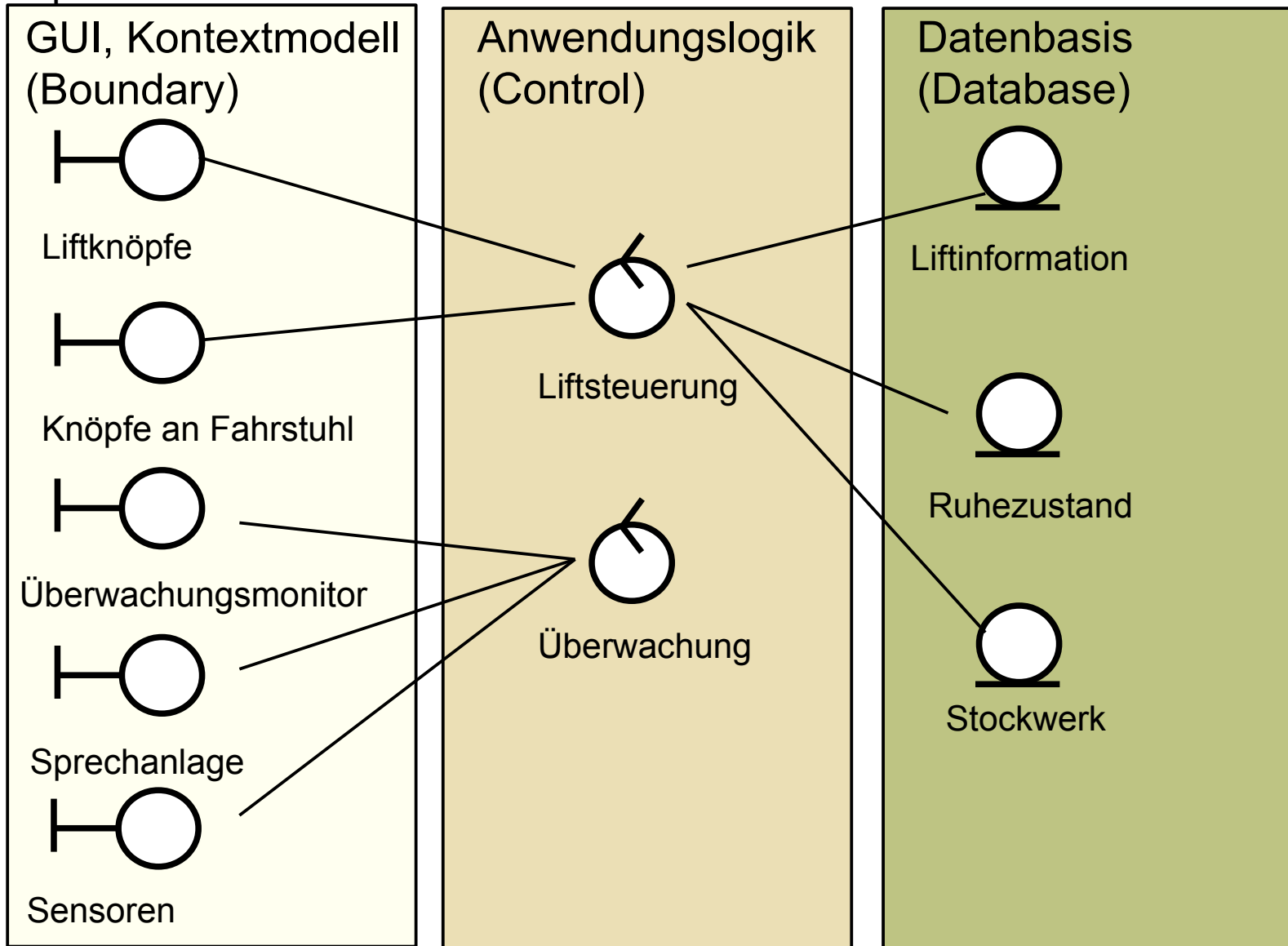
- ▶ BCD-Architektur: 3-Schichten-Architektur (3-tier architecture)

<<container>>

- ▶ BCED-Architektur: 4-Schichten-Architektur (4-tier architecture)

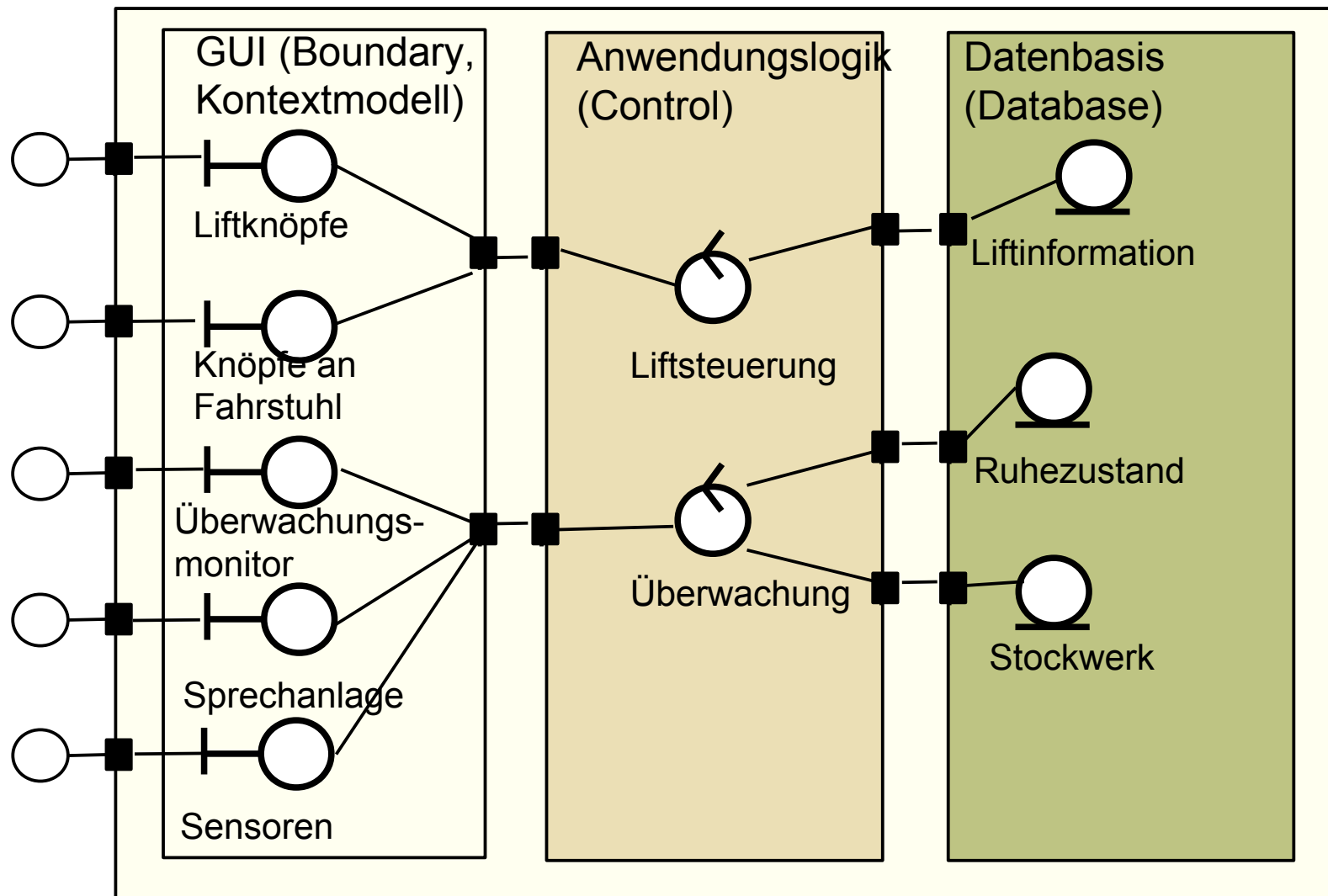
Bsp: Analyse-Klassenmodell Liftsteuerung im BCD-Stil

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



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

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



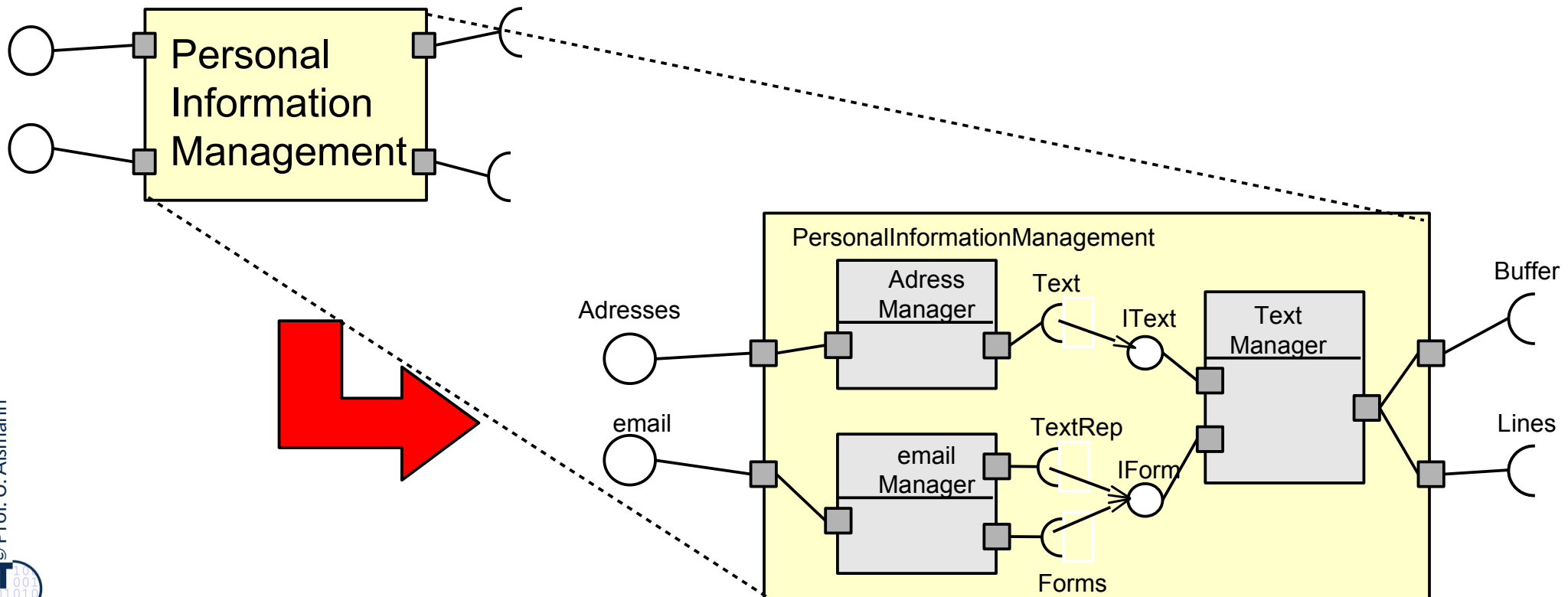
33.3 Top-Level-Architektur

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



Top-Level-Architektur

- ▶ 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 Komponenteklassen werden sichtbar



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

- ▶ 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

33.3 Asynchrone Systemmerkmale im Kontextmodell

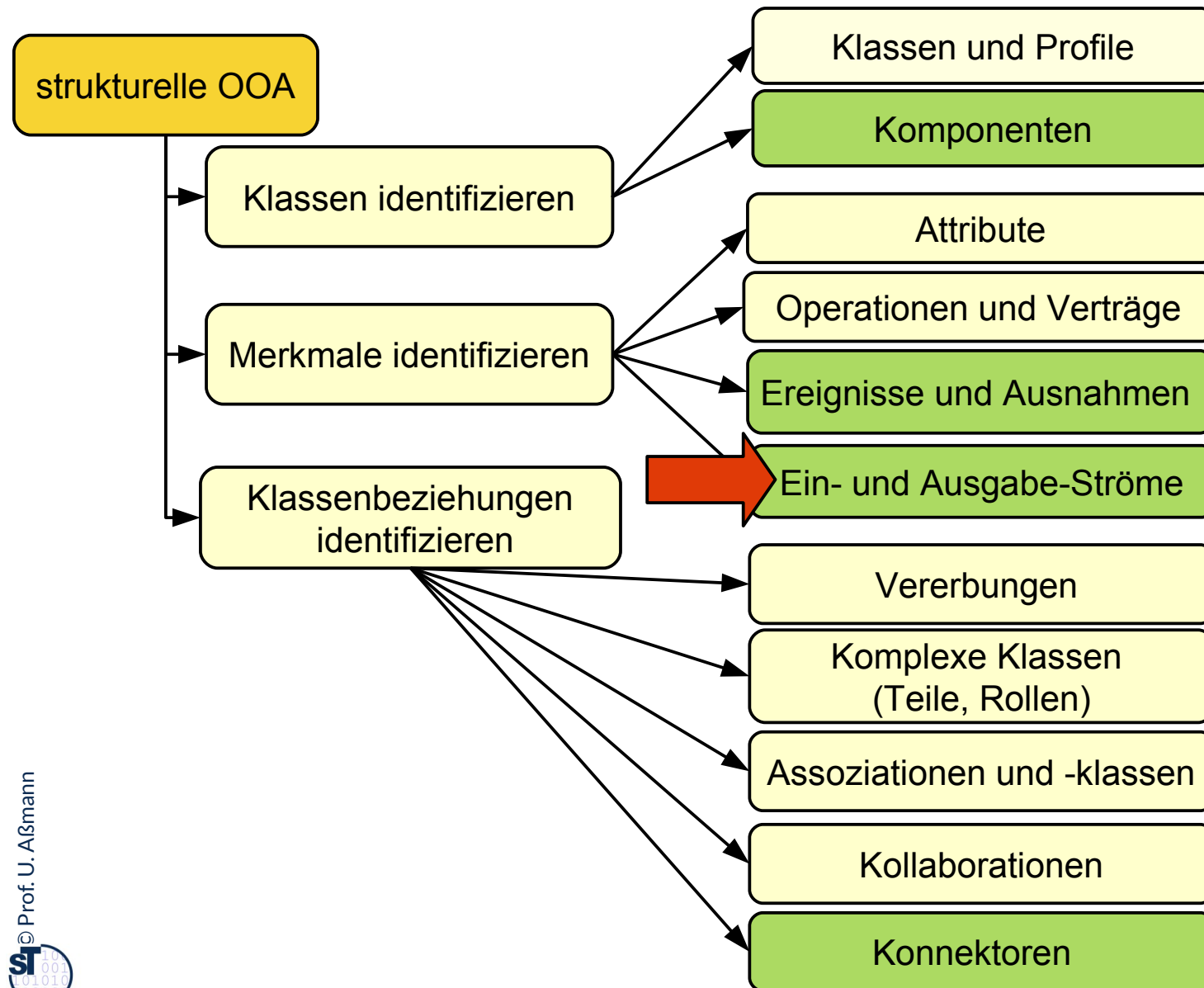
Ereignisse und Ausnahmen (Exceptions)



DRESDEN
concept
Exzellenz aus
Wissenschaft
und Kultur

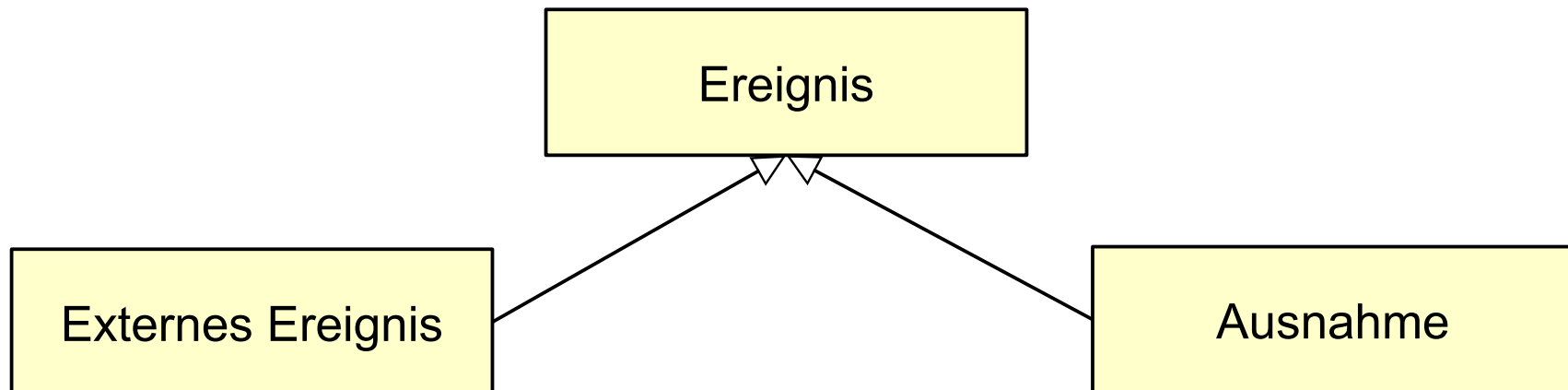
Q6: Schritte der strukturellen, metamodellgetriebenen Analyse

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



Asynchrones Verhalten des Systems

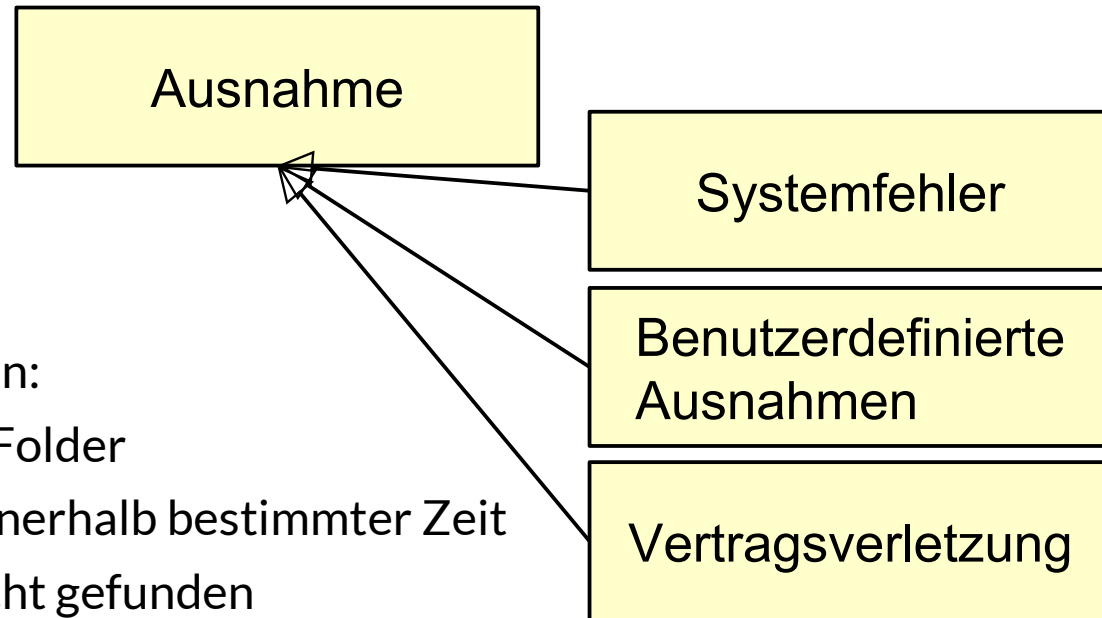
- ▶ Im Kontextmodell muß zusätzlich das *asynchrone Verhalten des Systems* spezifiziert werden
- ▶ **Externe Ereignisse** – und wie soll das System darauf reagieren?
 - Benutzererzeugte Ereignisse
 - Plattform-erzeugte Ereignisse (Platte voll, Power out, ...)
 - Sensorwerte
 - Externe Ereignisse können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



Ausnahmen im Kontextmodell

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

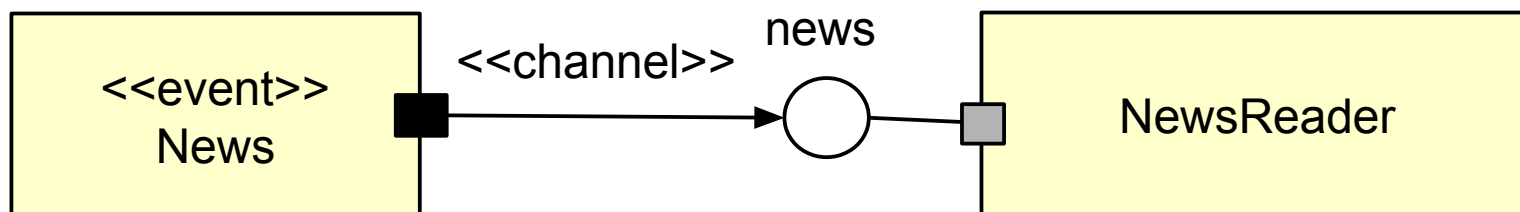
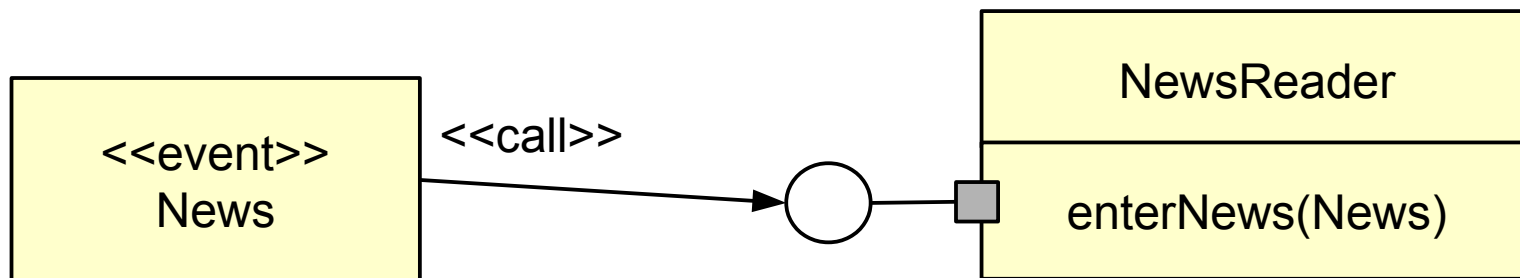
- Systemfehler:
 - Division durch 0
 - Zugriff durch null-Zeiger
 - Platte voll
- Benutzerdefinierte Ausnahmen:
 - Zu viele Dateien in einem Folder
 - Benutzer reagiert nicht innerhalb bestimmter Zeit
 - Material in Datenbank nicht gefunden
- Vertragsverletzungen von Methoden:
 - Zeiger null
 - integer-Wert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

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



Unterschied Domänen- vs. Kontextmodell

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

- ▶ 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?
- ▶ Warum ist ein System ein komplexes Großobjekt? Wie gliedert es sich?
- ▶ Erkläre den Zusammenhang zwischen Anschlüssen, Konnektoren und Ereignissen.



33.A.1 Einsatz von Adapter zum Brücken von Schnittstellen in der Top-Level-Architektur

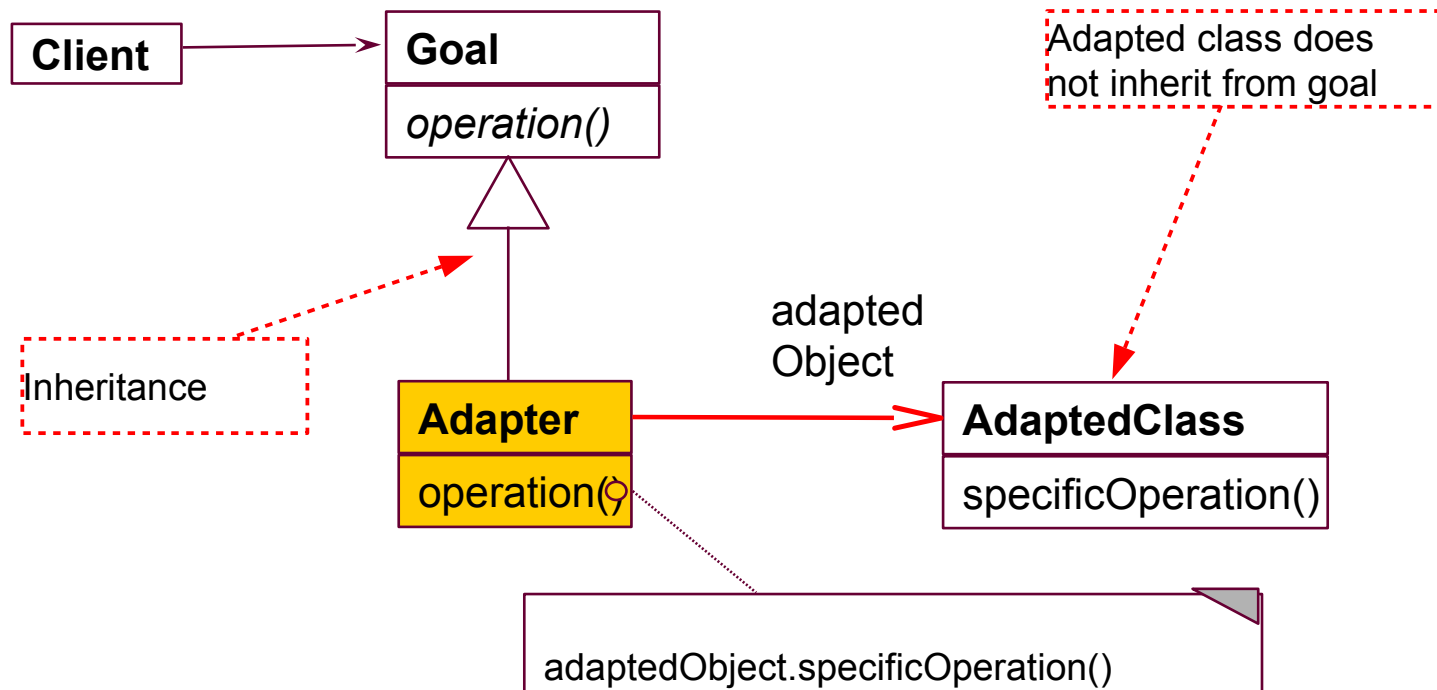


Aufgabe des Kontextmodells: Verbindung nach außen

- ▶ 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

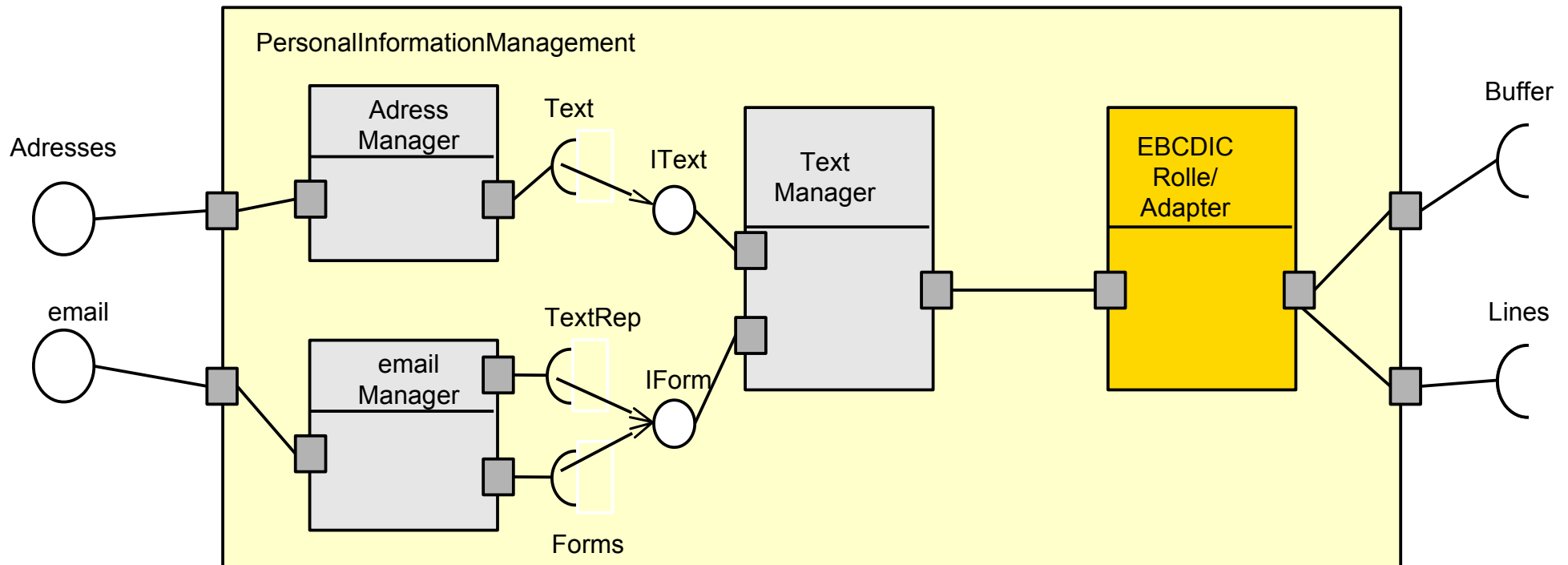
Entwurfsmuster Adapter (Objektadapter) (Wdh.)

- ▶ 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



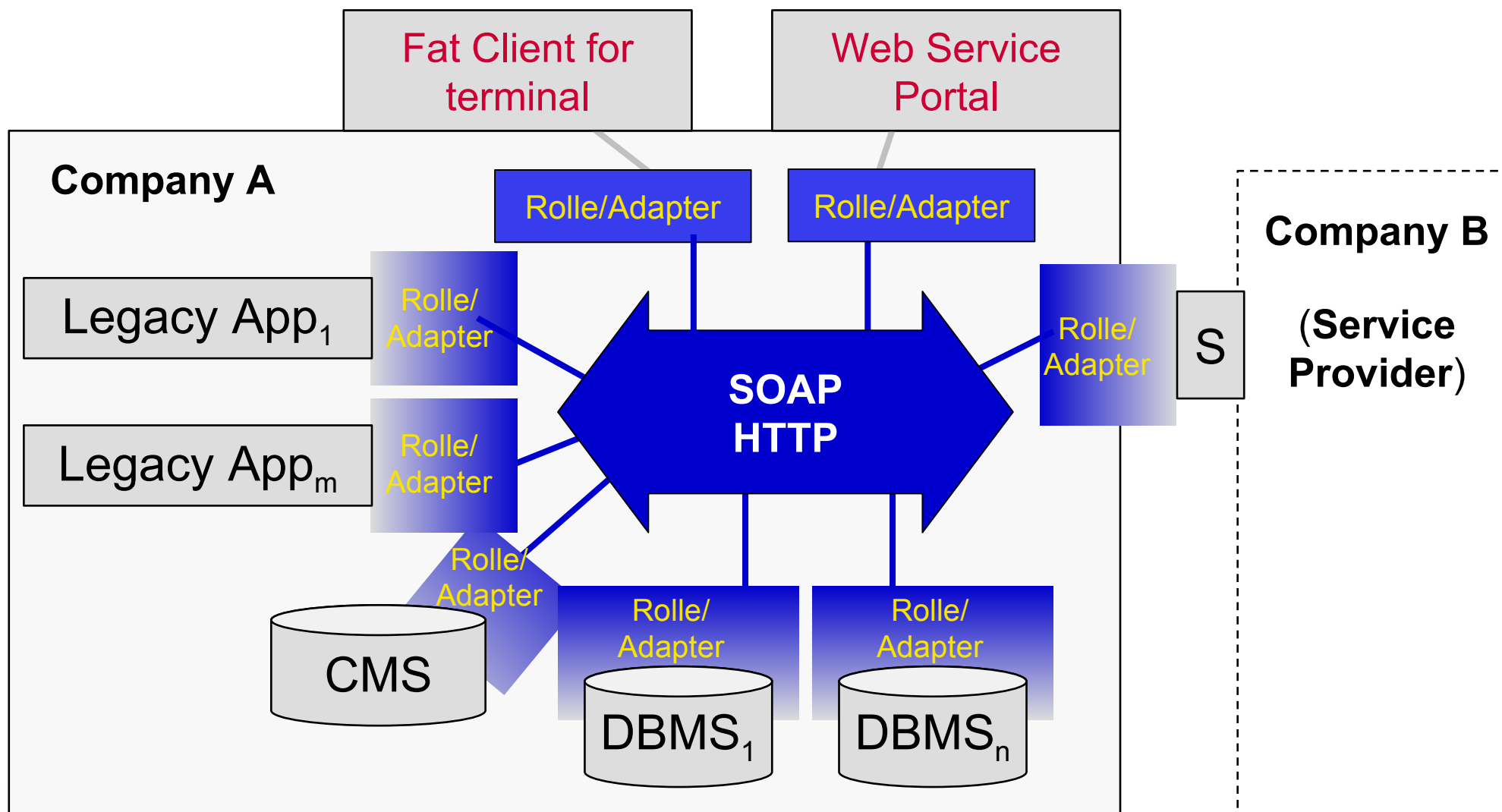
Top-Level-Architektur mit Adaptern

- ▶ 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 Entwurfsmuster Konnektor (SOAP/HTTP)

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



Profil “Aktive und Passive Klassen”

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

- Im einfachsten Fall existiert *eine* aktive Klasse

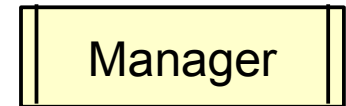
- ▶ Aktive Objekte

- 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

- ▶ 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 <<event>>
- ▶ Ereignisklasse (Event) <<organization>>
 - Ereignis in der Zeit, extern zum Objekt
- ▶ Organisation <<people>>
 - Verkörpert Wissen über eine Organisationseinheit
- ▶ Menschen (People)
- ▶ Plätze (Places class) <<place>>

Beispiel: Rumbaugh-Profil

- ▶ 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>>

33.A.2 Weitere Arten von Schnittstellen und Klassen

(zur Information)

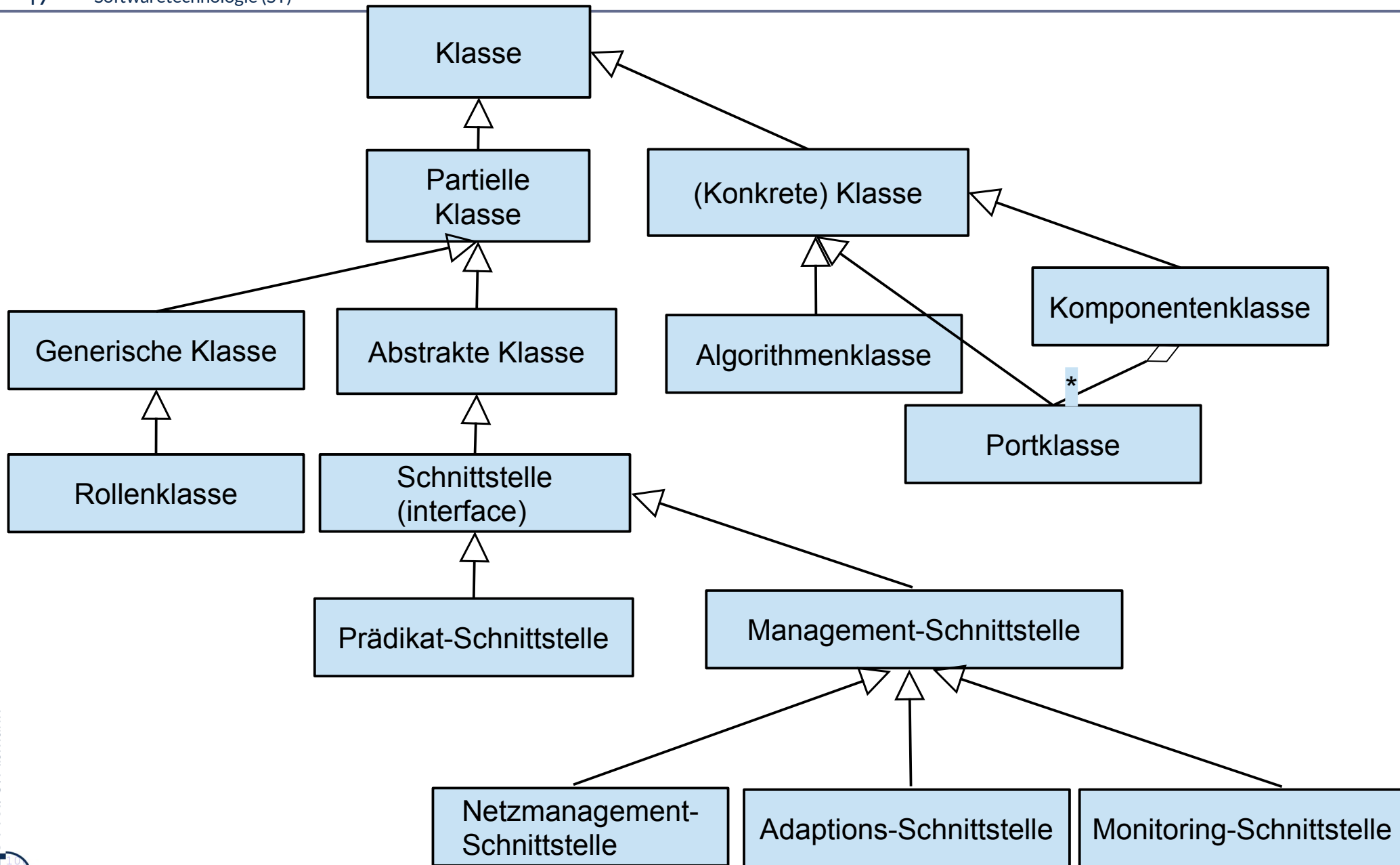


DRESDEN
concept
Exzellenz aus
Wissenschaft
und Kultur

Weitere Arten von Schnittstellen in komplexen Objekten

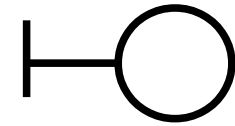
- ▶ **Funktionale Schnittstellen** enthalten Funktionen, die direkt auf den Zustand des Objekts zugreifen
 - **Prädikat-Schnittstellen**, die Prädikate auswerten
- ▶ **Managementschnittstellen** enthalten Funktionen, die das Objekt und seine Nachbarn verwalten:
 - **Netzmanagement-Funktionen** verändern 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)

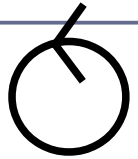


Identifikation von Systemschnittstellenklassen für das Kontextmodell

- ▶ **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



- ▶ **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



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

Wie man ein System auf grobkörniger Ebene strukturiert

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 19-0.1, 15.06.19

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



DRESDEN
concept
Exzellenz aus
Wissenschaft
und Kultur

Literatur

- ▶ 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 Softwaretechnologie (ST)

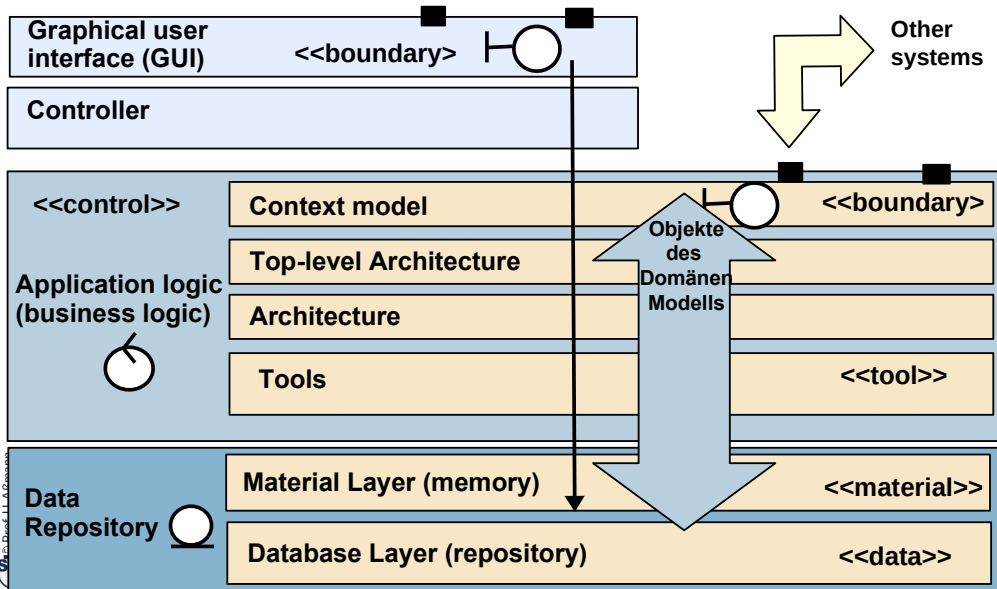
1. Überblick Objektorientierte Analyse
 1. (schon gehabt:) Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodelldgetriebene Modellierung mit UML
 1. Strukturelle metamodelldgetriebene Modellierung für das Domänenmodell
 2. Strukturelle Modellierung von komplexen Objekten
 3. Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen (Verhaltensanalyse)
 1. Funktionale Verfeinerung: Dynamische Modellierung und Szenarienanalyse mit Aktionsdiagrammen
 2. Funktionale querschnittende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen
 3. (Funktionale querschnittende Verfeinerung für komplexe Objekte)
4. Beispiel Fallstudie EU-Rent



Warum ist dieses Kapitel wichtig?

- ▶ Kontextmodelle gehören ins Pflichtenheft und bilden einen wesentlichen Teil des Vertrags zwischen Softwarefirma und Kunde.
- ▶ Ihre präzise Spezifikation ist essentiell für das Projekt.

Warum komplexe Objekte des Domänenmodells alle Schichten kreuzen, auch CM und TLA



Q5: Schritte der Modellierung in Bezug auf Schichten des Systems

	Schichten	Analyse	Entwurf	Feinentwurf	Implementierung
Benutzungsschnittstelle (Boundary)	Controller			Verfeinerung	
	Kontextmodell	Aufstellung aus Domänenmodell; Erarbeitung System-schnittstellen	stabil	Umsetzen auf jUML	stabil
Anwendungsschicht (Control)	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

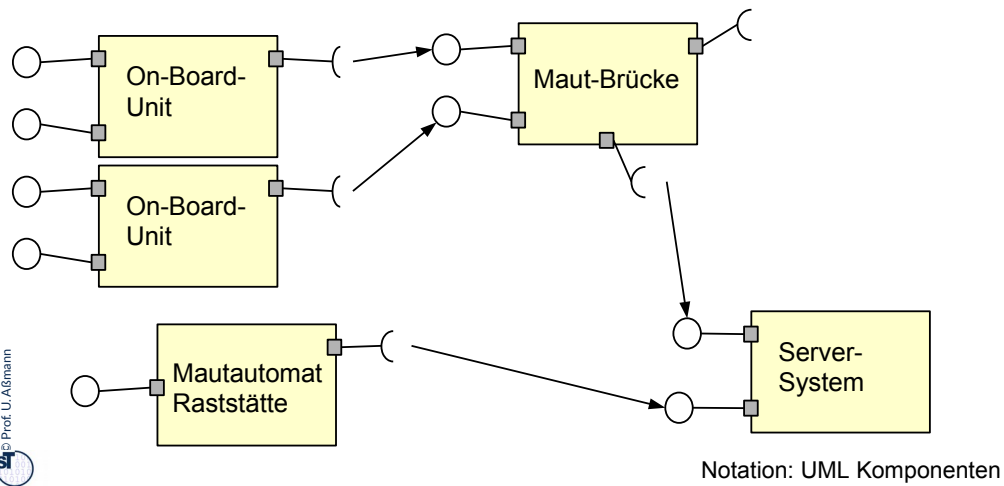
- ▶ Auftragnehmer Telekom und Daimler Financial Services
- ▶ Schätzung der Einnahmen auf 3 Mrd jährlich
- ▶ Inbetriebnahme geplant 31.8.2003, dann 2.11. 2003
 - Tägliche Vertragsstrafe 250k€
- ▶ Realisiert 1.1. 2005, vollständig 1.1.2006

- ▶ 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)

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



Die Fragen der Systemanalyse im Kundengespräch

- ▶ Def.: Die **Systemanalyse** fragt nach den Schnittstellen und der Struktur der obersten Schichten eines Systems.

Die obersten Schichten eines Systems sind meist hierarchisch gestaltet. Typische Fragen der Systemanalyse:

- ▶ **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)**





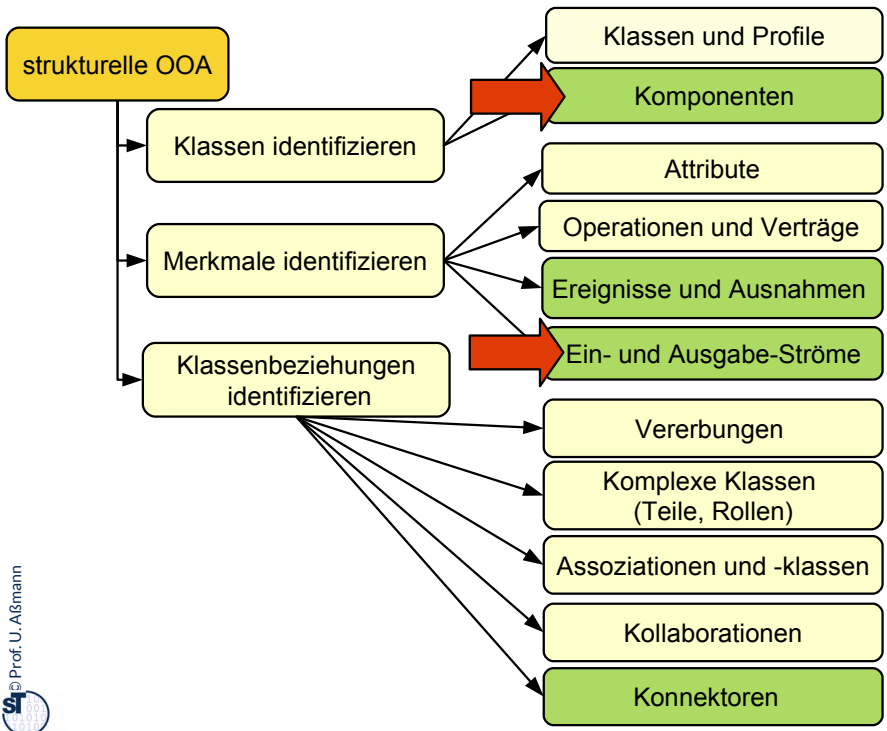
33.1 UML-Komponentenklassen für Verbundobjekte (Hierarchische komplexe Klassen)

- ▶ In der objektorientierten Systemanalyse wird das System als Großobjekt (komplexes Objekt) betrachtet, das
 - an die Umgebung Dienste anbietet
 - von der Umgebung Dienste beansprucht
- ▶ UML-Komponenten sind hierarchisch aggregierte komplexe Klassen mit *angebotenen* und *benötigten* Schnittstellen



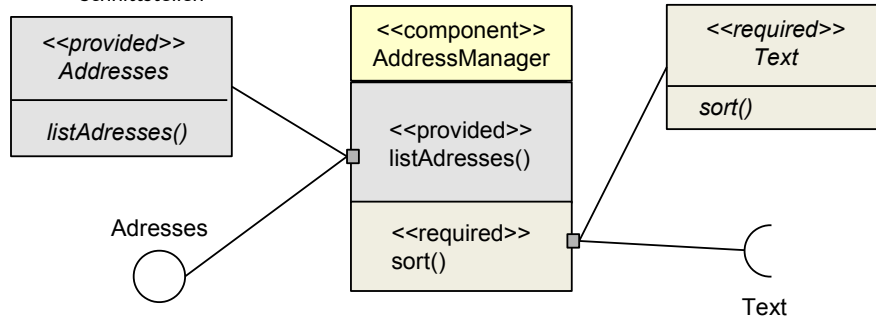
Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

► gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



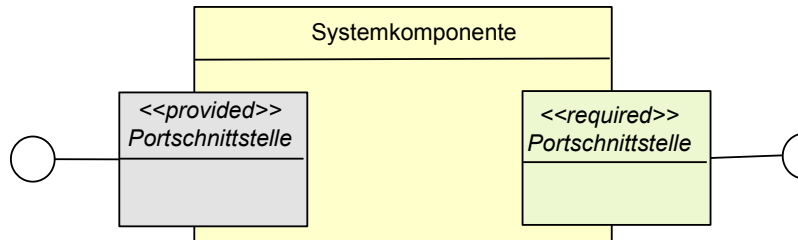
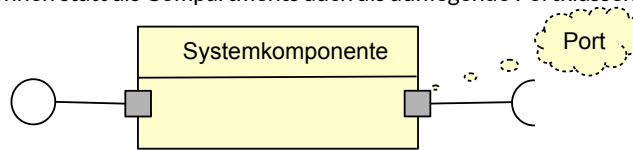
Komponenten mit Lollipops und Plugs (Balls and Sockets)

- ▶ Klassen, die *angebotene* (*provided*) von *benötigten* (*required*) Schnittstellen unterscheiden, heißen (Klassen-)Komponenten
 - Eine benötigte Schnittstelle spezifiziert, welche Partnerklasse die Klasse zum Ausführen benötigt (wird in separatem Compartment notiert)
 - Java: wird *nicht spezifiziert*, sondern vom Übersetzer herausgefunden
 - I.d.R. Sind UML-Komponenten komplexe Objekte mit vielen Unterobjekten
 - Komponenten sind einfacher wiederzuverwenden, da alle Aufrufabhängigkeiten explizit sind
 - Das **Kontextmodell** einer Komponente besteht aus der Menge ihrer angebotenen und benötigten Schnittstellen



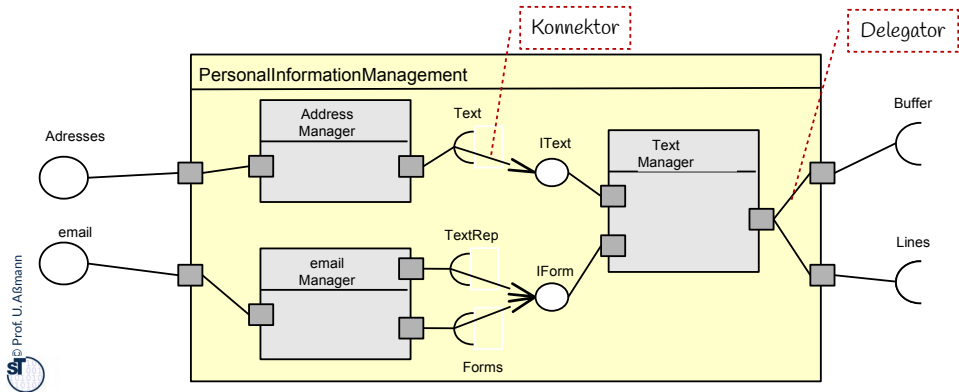
Anschlüsse (Ports) gruppieren Schnittstellen

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



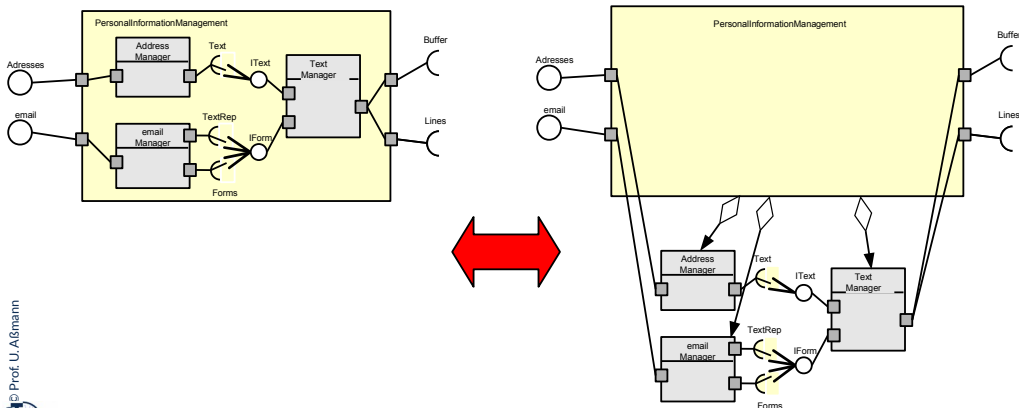
Schachtelung von Klassen zu UML-Komponenten

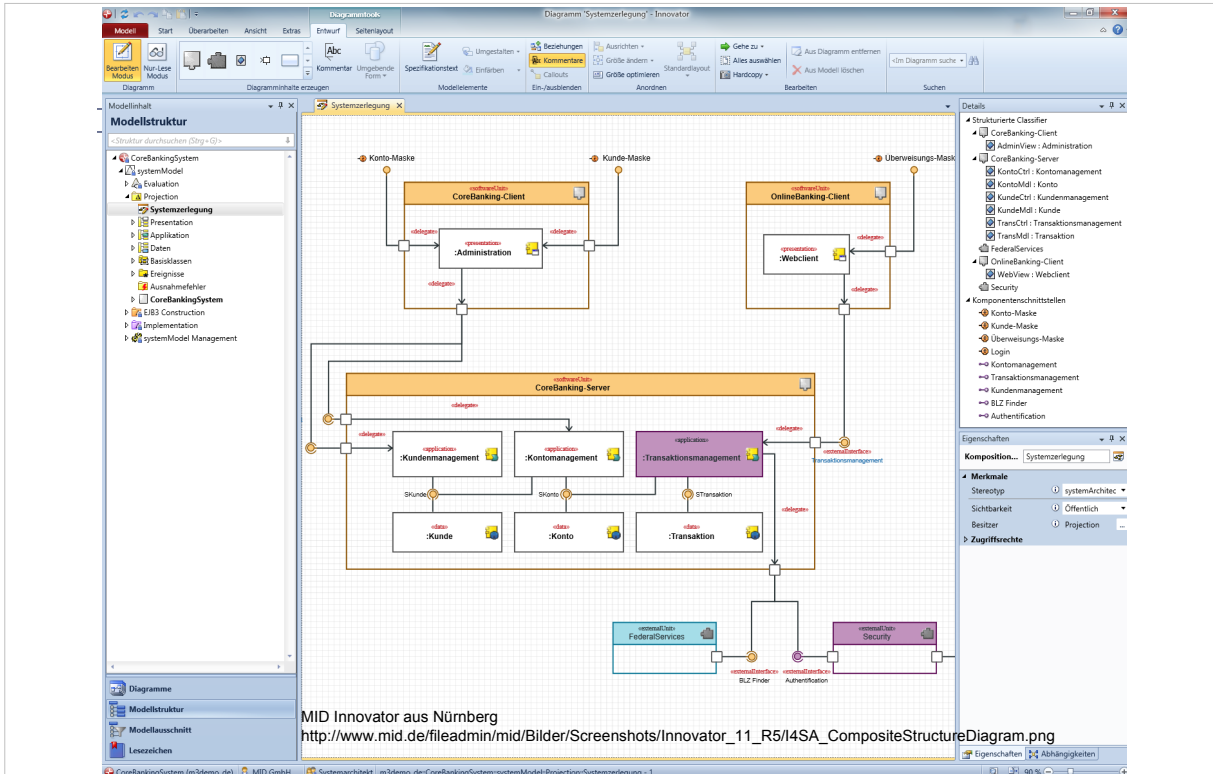
- ▶ Eine **UML-Komponente** ist eine Klasse mit angebotenen und benötigten Schnittstellen, mit Ports (Portklassen) und inneren Klassen
 - UML-Komponenten bilden hierarchische Klassen für hierarchische Objekte
 - Ports werden mit *Verbindern* (Links, einfachen Konnektoren) verbunden
 - *Delegator*: Link 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

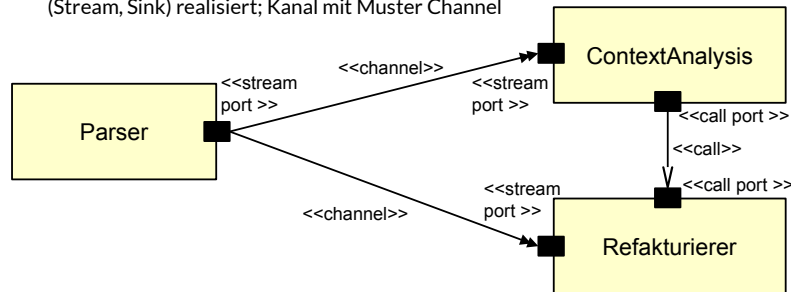
- ▶ UML-Komponenten sind komplexe Großobjekte, hierarchische Klassen, die ihre Unterkomponenten aggregieren und ihre Sichtbarkeit begrenzen
 - Eine Komponente ist gleichzeitig ein *Paket* und eine *Facade* für alle Unterkomponenten
 - I.d.R. wird eine UML-Komponente mit einem Facade oder mehreren Bridge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



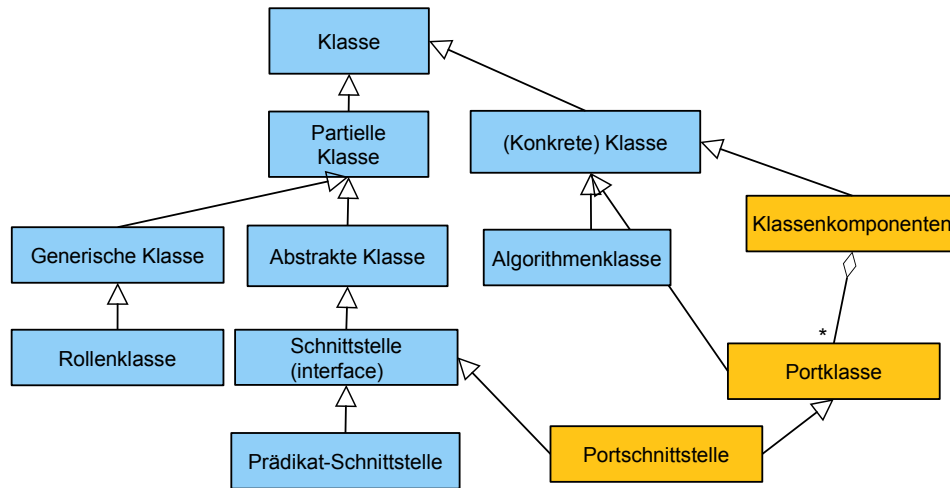


Strom-Anschlüsse von Komponentenklassen und Kanal-Konnektoren

- ▶ Ein **Aufrufanschluß (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 kontinuierlich 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 **Konnektoren (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)





33.2 Das Kontextmodell

- ▶ Neben der GUI (graphischen Oberfläche, interaktive Schnittstelle) beinhaltet das Kontextmodell alle Schnittstellen eines Systems.



Kontextmodell (Schnittstellenmodell)

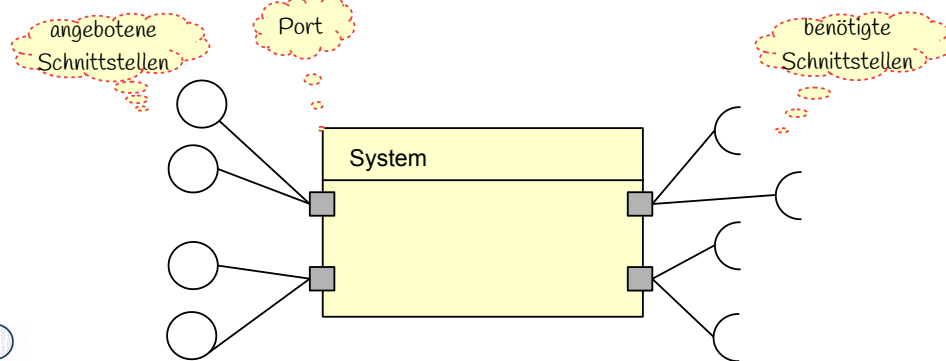
- ▶ Das System und seine obersten Schichten wird meist als *Klassenkomponente* verstanden
 - Mit angebotenen und benötigten Schnittstellen, mit ports
 - Hierarchisch durch die Ganz/Teile-Relation verfeinert
- ▶ Das **Kontextmodell (Schnittstellenmodell) eines Systems** enthält alle äusseren Schnittstellen 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

Das System wird oft als eine große komplexe Komponente gesehen, dessen Schnittstellen vom Kontextmodell beschrieben werden.



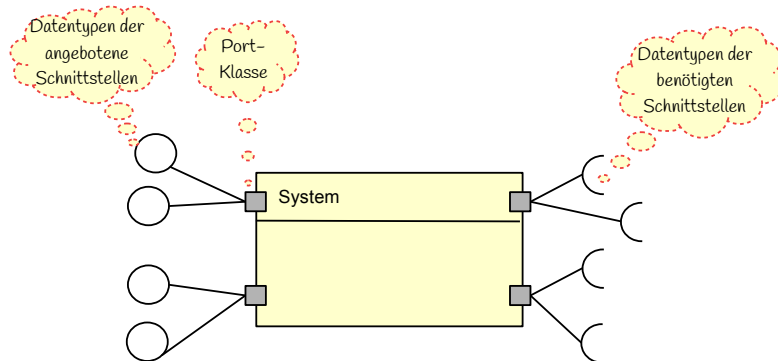
UML-Komponenten im Kontextmodell

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

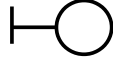



Datentypen des Kontextmodells stammen aus dem Domänenmodell


- ▶ 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

- ▶ **Schnittstellen-Klassen** (Portschnittstellen, Portklassen, GUI-Klassen, Grenzklassen, boundary classes)
 - Teil einer Schnittstelle oder Benutzerschnittstelle

<<boundary>>
 - ▶ **Steuerungsklassen** (control classes)
 - Aktive Klasse der Anwendungslogik, die die Ausführung eines Prozesses steuert
 - Mit oder ohne Zustand

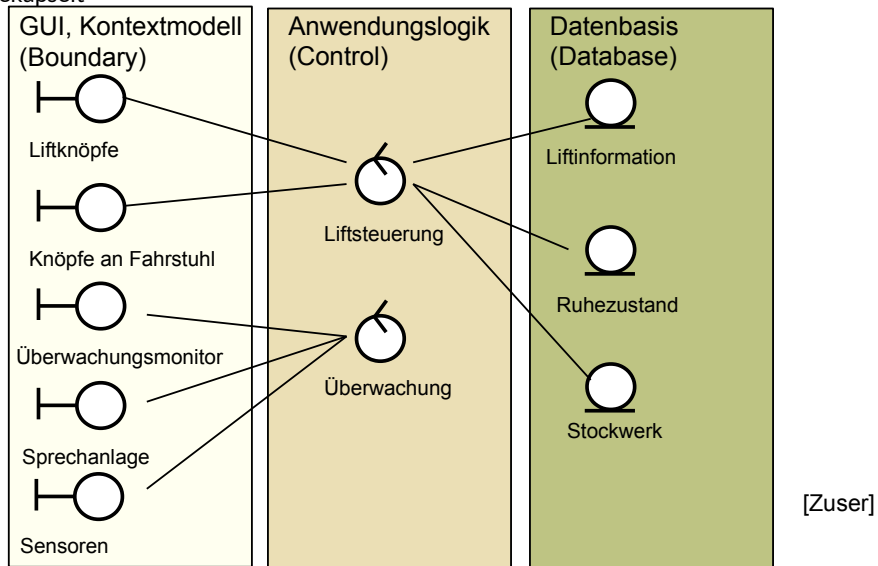
<<control>>
<<workflow>>
<<tool>>
<<process>>
 - ▶ **Material:** Passive Klasse, beschreibt Daten
 - Entitätsklassen (Entity): Beschreibt komposite, persistente Datenobjekte der Domäne
 - Datenklasse (Database): Adapterklasse für eine Entity in der Datenbank
 - Oft sind Entity and Database vereinigt
 - MaterialContainer: Container für Material

<<material>>
<<entity>>
<<database>>
 - ▶ BCD-Architektur: 3-Schichten-Architektur (3-tier architecture)
 - ▶ BCED-Architektur: 4-Schichten-Architektur (4-tier architecture)
- 
- <<container>>



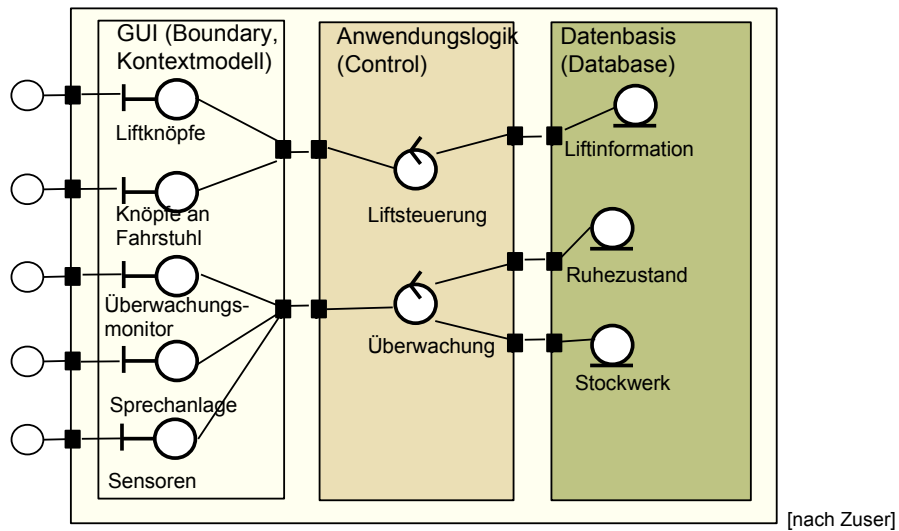
Bsp: Analyse-Klassenmodell Liftsteuerung im BCD-Stil

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



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

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





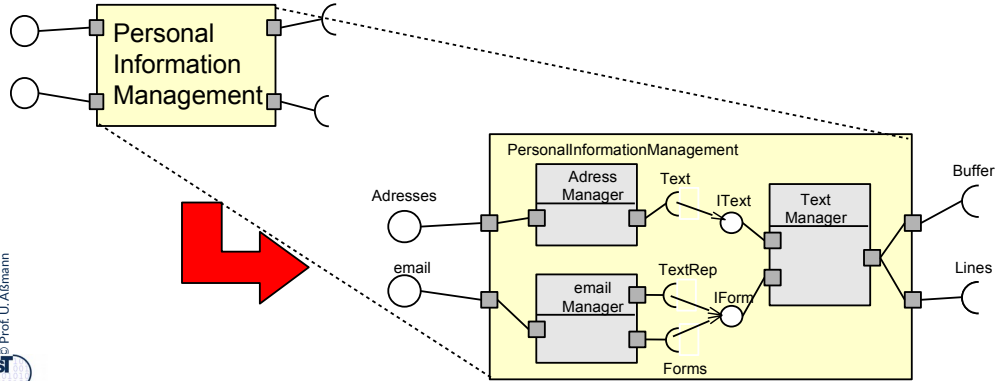
33.3 Top-Level-Architektur

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



Top-Level-Architektur

- ▶ 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 Komponenteklassen werden sichtbar



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

- ▶ 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



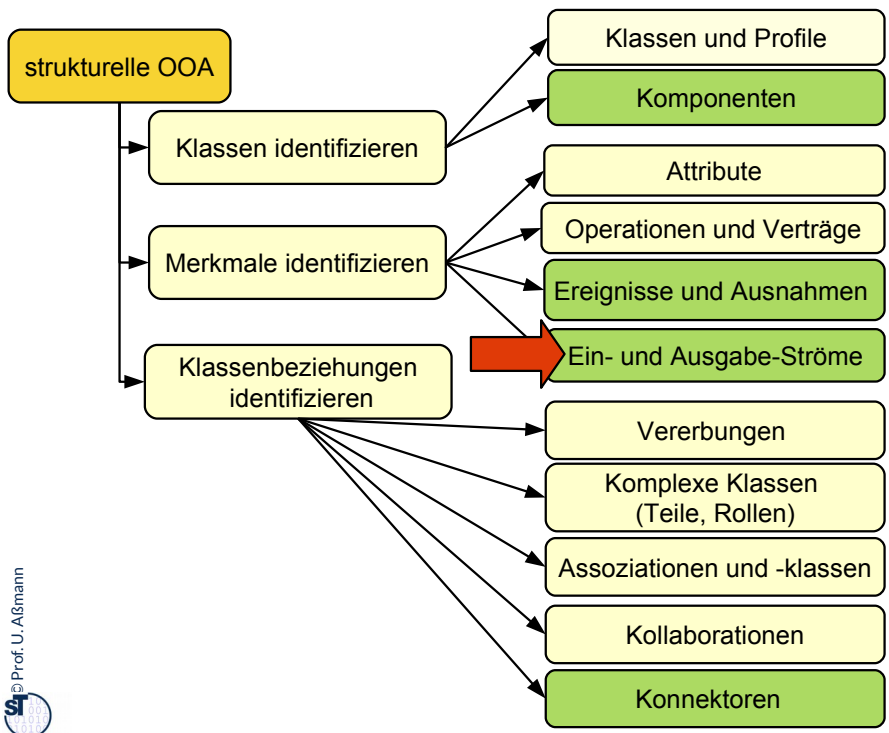
33.3 Asynchrone Systemmerkmale im Kontextmodell

Ereignisse und Ausnahmen (Exceptions)



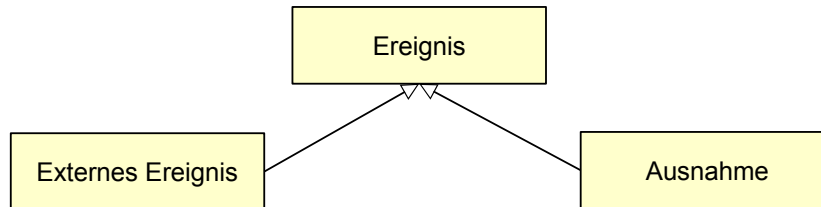
Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

► gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



Asynchrones Verhalten des Systems

- ▶ 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

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

- Systemfehler:

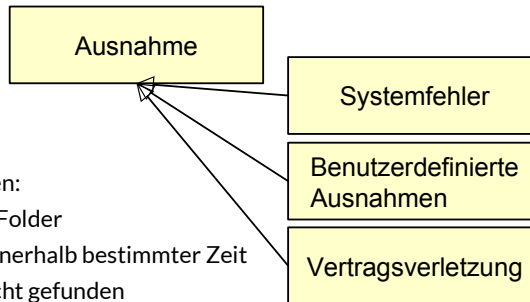
- Division durch 0
- Zugriff durch null-Zeiger
- Platte voll

- 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

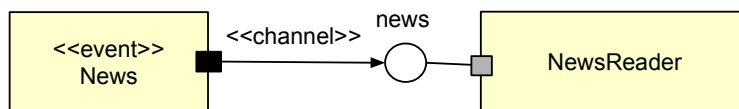
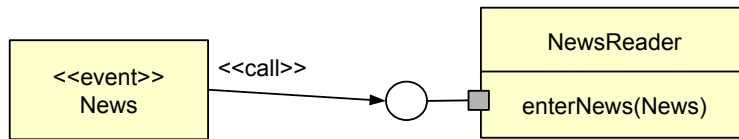


- ▶ Ausnahmen des Systems und die Reaktion des Kontextes auf diese gehören ins Kontextmodell!
- ▶ Implementierung: Ausnahmen sind als Konzept in Java vorhanden (exception)



Externe Ereignisse gelangen ins System über das Kontextmodell

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



Unterschied Domänen- vs. Kontextmodell

Domänen-Modell .- Das Abbild der realen Welt	Kontext-Modell - Teile der Technik, die der Kunde sehen muss
<p>Notation: UML</p> <p>Objekte: Fachgegenstände</p> <p>Klassen: Fachbegriffe</p> <p>Attribute ohne Typen</p> <p>Operationen: ohne Typen, Parameter und Rückgabewerte</p> <p>Assoziationen: partiell, bidirektional i. Allg. ohne Datentypen</p> <p>Aggregationen, Kompositionen</p> <p>Leserichtung, partielle Multiplizitäten</p> <p>Vererbung: Begriffsstruktur</p> <p>Annahme perfekter Technologie</p> <p>Funktionale Essenz</p> <p>Grobe Strukturskizze</p>	<p>Notation: UML</p> <p>Objekte: Softwareeinheiten</p> <p>Klassen: Komponenten, Grenzklassen, Ports, Interface, Stereotypen aus Profilen</p> <p>Attribute: Klassenattribute, Ports, Ströme</p> <p>Unidirektionale Assoziationen mit voller Multiplizität, Navigation</p> <p>Vererbung: Programmableitung</p> <p>Asynchrones Verhalten: Ereignisse, Ausnahmen</p> <p>Genauere Strukturdefinition in der Top-Level-Architektur: Adapter, Konnektoren</p>

Hussmann

The End – Anhang mit optionalem Material

- ▶ 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?
- ▶ Warum ist ein System ein komplexes Großobjekt? Wie gliedert es sich?
- ▶ Erkläre den Zusammenhang zwischen Anschlüssen, Konnektoren und Ereignissen.





33.A.1 Einsatz von Adapter zum Brücken von Schnittstellen in der Top-Level-Architektur

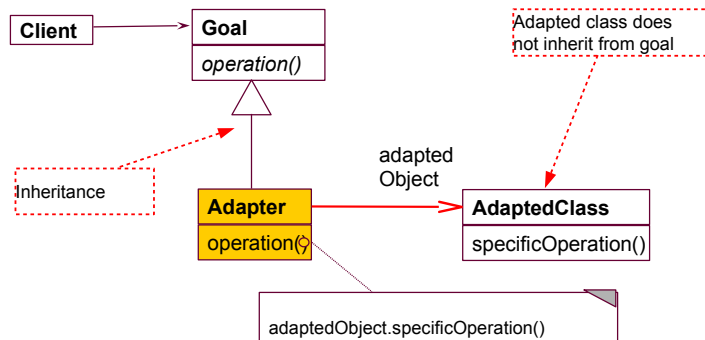


Aufgabe des Kontextmodells: Verbindung nach außen

- ▶ 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

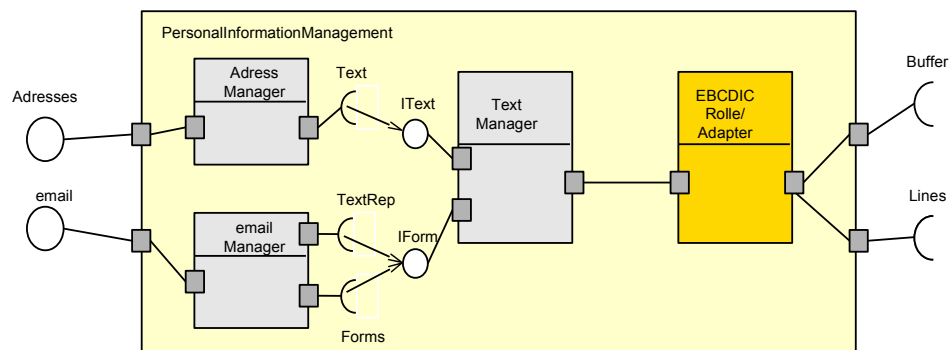
Entwurfsmuster Adapter (Objektadapter) (Wdh.)

- ▶ 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



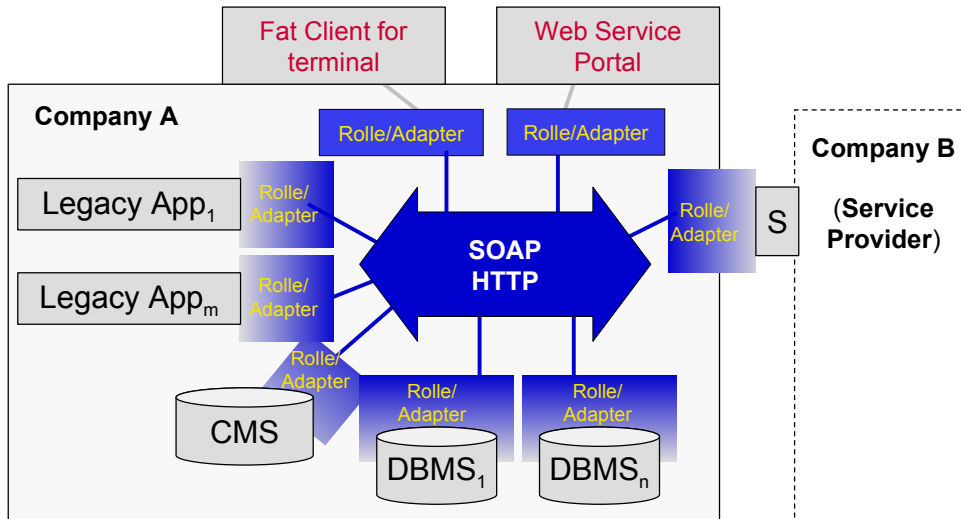
Top-Level-Architektur mit Adaptern

- ▶ 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 Entwurfsmuster Konnektor (SOAP/HTTP)

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



Profil "Aktive und Passive Klassen"

- ▶ 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) `<<active class>>` `<<process>>`
 - Werkzeuge (Tools) `<<tool>>`
- ▶ 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 `<<channel>>`
mit Senken geschrieben und mit Streams `<<call>>`
gelesen `<<stream>>`

Manager



Beispiel: Bahrami-Profil für Domänenmodell

- ▶ 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 <<event>>

- ▶ Ereignisklasse (Event) <<organization>>
 - Ereignis in der Zeit, extern zum Objekt

- ▶ Organisation <<people>>
 - Verkörpert Wissen über eine Organisationseinheit

- ▶ Menschen (People) <<place>>

- ▶ Plätze (Places class)



Beispiel: Rumbaugh-Profil

- ▶ 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>>



33.A.2 Weitere Arten von Schnittstellen und Klassen

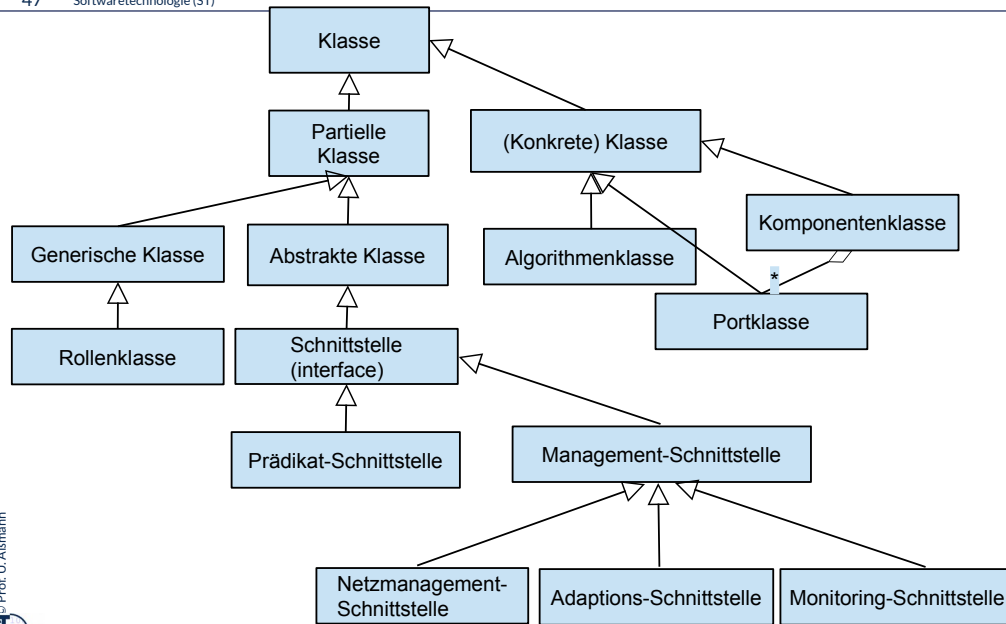
(zur Information)



Weitere Arten von Schnittstellen in komplexen Objekten

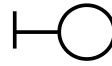
- ▶ **Funktionale Schnittstellen** enthalten Funktionen, die direkt auf den Zustand des Objekts zugreifen
 - **Prädikat-Schnittstellen**, die Prädikate auswerten
- ▶ **Managementschnittstellen** enthalten Funktionen, die das Objekt und seine Nachbarn verwalten:
 - **Netzmanagement-Funktionen** verändern 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)




Identifikation von Systemschnittstellenklassen für das Kontextmodell

- ▶ **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

- ▶ **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

