32. Komplexe Objekte

Prof. Dr. rer. nat. habil. Uwe Aßman
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 13-1.2, 14.06.13

1) Modellierung von Hierarchien
2) UML-Komponenten (Hierarchische Objekte mit angebotenen und benötigten Schnittstellen)
3) Modellierung von komplexen Objekten
    1) Komplexe Objekte
    2) Objektanreicherung
    3) Natürliche Typen und Rollen
    4) Private und essentielle Teile
4) Anhang
    1) Facetten
    2) Phasen

Überblick Teil III: Objektorientierte Analyse (OOA)

1. Überblick Objektorientierte Analyse
    1. (schon gehabt:) Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodellgetriebene Modellierung mit UML für das Domänenmodell
    1. Strukturelle metamodellgetriebene Modellierung
    2. Modellierung von komplexen Objekten
        1. Modellierung von Hierarchien
        2. Modellierung von Komponenten mit angebotenen und benötigten Schnittstellen
        3. Modellierung von komplexen Objekten und ihren Unterobjekten
    3. Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen
    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

Obligatorische Literatur

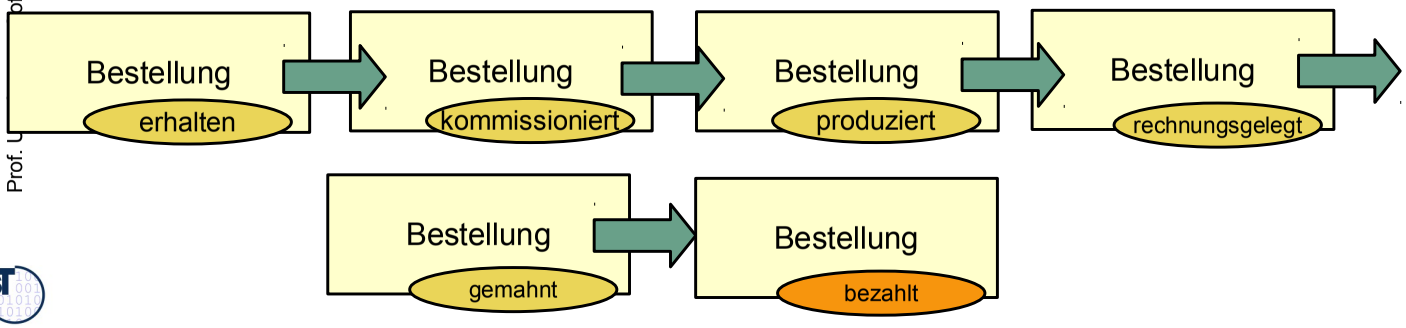
- Störrle 5.3, 5.4
- Weitere Literatur:
  - L. Maciaszek. Requirements Analysis and System Design – Developing Information Systems with UML. Addison-Wesley.
  - Giancarlo W. Guizzardi. Ontological foundations for structure conceptual models. PhD thesis, Twente University, Enschede, Netherlands, 2005.
  - Nicola Guarino Chris Welty. Supporting ontological analysis of taxonomic relationships. Data and Knowledge Engineering, 39:51-74, 2001.
  - Friedrich Steimann. On the representation of roles in object-oriented and conceptual modelling. Data Knowl. Eng, 35(1):83-106, 2000.

Obligatorische Literatur

- Störrle 5.3, 5.4
- Weitere Literatur:
  - L. Maciaszek. Requirements Analysis and System Design – Developing Information Systems with UML. Addison-Wesley.
  - Giancarlo W. Guizzardi. Ontological foundations for structure conceptual models. PhD thesis, Twente University, Enschede, Netherlands, 2005.
  - Nicola Guarino Chris Welty. Supporting ontological analysis of taxonomic relationships. Data and Knowledge Engineering, 39:51-74, 2001.
  - Friedrich Steimann. On the representation of roles in object-oriented and conceptual modelling. Data Knowl. Eng, 35(1):83-106, 2000.

Geschäftsobjekte (Business Objects)

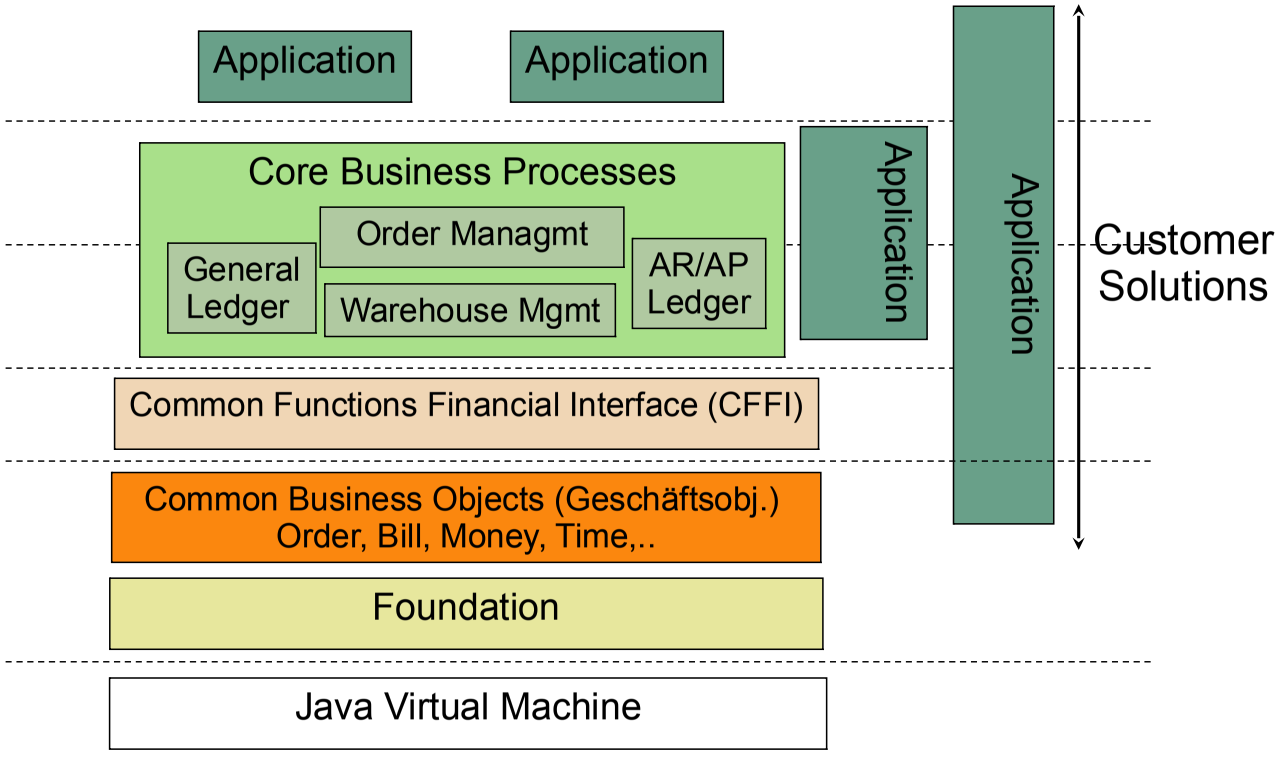
- In großen objektorientierten Frameworks werden die Objekte sehr komplex
- `Bestellung` als Beispiel:
  - eine Bestellung ist ein langlebiges, persistentes komplexes Objekt, das von dem
    - Auftragseingang des Kundens an durch die
    - Produktion, Kommissionierung, Rechnungserstellung,
    - Auslieferung und Mahnwesen
  - erhalten bleiben muss
- Dynamische Erweiterbarkeit nötig:
  - Durchläuft verschiedene verschiedene Phasen, spielt viele verschiedene Rollen
  - Enthält viele Teile, und wird in neuen Phasen ständig mit neuen Teilen, Attributen und Methoden versehen
  - Verhalten muss an die Phase adaptiert werden



Architektur von IBM San Francisco

Java-Framework für Geschäftsanwendungen (ERP)

- P. Monday, J. Carey, M. Dangler. SanFrancisco Component Framework: an introduction. Addison-Wesley, 2000.



Beispiel: Geschäftsobjekte in der Bibliothek

- Wertobjekte:
  - Adresse, Währung, Kalender

Allgemeine Geschäftsobjekte:

- Firma
- Geschäftspartner
- Kunde (und Person)
- Zahlen mit beliebiger genauigkeit mit Nachkommastellen
- Fiskalische Kalender
- Bezahlmethode
- Maßeinheit

- Finanzielle Geschäftsobjekte
  - Geld
  - Währungsgewinn
  - Konto
  - Verlustkonto
- Allgemeine Mechanismen für Geschäftssoftware:
  - Buchführungskonten
  - Klassifikationen
  - Schlüsselobjekte

Problem

- Start des Projekts: 1995
- IBM stellte 1999 sein Framework wieder ein
- Technische Gründe liegen in Java:
  - Einfache Vererbung von Java erschwert die Wiederverwendung von Code
  - Phasen und Rollen können nicht einfach abgebildet werden. Dynamische Anpassung ist zu komplex
  - In Wirklichkeit hat Java Probleme, komplexe dynamische Objekte zu beschreiben. Es braucht dazu Entwurfsmuster, aber auch Mechanismen für große Objekte
- Das Folgende stellt Modellierungsmethoden für komplexe Objekte vor

Ein **komplexes Objekt (Subjekt, big object)** ist ein Objekt,
das auf Programmierniveau
wegen seiner Komplexität durch mehrere Objekte dargestellt wird.

Seine innere Struktur ist meist hierarchisch, immer aber azyklisch angelegt.

- Große Objekte leben oft parallel zueinander
  - kommunzieren oft über Ströme, Senken, Kanäle

32.1 Modellierung von Hierarchien und komplexen Objekten (Taxonomien und Teilehierarchien)

Bitte hier nochmal Kap. "Strukturelle Analyse->Aggregation lesen

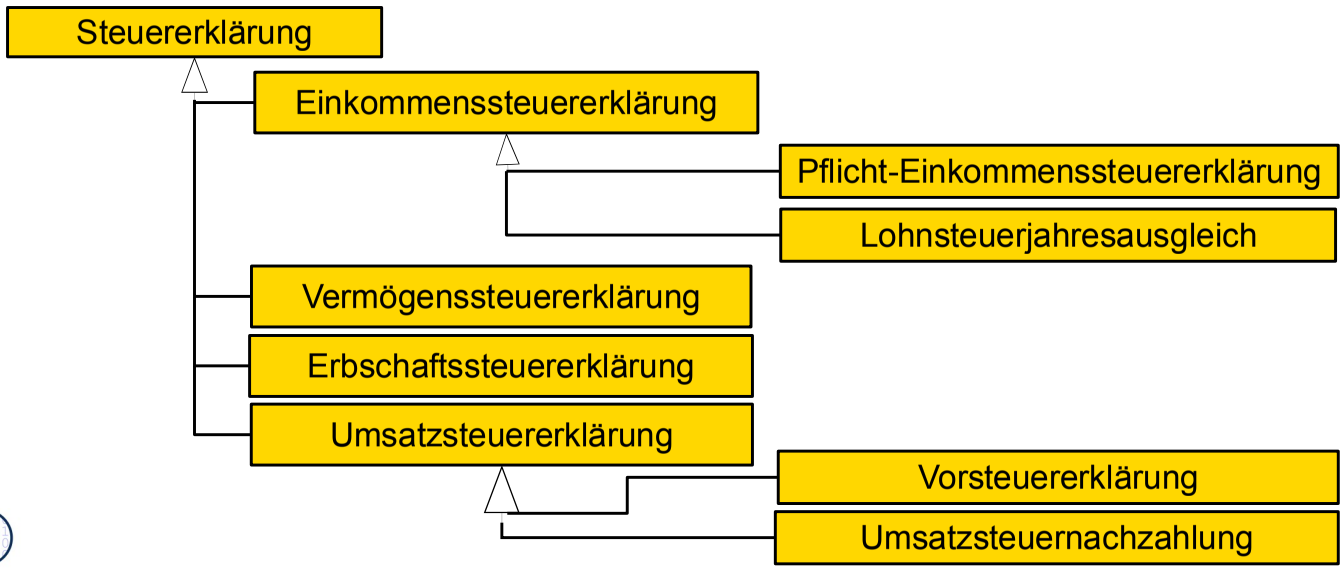
Darstellung von Hierarchien und kompositen Objekten

- Hierarchien und komposite Objekte können mit visuellen Darstellungen für Bäume dargestellt werden
  - Baum
  - Venn-Diagramm
  - Zeilenhierarchie (Tree-Widget)
  - Mind map
- Alle Darstellungen sind äquivalent und können ineinander umgewandelt werden

Als Baumrelation können verschiedene Relationen dienen: Vererbung, Teil, Aufruf, Operatoren, Unterobjekte (s. später)

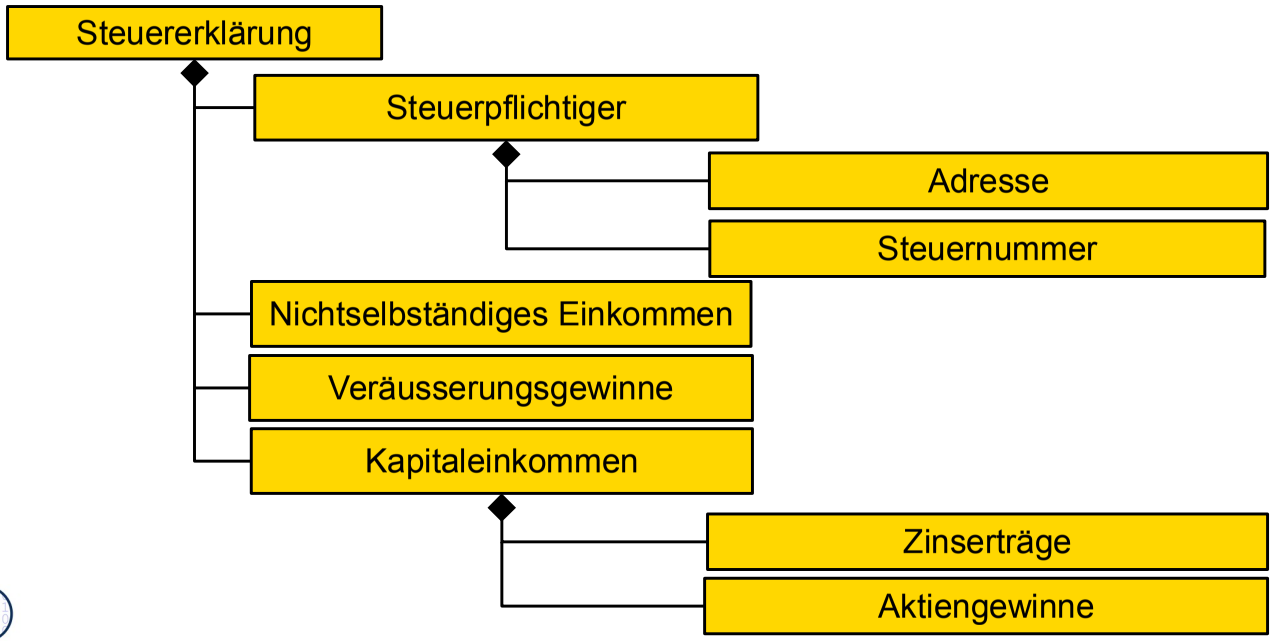
Zeilenhierarchien

- Hierarchien kann man als auch als *Zeilenhierarchien (horizontal dekomponierte Textbäume)* anordnen, analog zu einem *tree widget*
  - Teile können leicht zu- und aufgeklappt werden
  - Textuelle Annotationen können einfach in den Zeilen hinzugefügt werden
- Am einfachsten sieht man das an einer **Begriffshierarchie (Taxonomie)**
  - Ontologieeditoren benutzen dieses Format (z.B. Protege)
  - Zum Lernen verwendet man Begriffshierarchien [Wolf, 2.1]



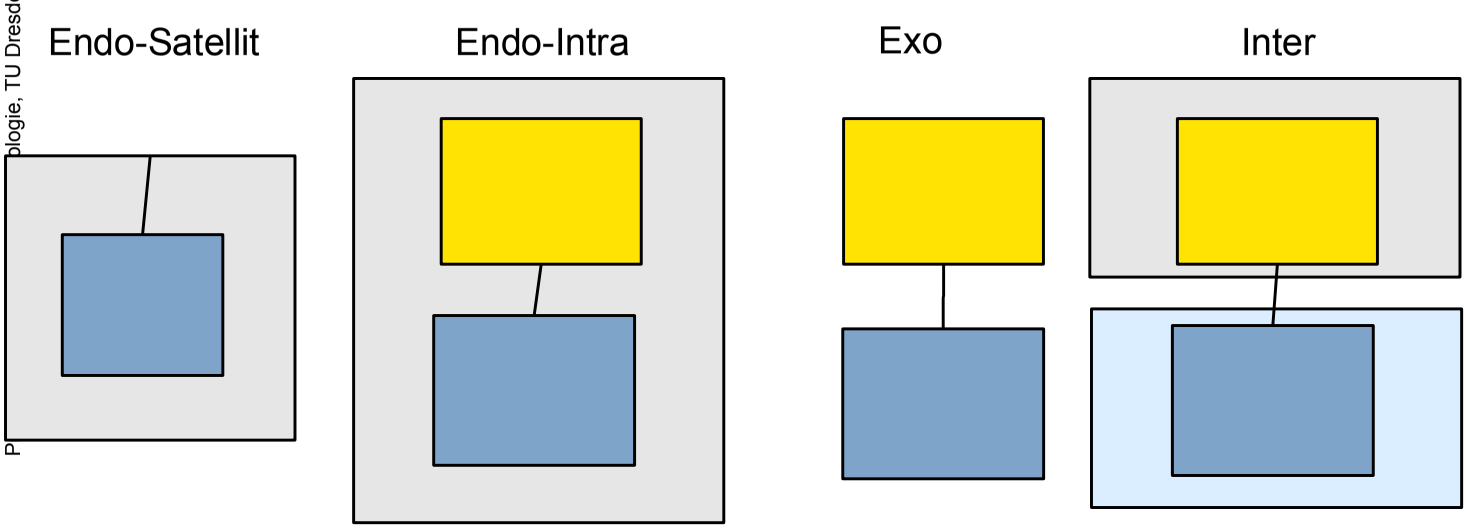
Ganzes/Teile-Zeilenhierarchie (whole-part hierarchies)

- Komplexe Objekte bestehen oft aus Ganz-Teile-Hierarchien (owns-a-Hierarchien)
- Man kann sie als auch als Zeilenhierarchien (Textbäume) anordnen (In UML: *Kompositionshierarchie*)
  - Ganzes/Teile-Hierarchien können mit Klassen oder auch Objekten gezeichnet werden



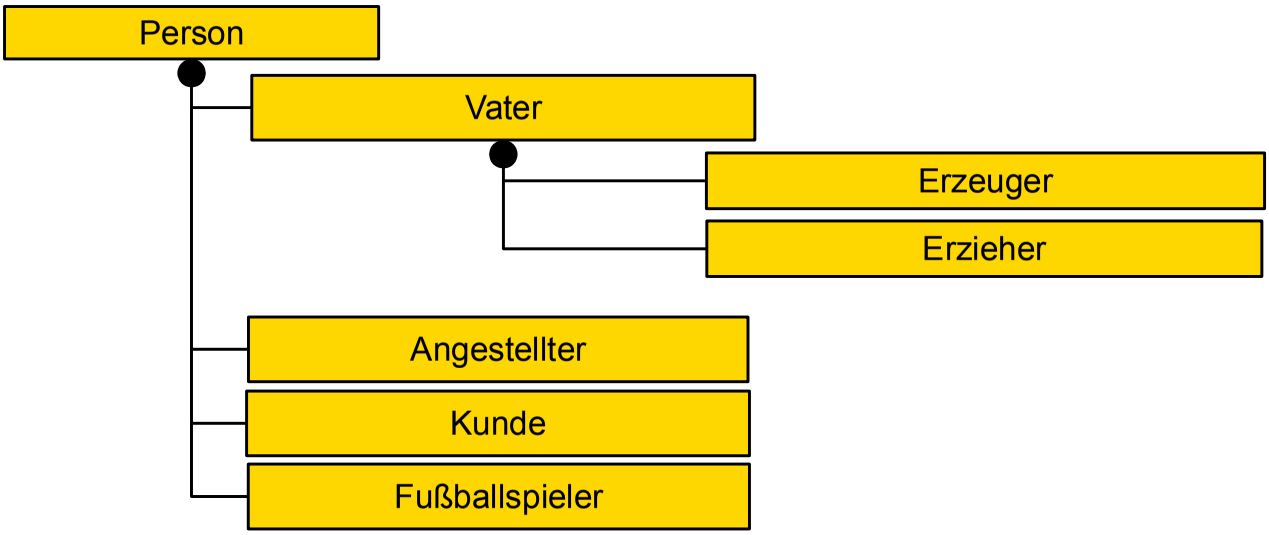
Endo vs. Exo-Assoziation

- Komplexe Objekte verwenden Assoziationen innerhalb und außerhalb.
- **Endo-Assoziation**: Assoziation innerhalb eines komplexen Objektes
  - **Intra-Assoziation:** beide werden von einem dritten umschlossen
  - **Satellit-Assoziation:** einer der Partner umschließt den anderen.
    - "Teil" ist eine Satellit-Endo-Assoziation vom Ganzen zum Teil
- **Exo-Assoziation:** keiner der Partner umschließt den anderen.
  - **Inter-Assoziation:** jeder der Partner wird von einem anderen umschlossen



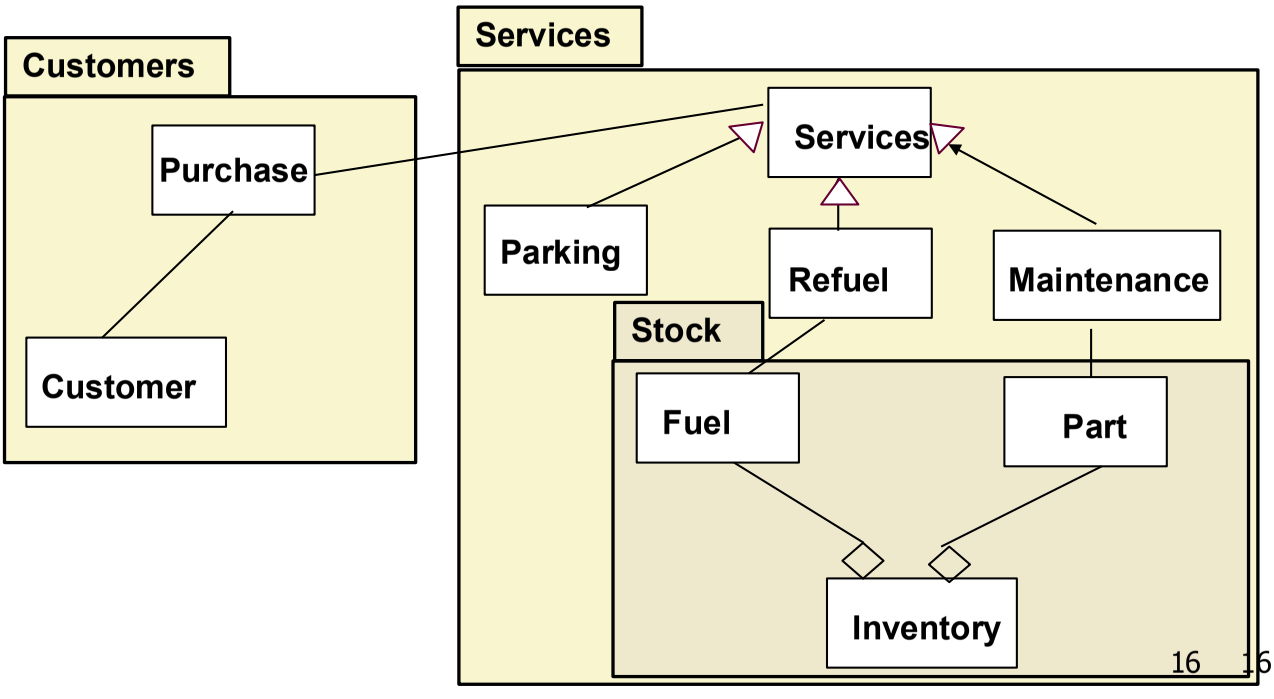
Eigenschaftshierarchie

- Auch integrates-a-Hierarchien kann man als auch als Zeilenhierarchien anordnen
- Integrates-a stellt Hierarchien von Prädikaten über ein großes Objekt dar
- Eine Integrationshierarchie verwendet die Endo-Assoziation integrates-a



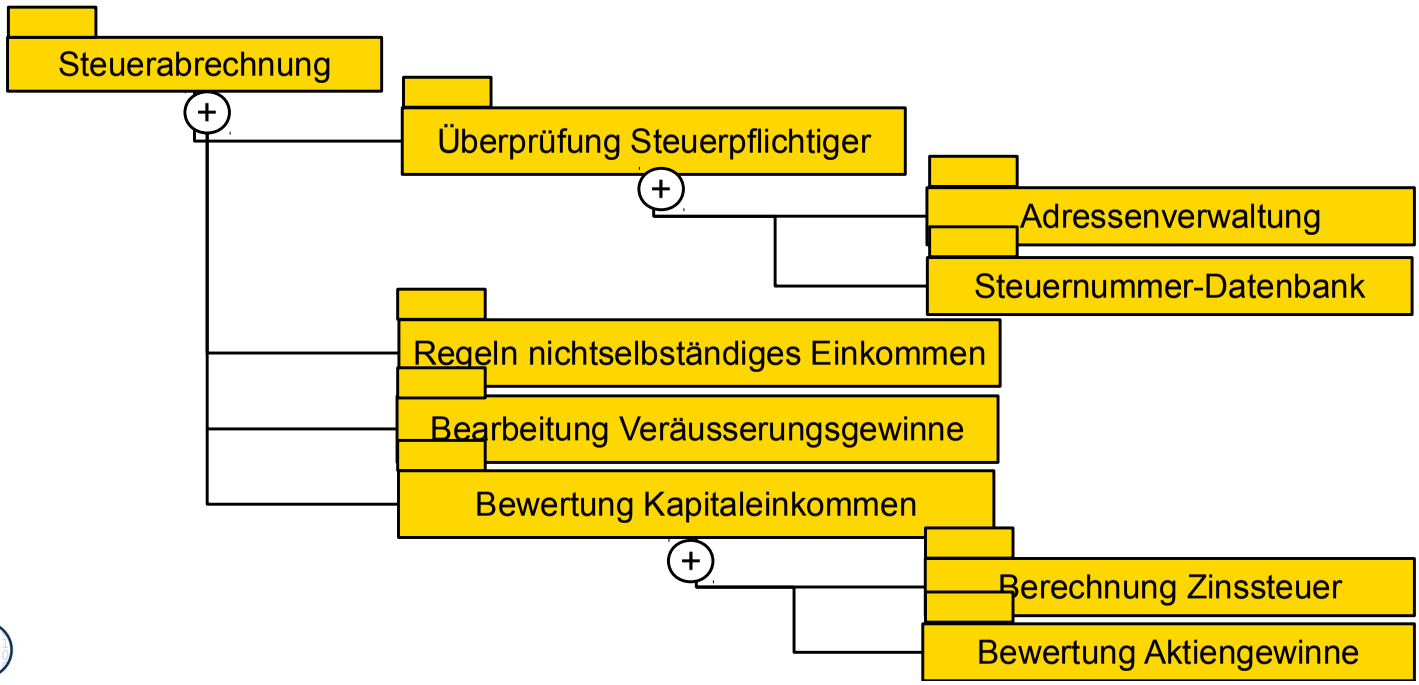
Bsp: Pakete und ihre Relationen in UML

- Klassen sind *flach,* Pakete dagegen hierarchisch strukturierbar
  - Pakete gruppieren Klassen, Objekte, andere UML-Fragmente
- UML-Paketrelationen sind hierarchisch oder azyklisch
  - Sind einfach auf Paketkonzepte von Programmiersprachen abbildbar



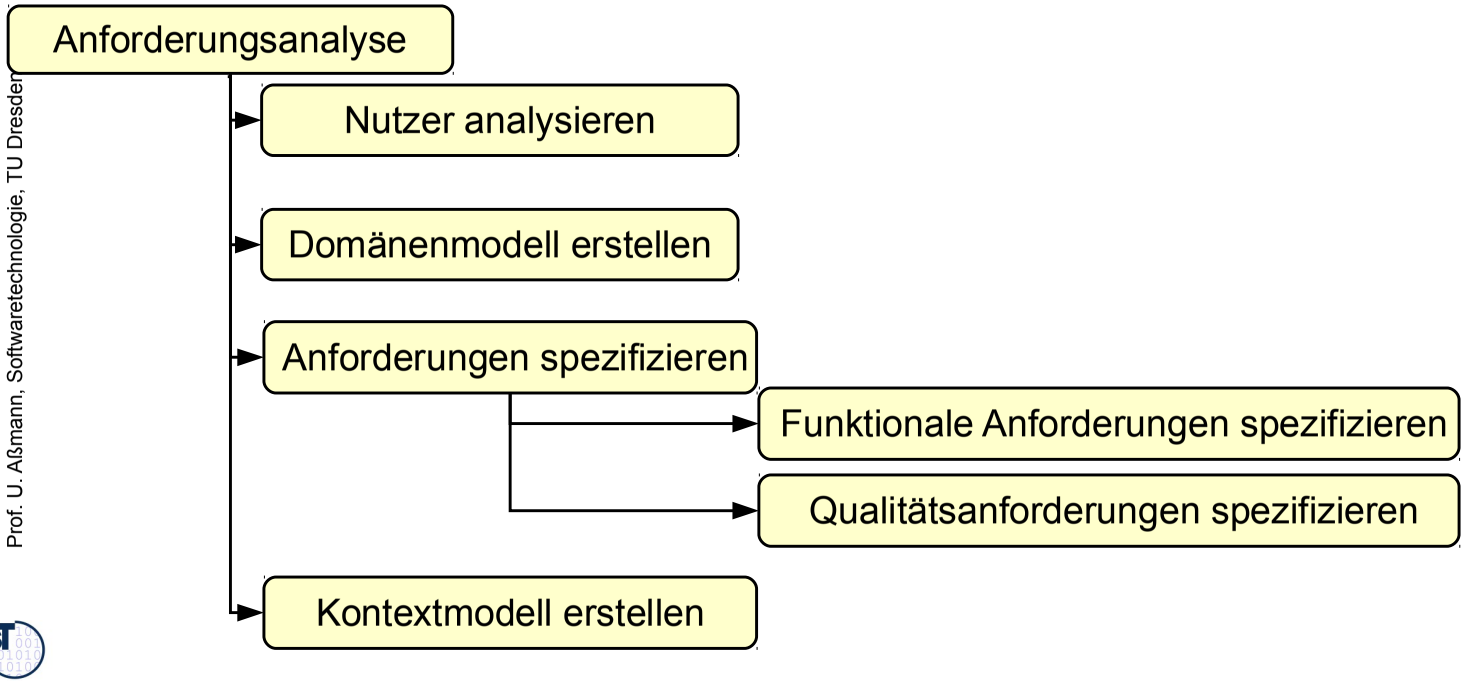
Paket-Zeilenhierarchie

- Pakete sind Container von UML-Fragmenten (z.B. von Klassen)
- Auch Paket-Hierarchien kann man als auch als Zeilenhierarchien (Textbäume) anordnen
- Operator ist die Paketvereinigung



Bsp. Aktivitätenhierarchie (Aktivitäts- oder Funktionsbaum)

- Aktivitäten werden in UML durch leicht abgerundete Rechtecke dargestellt
- Strukturierte Aktivitäten können in Zeilenhierarchien dargestellt werden

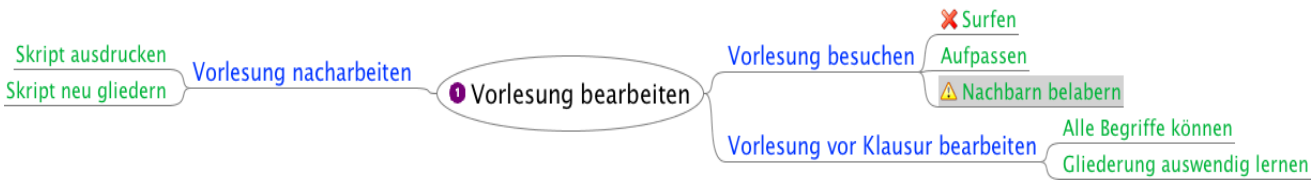


Exkurs: Lernen und Hierarchien

- **Zum Lernen verwendet man Begriffshierarchien** [Wolf, 2.1]
  - Einen Lernstoff arbeitet man von einer Quelle aus in eine Begriffshierarchie um
  - Aus einem Buch, aus einer Vorlesung, aus einer Diskussion oder Brainstorm
- Die Begriffshierarchie formt man so lange um, bis sie vollständig das Lerngebiet abdeckt und man sich abei allen Begriffen “wohlfühlt”
- Wohlfühlen heißt:
  - Man kann für jeden Begriff der Begriffshierarchie eine Definition geben
  - Man kann den Begriff anderen erklären
  - Man kann die Begriffe gegen verwandte Begriffe abgrenzen bzw. sie vergleichen
  - Man kann zu einem Begriff ein Problem schildern, bei dem der Begriff eine Rolle spielt
  - Man kann eine Aufgabe lösen, die mit dem Begriff zu tun hat
- Die Begriffshierarchie muss vollständig auswendig gelernt werden: sie muss zuerst in den Kopf, dann ins Herz wandern (siehe Bloomsche Taxonomie des Lernens)

Bsp.: Hierarchien mit Wissenskarten (Mind Maps)

- Mindmaps (Wissenskarten) sind Begriffshierarchien, die von der Mitte des Blattes her *monozentral* gezeichnet werden (“mapping your mind”)
- Sie können als Startpunkt beim Lernen eingesetzt werden
  - Untersuchen Sie, ob Sie mit Zeilenhierarchien oder Mindmaps besser zurecht kommen
  - Aufgabe: Suchen Sie mit Google nach freien Mindmap-Werkzeugen für den Computer Ihrer Wahl
  - Zeichnen Sie eine Mind map von diesem Kapitel

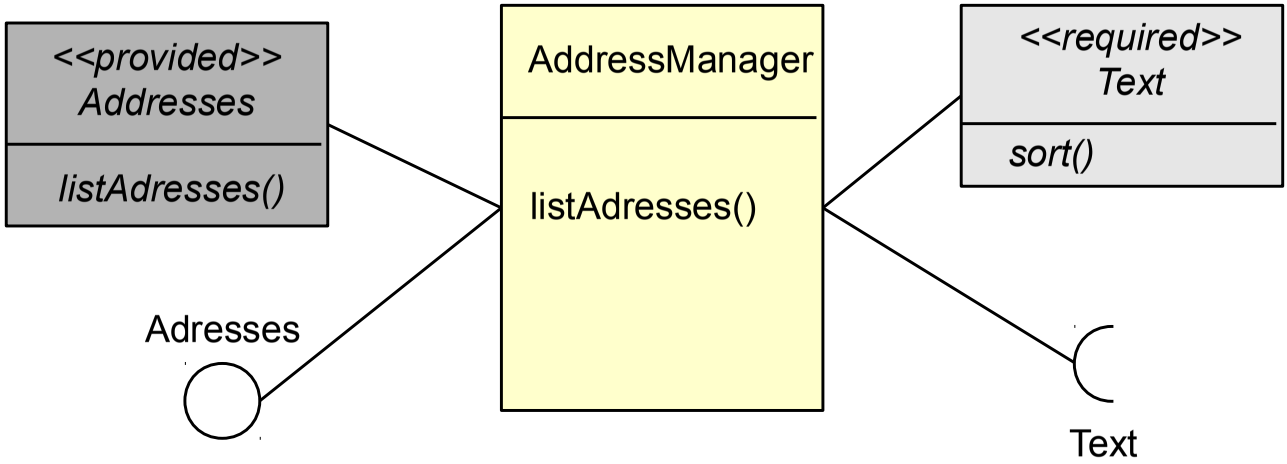


32.2 UML-Komponenten für Großobjekte (Hierarchische komplexe Klassen)

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

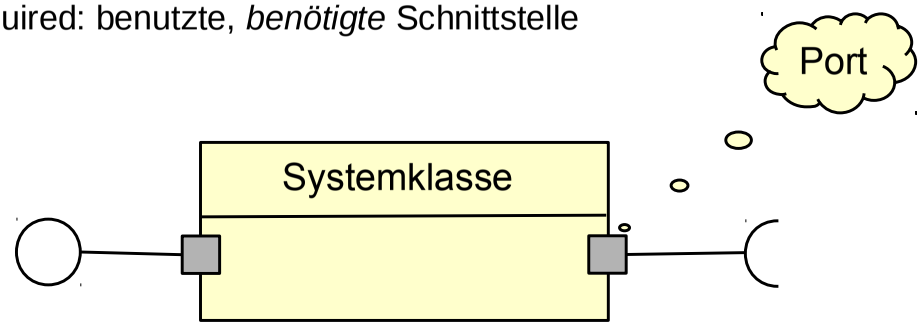
Lollipops und Plugs (Balls and Sockets)

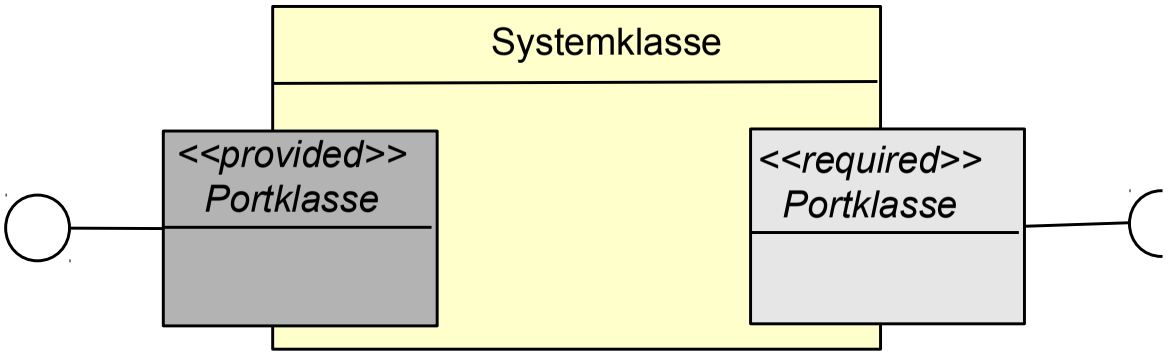
- Für eine Klasse kann man *angebotene (provided)* von *benötigter (required)* Schnittstelle unterscheiden
  - Eine benötigte Schnittstelle spezifiziert, welche Partnerklasse die Klasse zum Ausführen benötigt.
  - In Java wird die benötigte Schnittstelle *nicht spezifiziert,* sondern vom Übersetzer herausgefunden



Anschlüsse (Ports)

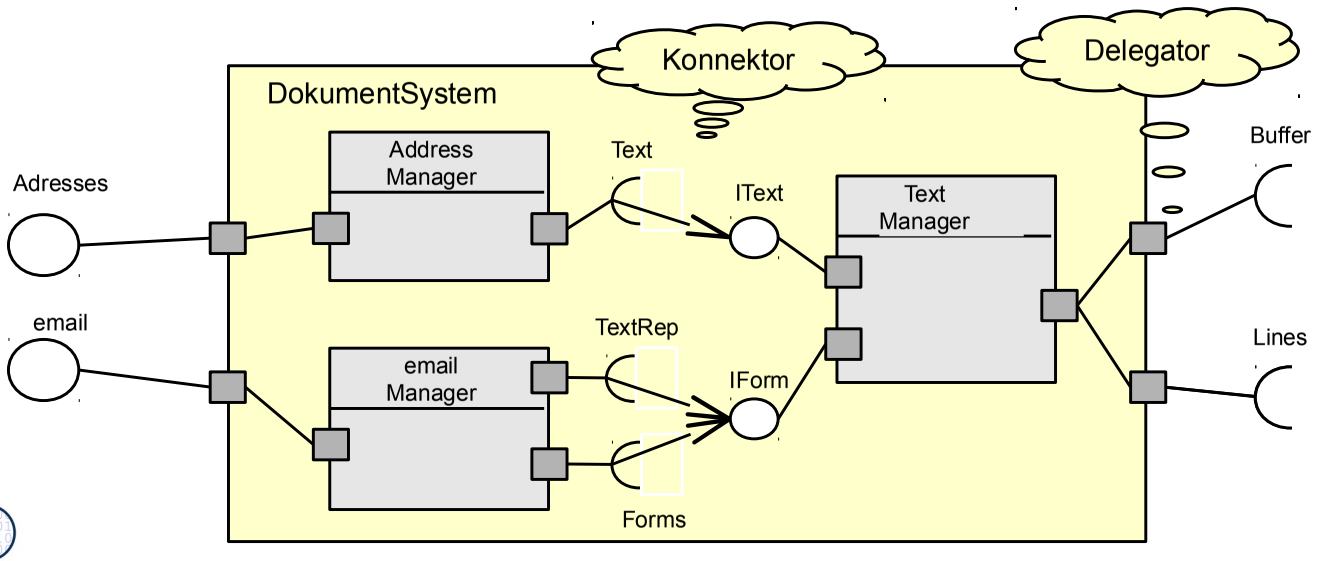
- Anschlüsse (ports) bestehen aus verkapselten (Port-)Klassen mit Schnittstellen
  - provided: normale, *angebotene* Schnittstelle
  - required: benutzte, *benötigte* Schnittstelle





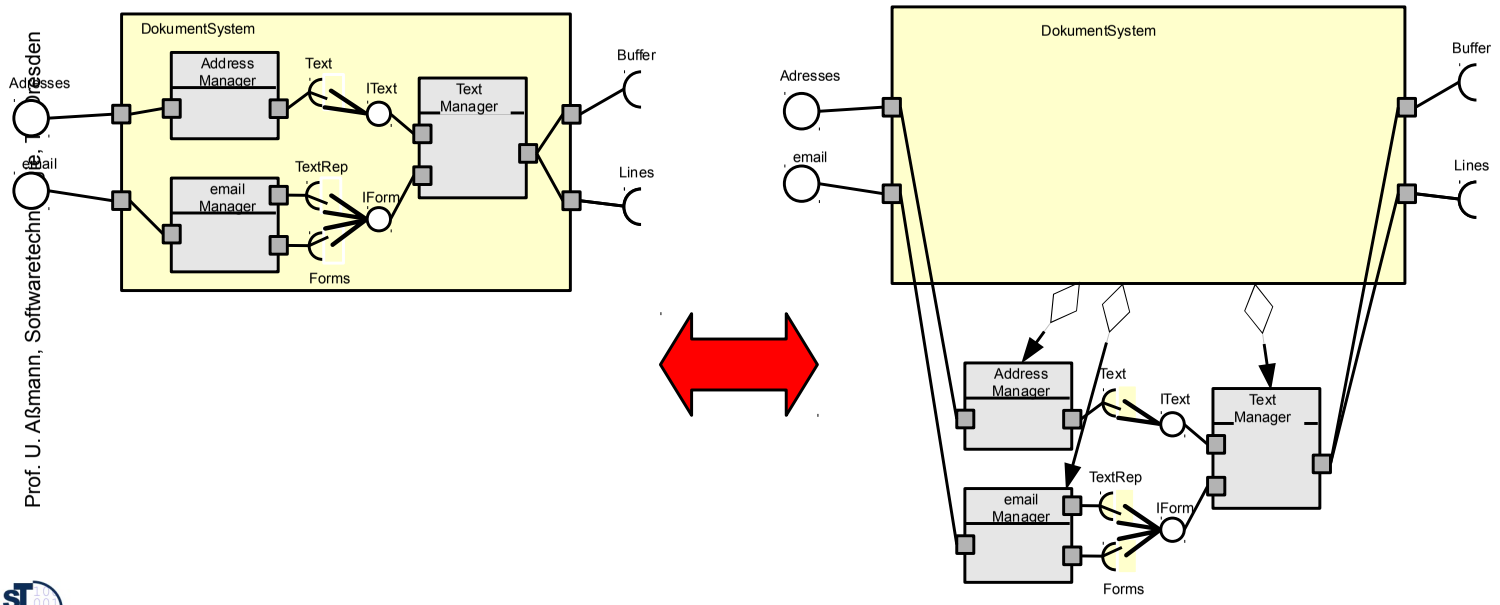
Schachtelung von Klassen zu UML-Komponenten

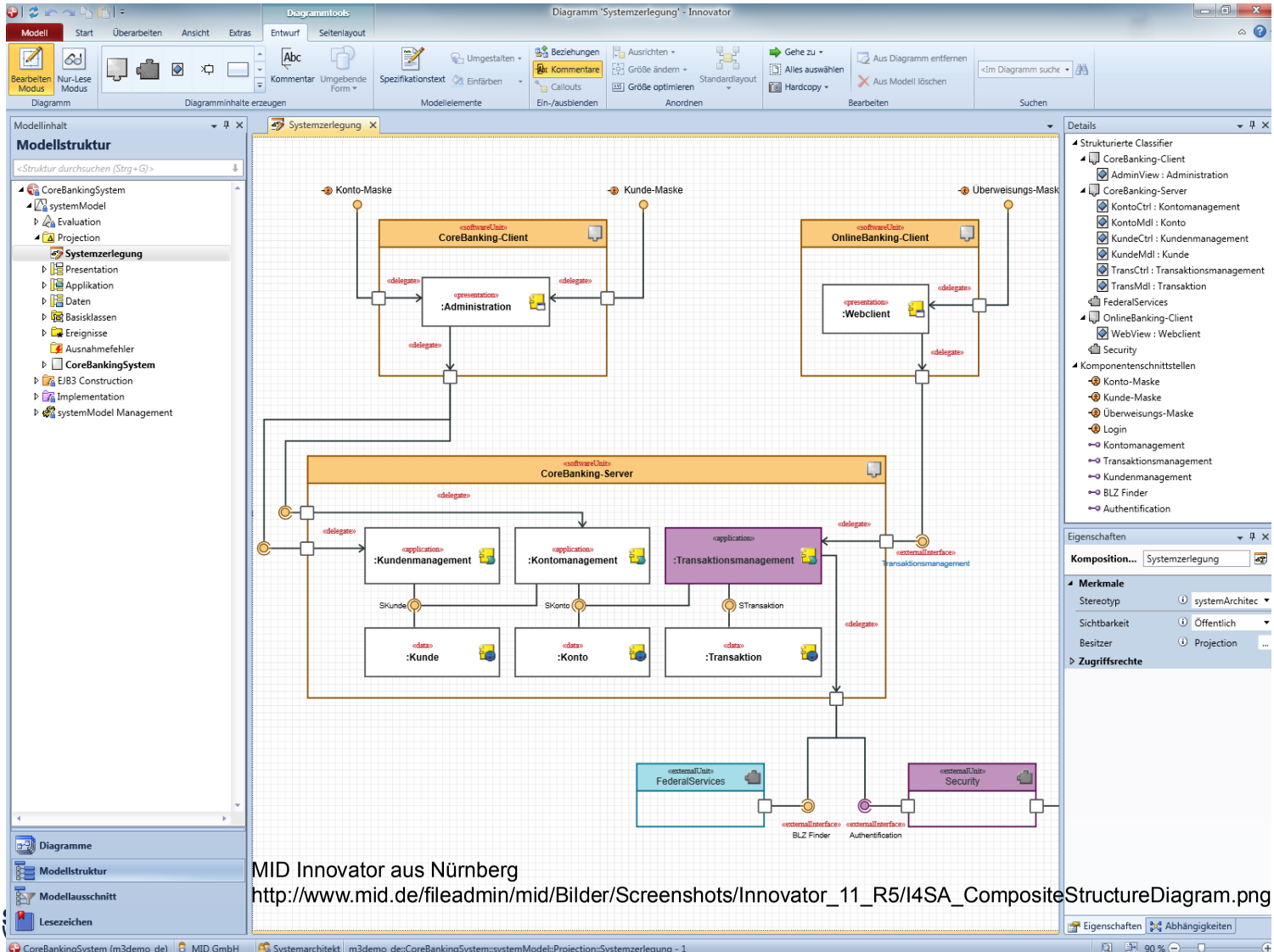
- Klassen mit angebotenen und benötigten Schnittstellen heißen (in UML) *Komponenten (Hierarchische Klassen für hierarchische Objekte)*
  - Ports werden mit *Verbindern (Links, einfachen Konnektoren)* verbunden
  - *Delegator*: Link von äußerem zu innerem Port
  - Achtung: jeder Port kann eine Schnittstelle oder einer zusätzlichen Klasse entsprechen
  - Facade Pattern: Komponente spielt eine Facade für die Unterkomponenten
- Komponenten sind hierarchisch schachtelbar



Schachtelung bedeutet Aggregation

- *Komponenten (Hierarch. Klassen/objekte)* aggregieren ihre Unterkomponenten und begrenzen ihre Sichbarkeit
  - Eine Komponente ist gleichzeitig ein *Paket* und eine *Facade* für alle Unterkomponenten

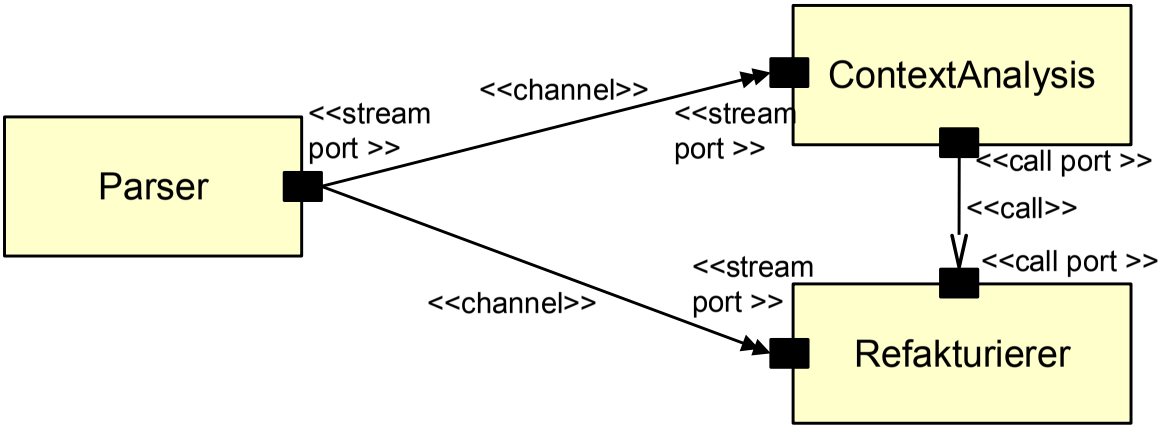




MID Innovator aus Nürnberg
http://www.mid.de/fileadmin/mid/Bilder/Screenshots/Innovator_11_R5/I4SA_CompositeStructureDiagram.png

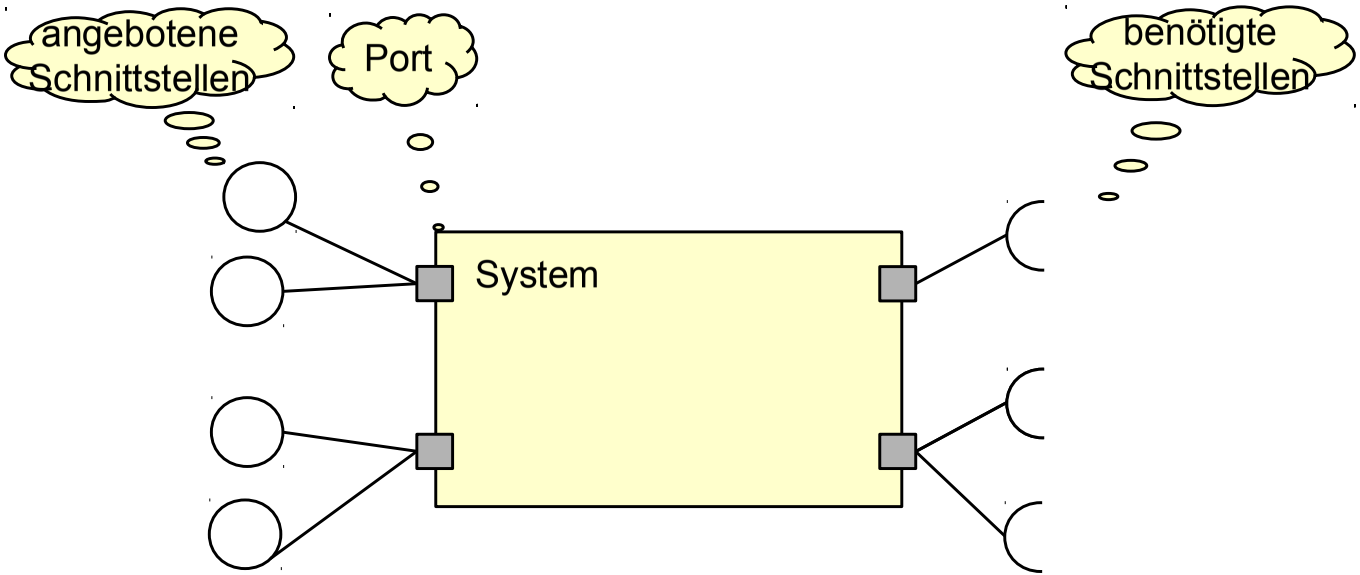
Strom-Anschlüsse und Ströme

- Ein *Aufrufsanschluß (call port)* ist eine angebotene oder benötigte Operation eines Objekts, über das es synchron aufgerufen wird oder synchron aufruft
- Ein *Stromanschluß (stream port)* ist ein Ein- oder Ausgabekanal eines Objekts, 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)
  - Strom-Ports werden durch *Strom-Kanäle (channels, pipes)* zu Strömen verbunden
  - Es entstehen Pipe-und-Filter-Architekturen (Datenflussarchitekturen)
- Entwurfs- und Implementierungsmodell: Stromkonzept wird mit Muster Iterator realisiert



Klassen-Komponenten im Kontextmodell

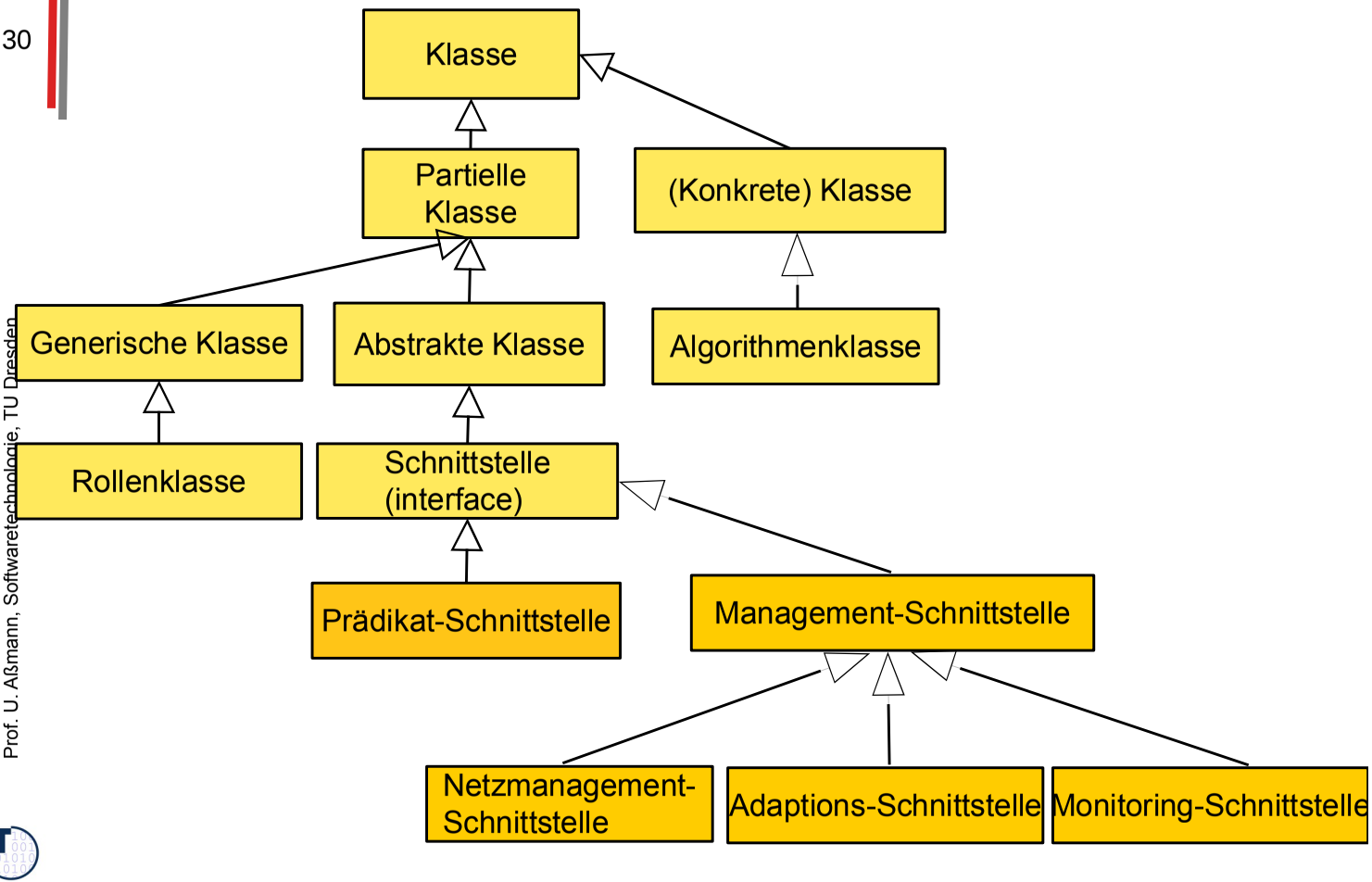
- Klassenkomponenten werden im **Kontextmodell (Schnittstellenmodell)** des Softwaresystems eingesetzt
  - Das System ist selbst eine Komponente
  - Die oberen (äußeren) Schichten stellen geschachtelte Komponenten dar
- Das Kontextmodells enthält *angebotene* und *benötigte* Schnittstellen:
  - Funktionale und strombasierte Schnittstellen (call und stream ports)
  - GUI-Bildschirme, Masken, Formulare



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

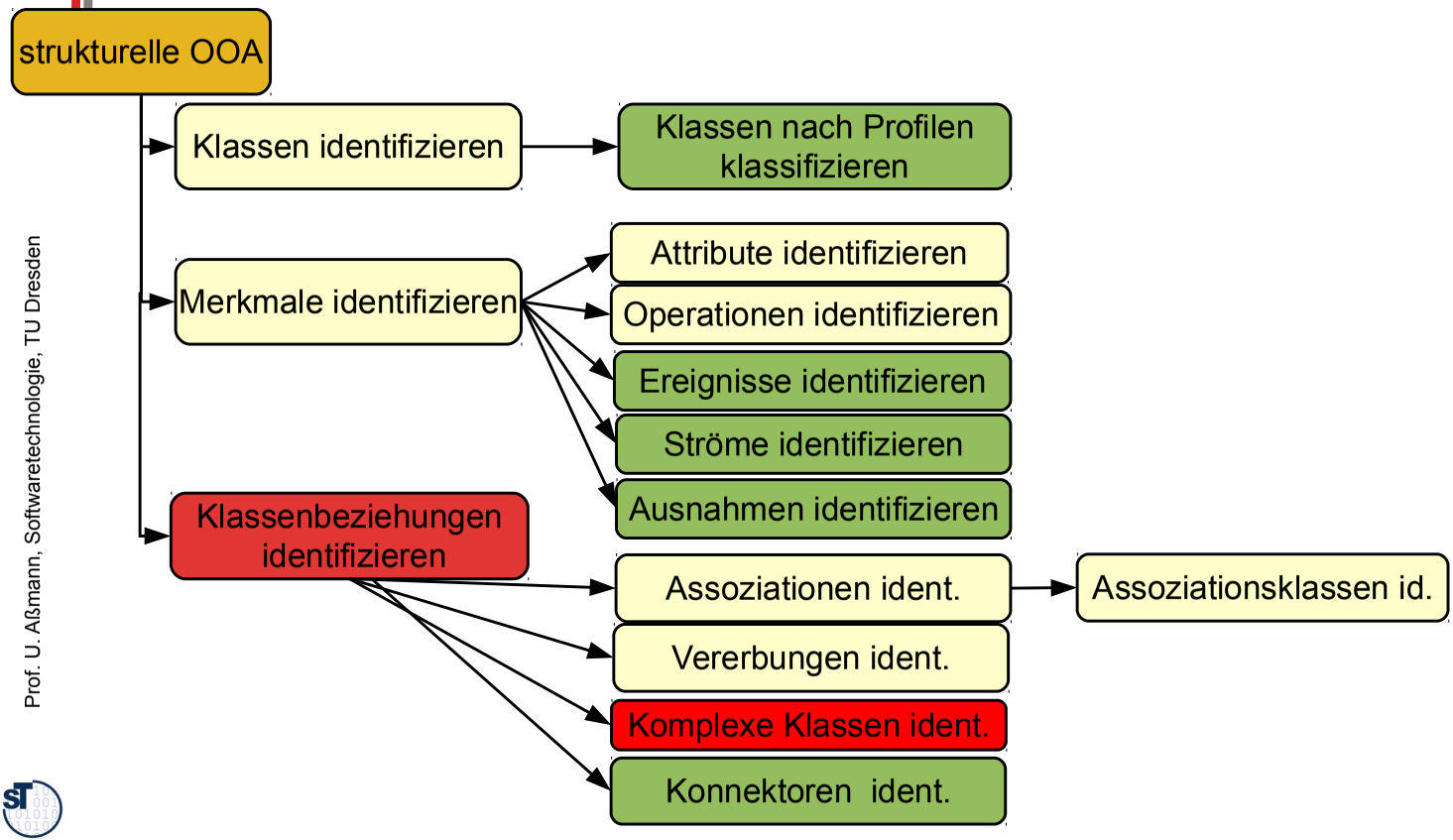
Begriffshierarchie von Klassen (Erweiterung)



32.3 Modellierung von komplexen Objekten aus Kernen und Unterobjekten

Schritte der Analyse

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



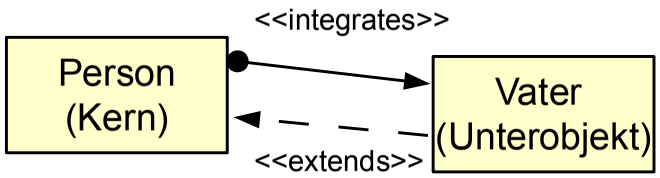
Komplexe Objekte (Big Objects)

- Meist wird ein Objekt einer fachlichen Domäne (komplexes, großes, logisches Objekt) mit *mehreren* Implementierungsobjekten (physikalischen Objekten in einer Implementierung) repräsentiert

- **Analyseobjekte** modellieren die Eigenschaften eines Objekts aus der Domäne
  - Sind komplex mit Lebenszeit, Kontexte, Struktur, Hierarchie, Verhalten
- **Entwurfsobjekte** modellieren zusätzlich technische Eigenschaften
  - Komplex; mit Abbildungsinformation auf technische Plattformen, Resourcenbeschränkungen
- **Implementierungsobjekte** (einfache, physikalischen Objekte) sind einfach, flach und passen direkt auf die Maschine
  - Sie entsprechen Verbunden (records)
  - Sie tragen keine Analyseinformation mehr, sind nackt

Kernobjekte, Unterobjekte und Erweiterungen

- Ein **Unterobjekt (Satellit)** ist ein Objekt, das an ein **Kernobjekt** angelagert ist und mit ihm ein integriertes **komplexes Objekt** bildet
  - Das Unterobjekt hat also keine eigene Identität, sondern teilt seine Identität mit dem Kernobjekt
  - Das Unterobjekt repräsentiert eine Eigenschaft bzw. einen Teilzustand des komplexen Objekts
- Die **Integrationsrelation** <<integrates>> ist eine Relation, die die Erweiterung eines Kernobjektes mit einem Unterobjekt beschreibt
  - Kern und Unterobjekt sind logisch *eins*, d.h. teilen ihre Identität
  - Ihre Inverse ist die Erweiterungsrelation <<extends>>
- Es gibt verschiedene Arten von Unterobjekten, z.B. Teile (Aggretate, Komposita), Rollen, Phasen,..



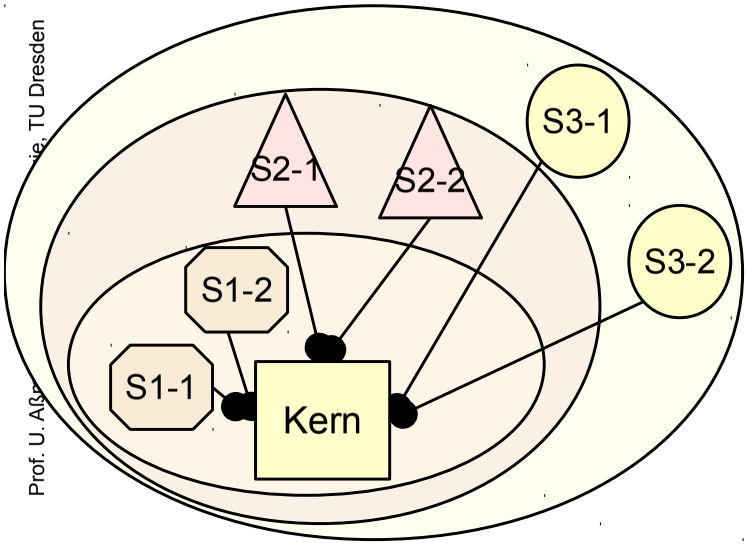
Verwendung der Hierarchie-Notationen in der Analyse

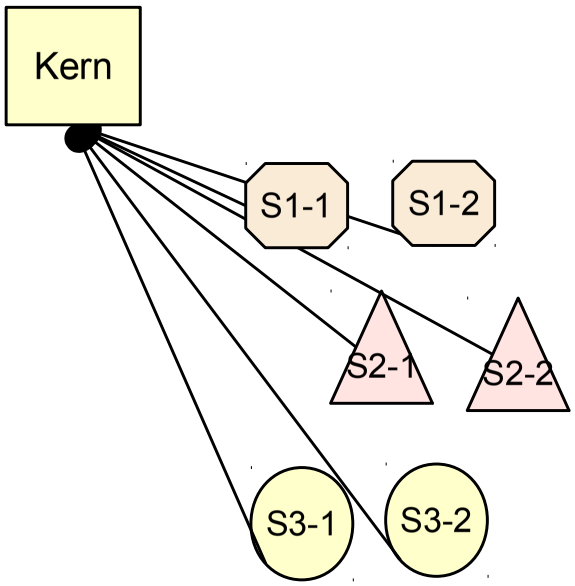
In der Analyse werden komplexe Objekte aus einem *Kern* und einer Menge von zugehörigen *Unterobjekten* dargestellt

- Komplexe Objekte als Kern mit **Satelliten**, die in den Kern mit integrates-a integriert sind
- Komplexe Objekte sind immer *hierarchisch* oder *azyklisch*, bilden also immer Hierarchien oder gerichtete azyklische Graphen:
  - Hierarchische Objekte in **Baum-Notation** in einem **hierarchischen** komplexen Objekt (Kernobjekt als Wurzel)
  - Geschichtete Objekte in **Dag-Notation** in einem **geschichteten** komplexen Objekt (Kernobjekt als Wurzel eines gerichteten azyklischen Graphen, directed acyclic graph, dag)

Satelliten-Notation von komplexen Objekten

- Die **Satelliten-Notation** zeigt eine Schichtung, Hierarchie oder azyklische Struktur, von Satelliten um einen Kern
- Kern und Satellit werden durch die Endoassoziation integrates-a verbunden





Arten von komplexen Objekten (Analyse- und Entwurfsobjekten)

- **Analyseobjekte:**
  - **Domänenobjekte**
    - **Simuliertes Objekt**: kommt in der Umgebung des Programms vor (Simulation der Umwelt)
      - Kunde, Huhn, Auto, Radar, ...
    - **Geschäftsobjekt**: abstraktes Objekt im Geschäftsbereich der Anwendung
      - Rechnung, Termin, Bestellung, Bestellposition
  - Objekte in der Top-Level-Architektur
- **Entwurfsobjekte**:
  - **Schnittstellenobjekte** (im Kontextmodell): Widget, Stream, File
  - **Technische Objekte**: Objekte des Systems, von dem der Kunde nichts sieht: Komponente, Treiber, Datenbank, ...
- Alle komplexen Objekte haben einen **Lebenszyklus**, der durch eine Zustandsmaschine beschrieben wird
  - Interpretierer-Funktion: Sie empfängt Befehle wie Create(), Open(), Read(), Write(), Do(), Enter()
  - Steuerungsfunktion: Sie sendet weitere Befehle aus (Steuerungsmaschine)

Arten der Verfeinerung des Analysemodells

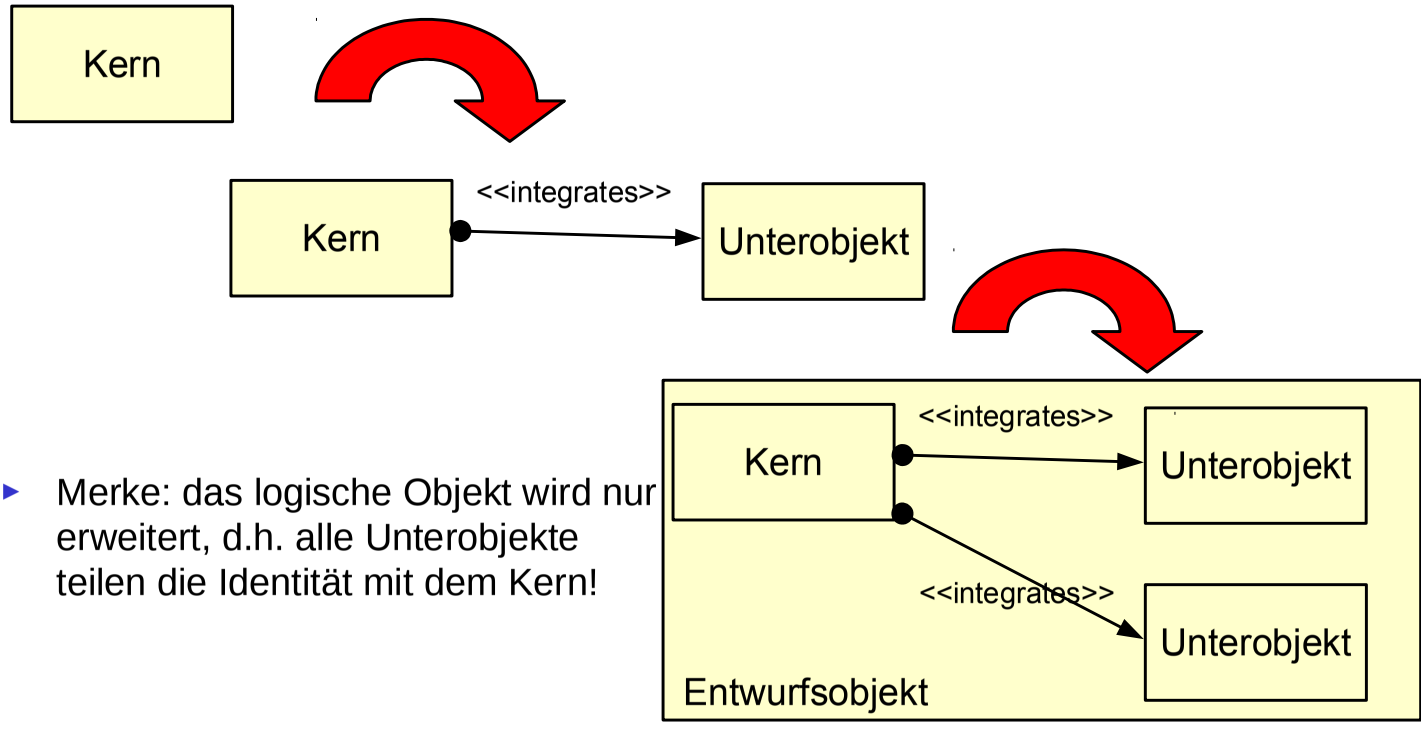
Auf dem Weg vom Analysemodell über das Entwurfsmodell zum Implementierungsmodell und Code werden wir folgende Arten von Verfeinerungsmethoden kennenlernen:

1) **Verfeinerung durch strukturelle Anreicherung von Klassendiagrammen**
   - Ausfüllen von Einzelheiten (Kapitel OOD.2)
2) **Punktweise Verfeinerung von Lebenszyklen**: Abbilden von Lebenszyklen auf niederere Schichten
   - Interpreterverfeinerung (Automatverfeinerung, Verfeinerung von abstrakten Maschinen) (Kapitel OOD.3)
   - Tool-Verfeinerung
   - Material-Verfeinerung
3) **Querschneidende Objektanreicherung (object fattening)**: Verfeinerung von Objekten aus dem Domänenmodell durch Integration von Unterobjekten
   - Jedes Unterobjekt stellt eine neue Eigenschaft des Domänenobjektes dar
   - Szenarioanalyse (Kapitel OOA.4)
   - Kapitel OOD.4

32.3.2 Objektanreicherung

Integration von Unterobjekten in Analyseobjekte zu komplexen Objekten

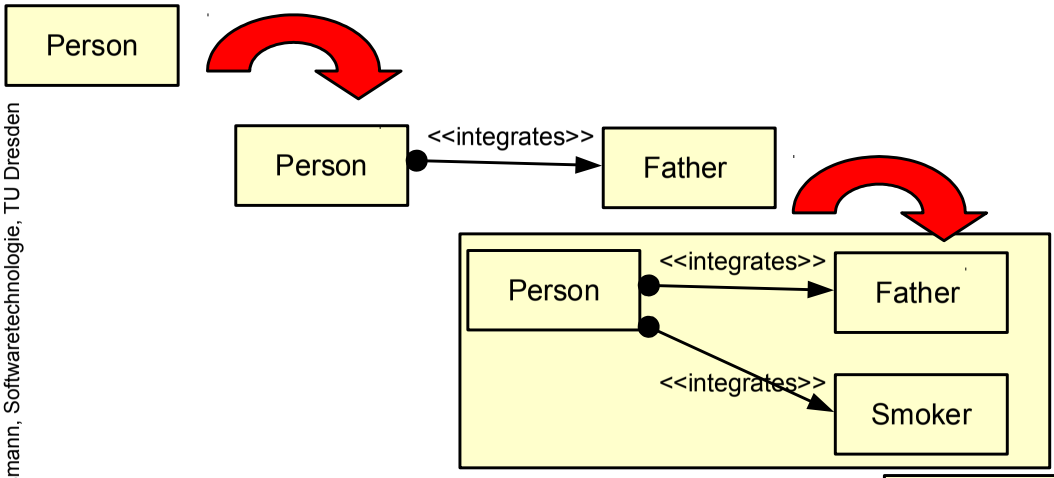
- **Integration von Unterobjekten** besteht aus der Anreicherung von Analyseobjekten aus dem Domänenmodell



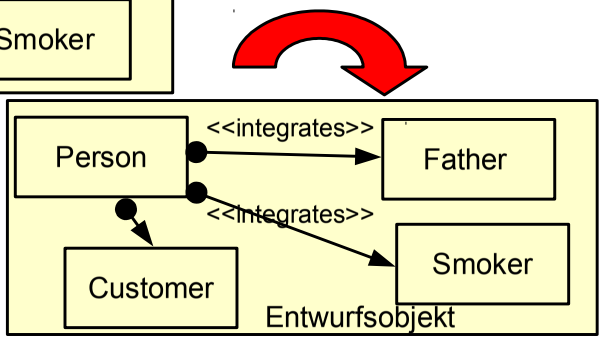
- Merke: das logische Objekt wird nur erweitert, d.h. alle Unterobjekte teilen die Identität mit dem Kern!

Beispiel: Integration von Unterobjekten in Personen

- Die Modellierung von Personen ist ein wesentliches Problem vieler Anwendungen.
- Hier kann mit der **Integration von Unterobjekten** an einen Personen-Kern zusätzliche Funktionalität modelliert werden

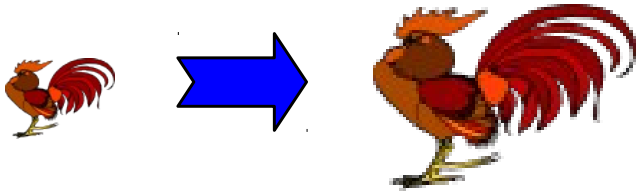


- Merke: das logische Objekt Person wird nur erweitert, d.h. alle Unterobjekte Father, Smoker, Customer teilen die Identität mit dem Kern!



Objektanreicherung (Object Fattening)

- **Objektanreicherung (Object fattening, Objektverfettung)** ist ein Verfeinerungsprozess zur *Entwurfszeit*, der an ein *Kernobjekt aus dem Domänenmodell* Unterobjekte anlagert (Domänenobjekt-Verfeinerung durch Integration), die
  - Unterobjekte ergänzt, die Beziehungen klären zu
    - Plattformen (middleware, Sprachen, Komponenten-services)
    - Komponentenmodellen (durch Adaptergenerierung)
  - Ziel: Ableitung von Entwurfs- und Implementierungsobjekten
- Merk-Brücke “object fattening”:
  - Objekte (“Hühner”) aus dem Domänenmodell werden Stück für Stück mit Implementierungsinformation in Unterobjekten verfettet
  - Die Objekte aus dem Domänenmodell bleiben erhalten, werden aber immer dicker



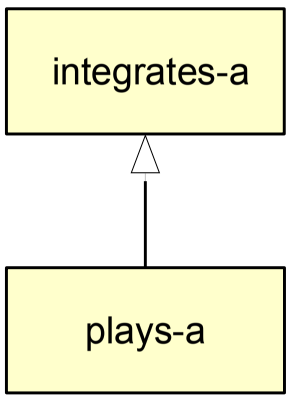
Arten von Verfeinerung durch Integration von Unterobjekten (Object fattening)

Bei der Objektanreicherung können verschiedene Arten von Unterobjekten zur Verfeinerung benutzt werden:

- **Rollenverfeinerung**:
  - Finden von **Konnektoren (teams, collaborations)** zwischen Anwendungsobjekten
- **Teileverfeinerung**:
  - Finden von privaten Teilen von komplexen Analyseobjekten, die integriert werden
  - Schichtung ihrer Lebenszyklen
- Optional:
  - **Facettenverfeinerung**: Finden von Facetten-Unterobjekten, die Mehrfachklassifikationen ausdrücken
  - **Phasenverfeinerung:** Finden von Phasen-Unterobjekten, die Lebensphasen des komplexen Objektes ausdrücken

Unterobjekte gehören immer zu einer dieser speziellen Kategorien.

32.3.3 Natürliche Typen und Rollen



Verschiedene Arten von Unterobjekten

- Die verschiedene Arten von Unterobjekten ergeben sich aus einer Matrix von Typqualitäten
  - Hier: Überblick über das Folgende

	Nicht-Fundiert	Fundiert
Rigide	Natürlicher Typ Kern Facette Privates Teil	
Nicht-rigide	Phase	Rolle

Rigide Typen

Besitzt ein Objekt einen **rigiden Typ**, stirbt es, sobald es die Typeigenschaft verliert [N. Guarino]

- Beispiel:
  - *Buch* ist ein rigider Typ
  - *Leser* ist ein nicht-rigider Typ
  - Ein Leser kann aufhören zu lesen, aber ein Buch bleibt ein Buch
  - Weitere rigide Typen: Begriffe der realen Welt: *Person, Car, Chicken*
- Rigidität ist eine Typqualität:
  - Rigide Typen sind verknüpft mit der *Identität* der Objekte
  - Nicht-rigide Typen sind dynamische Typen, die den Zustand eines Objektes anzeigen

Fundierte Typen

Ein **fundierter Typ (kontextbezogener Typ)** beschreibt Eigenschaften eines Objektes, die in Abhängigkeit von einem Kooperationspartner bestehen

- Beispiel:
  - *Buch* ist ein nicht-fundierter Typ
  - *Leser* ist ein fundierter Typ
  - Ein Leser ist nur ein Leser, wenn er ein Buch liest (kontextbasiert), während ein Buch ein Buch ist, auch wenn es niemand liest
- Fundiertheit ist eine Typqualität, die Kontextbezug eines Typs ausdrückt

Natürliche Typen und Rollen

Ein natürlicher Typ ist ein nicht-fundierter und rigider Typ.

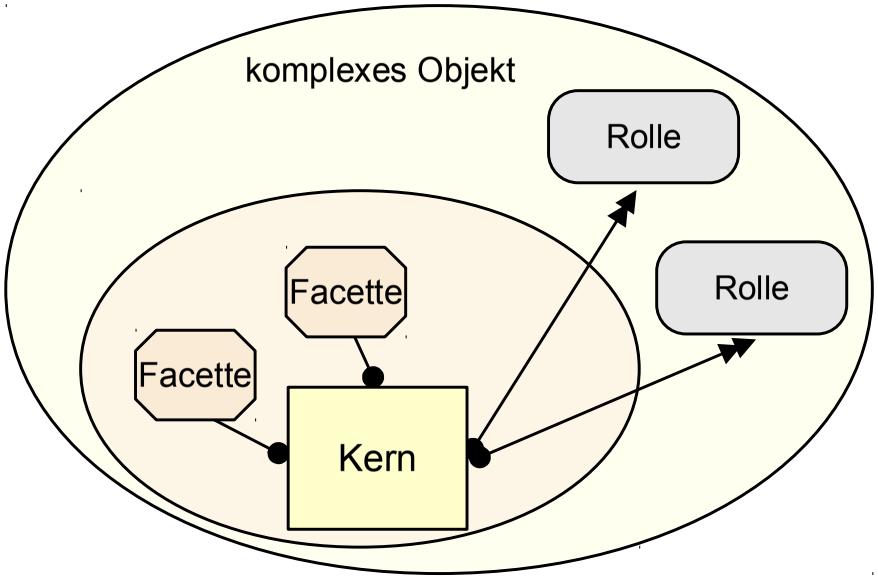
Ein *Rollentyp (Fähigkeit)* ist ein fundierter und nicht-rigider Typ.

- **Rollentypen (Fähigkeiten)** existieren nur in Abhängigkeit von einer Kollaboration.
  - Ein Rollentyp entspricht also einer partiellen Klasse
  - in UML: **Role (or role type)**: “The named set of features defined over a collection of entities participating in a particular context.”

	Nicht-Fundiert	Fundiert
Rigide	(* natürlich *) Kern, Facette, privates Teil	
Nicht rigide	Phase	Rolle

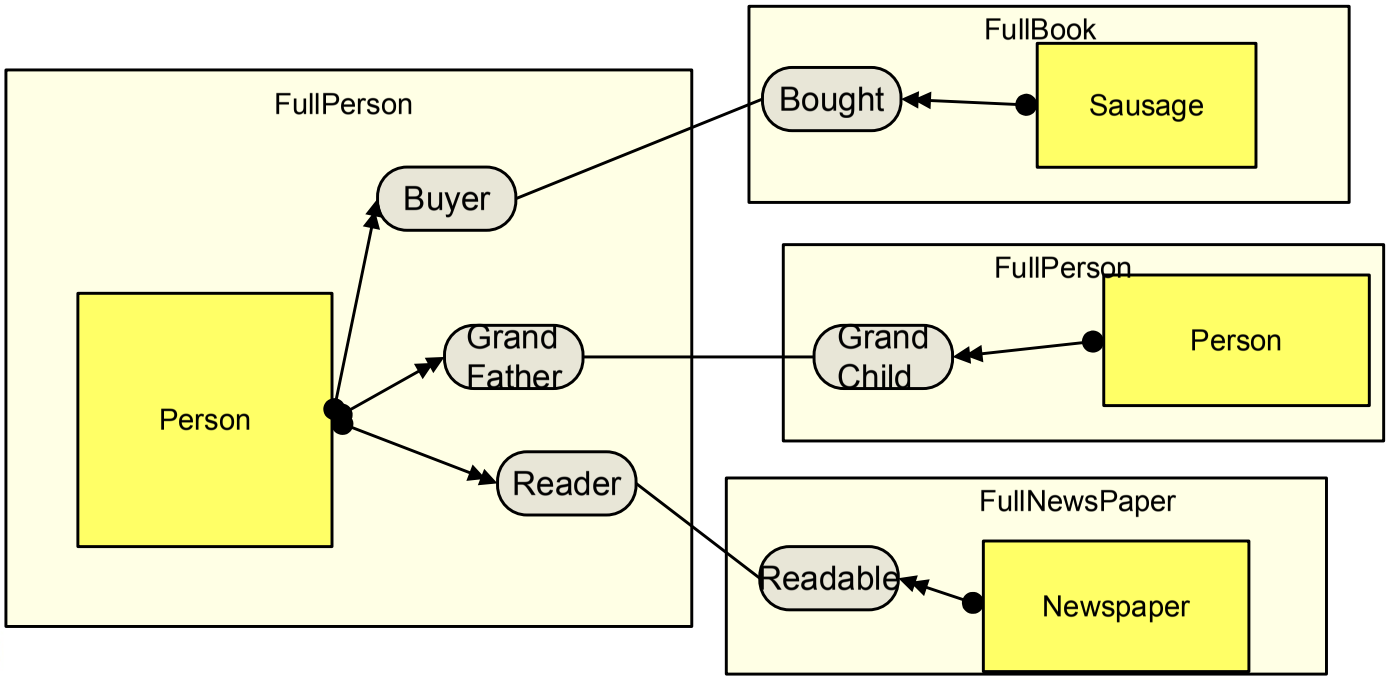
Komplexe Objekte mit Rollen

- Man sagt: ein Objekt *spielt eine Rolle (object plays a role)*
  - Ein Rollentyp (Fähigkeit) wird zur Laufzeit durch ein fundiertes Unterobjekt repräsentiert, ein Unterobjekt eines Kernobjekts, das in Abhängigkeit von einem Kooperationspartner existiert



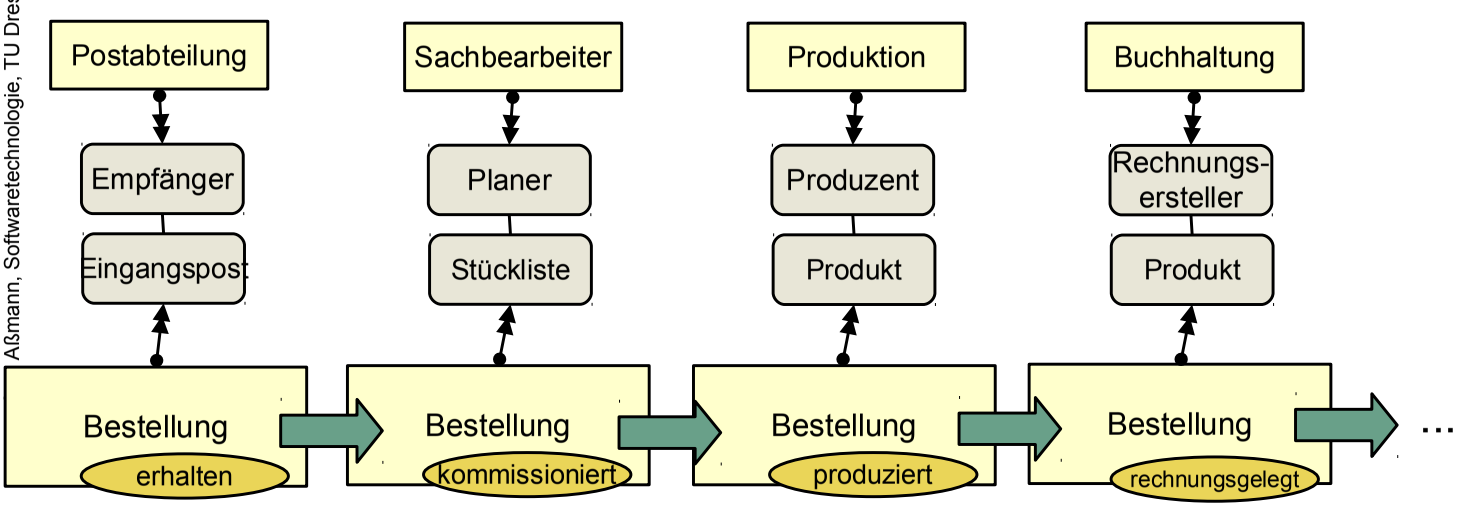
Die Steimann-Faktorisierung [Steimann]

- Man teilt den Typ eines Objektes in seine *natürlichen Kern* und seine *Rollen-Unterobjekte* auf
  - FullType = Natural-type x (role-type, role-type, ...)
  - Bsp.: FullPerson = Person x (Reader, Father, Customer, ..)



Rollen in Geschäftsobjekten (Business Objects)

- In Geschäftsobjekten kommen immer Rollen-Unterobjekte vor
- `Bestellung` als Beispiel: eine Bestellung durchwandert mehrere Bearbeiter
  - Auftragseingang des Kunden, Produktion, Kommissionierung, Rechnungserstellung, Auslieferung und Mahnwesen
- Dynamische Erweiterbarkeit bzw. Adaption durch neue bzw. wechselnde Rollen-Unterobjekte nötig:

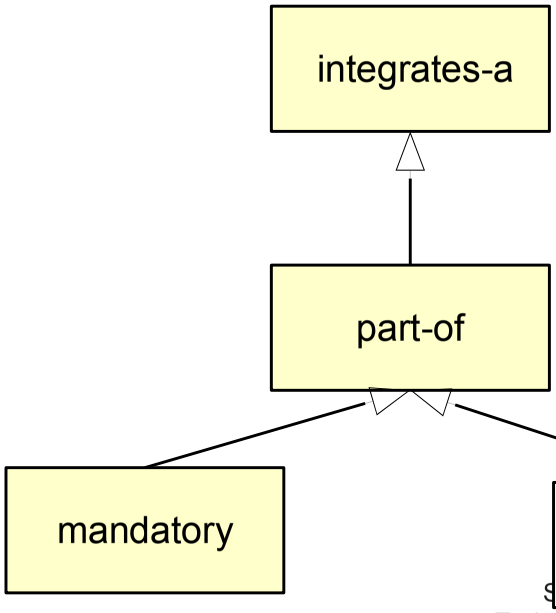


Informationssysteme

- Ein **Informationssystem** ist ein Softwaresystem, dessen primäre Aufgabe in der Information und Verwaltung von physischen oder immateriellen Materialien besteht (Produkte, Vorräte, Teile, Geld, Guthaben, etc.)
- Der Fluss der Materialien durch die Firma verändert die Materialien

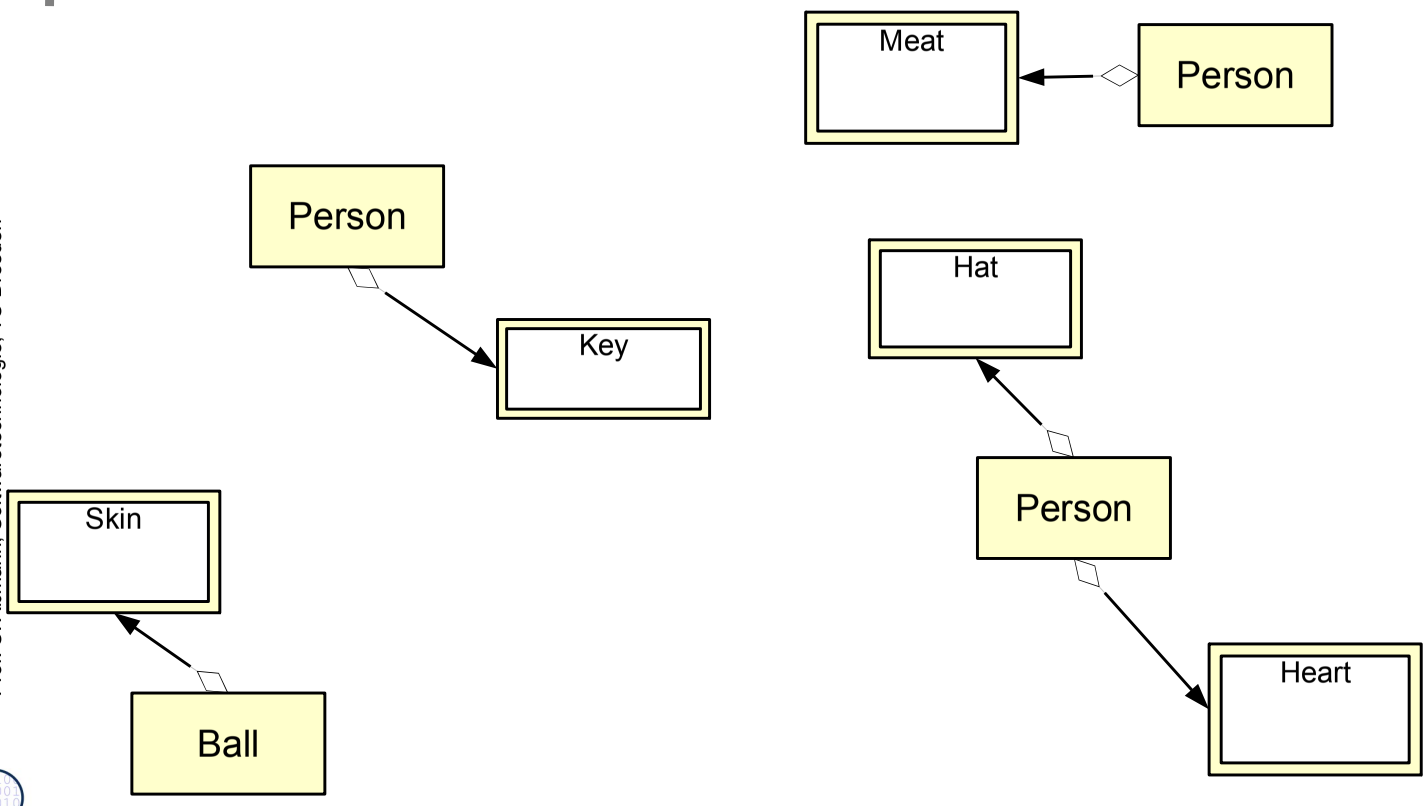
In Informationssystemen modellieren Rollen die wechselnden Kontexteigenschaften von Materialien.

32.3.4 Private und essentielle Teile von kompositen Objekten



Private und essentielle Teile

- Gibt es hier einen Unterschied in der Bedeutung der Aggregationen?



Private Teile für komposite Objekte

Private Teile

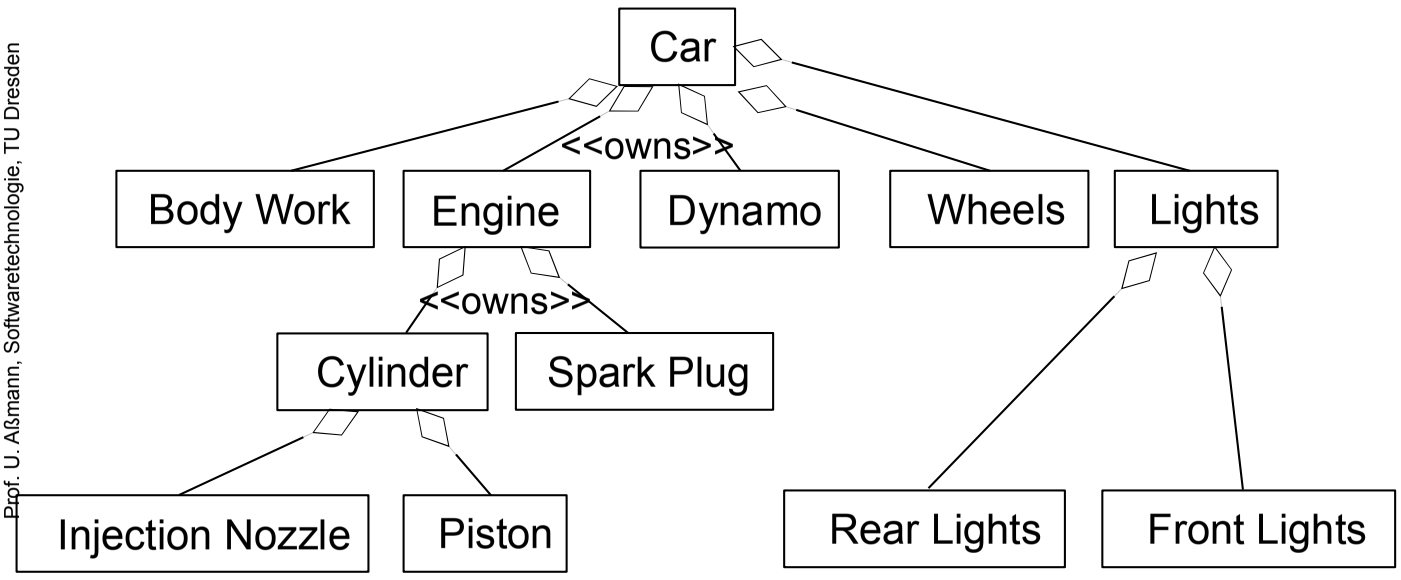
- Ein **privates Teil (owned part)** ist ein nicht-fundiertes Unterobjekt, das ausschließlich zu einem Kernobjekt gehört
  - Beispiel: Eine Person hat einen Hut
  - Zu einer Zeit hat das Teil genau einen Eigner (alias-freie Ganz/Teile-Beziehung), aber das Teil kann das Ganze wechseln
  - Stereotyp <<owns>>
- Ein **zugeeignetes Teil (exclusively owned part)** ist ein rigides privates Unterobjekt, das immer zu einem Kernobjekt gehört
  - Beispiel: eine Person hat einen Arm
  - Teil gehört exklusiv dem Ganzen und kann die Zugehörigkeit nicht ändern
  - Stereotyp <<exclusively-owns>>

Obligatorische Teile:

- Ein **obligatorisches Teil (mandatory part)** ist ein zugeeignetes Unterobjekt, von dessen Typ das Kernobjekt unbedingt eines braucht
  - Beispiel: Eine Person braucht ein Herz
  - das Ganze braucht ein Teil vom Typ des Teils
  - Stereotyp <<mandatory>>
- Ein **wesenhaftes Teil (essential part)** ist ein essentielles Teilobjekt, das nicht ausgewechselt werden kann, ohne das Kernobjekt zu zerstören
  - Beispiel: Eine Person braucht ein Gehirn
  - Das Ganze braucht genau dieses Teilobjekt zum Leben
  - Stereotyp <<essential>>

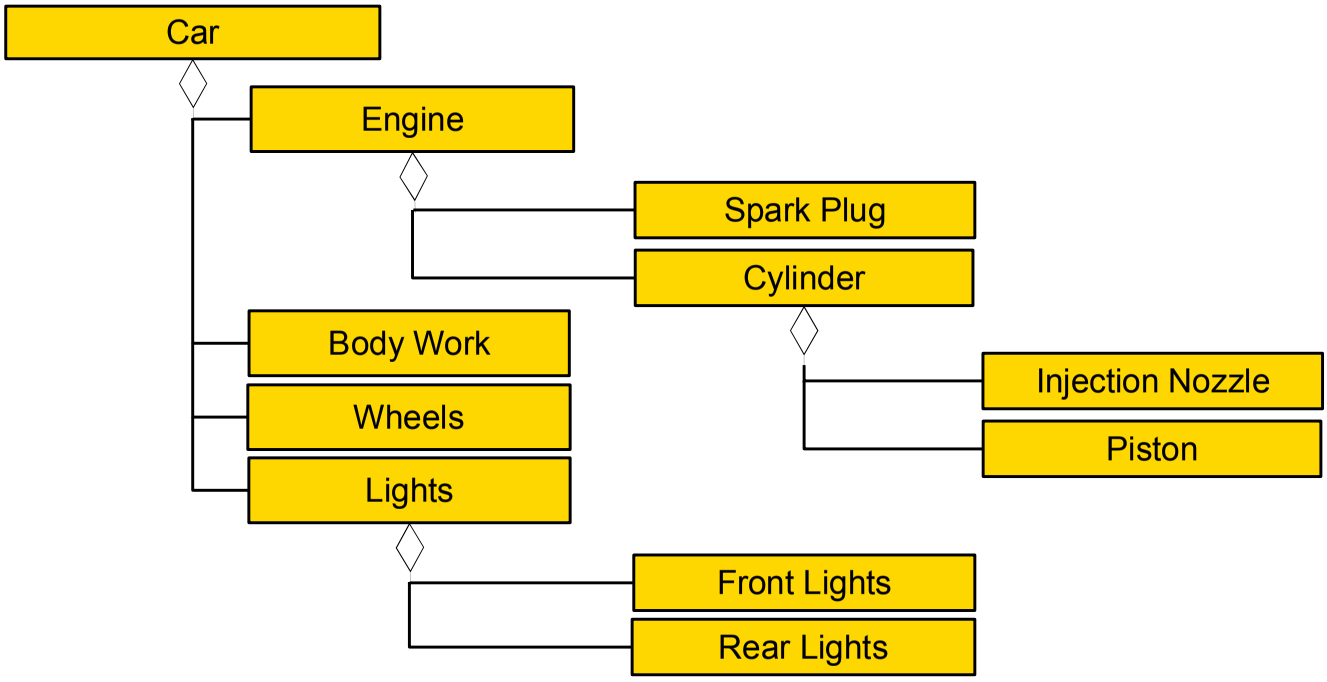
Hierarchische Systemzerlegung mit privaten Teilen (<<owns>>)

- Bei Eigentumsbeziehungen gibt es kein Teilen von Unterteilen (kein *sharing*, kein *aliasing*)
- Merke: die spezielle Semantik von Teile-Relationen kann durch *Stereotypen* angegeben werden



Darstellung komplexer Objekte mit privaten Teilen

- Komplexe Objekte können dargestellt werden
  - Mit Zeilenhierarchien
  - Mit Mind maps

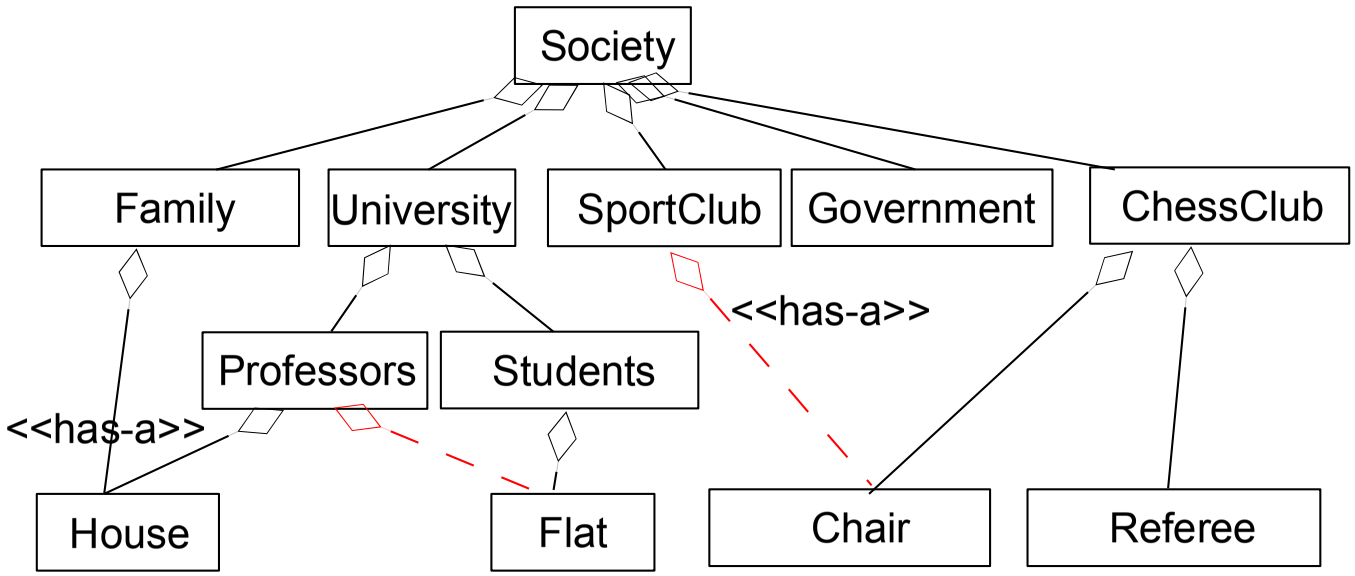


Weitere Ganz/Teile-Beziehungen (Whole-Part Relationships)

- **Eigentumsbeziehungen** bilden baumförmige Relationen
  - *Private Teile (s. vorige Folie)*
  - *Abhängiges Teil (composed-of)* (Komposition in UML): das Teil hat die gleiche Lebenszeit wie das Ganze und kann nicht alleine existieren
- **Einfache Teilebeziehungen** sind azyklisch, bilden aber keine Hierarchien
  - *has-a*: Aggregation, einfache Teilebeziehung. Das Teil kann Teil von mehreren Ganzen sein (Aliase möglich)
  - *member-of*: Wie has-a, aber Gleichheit mit Geschwistern gefordert

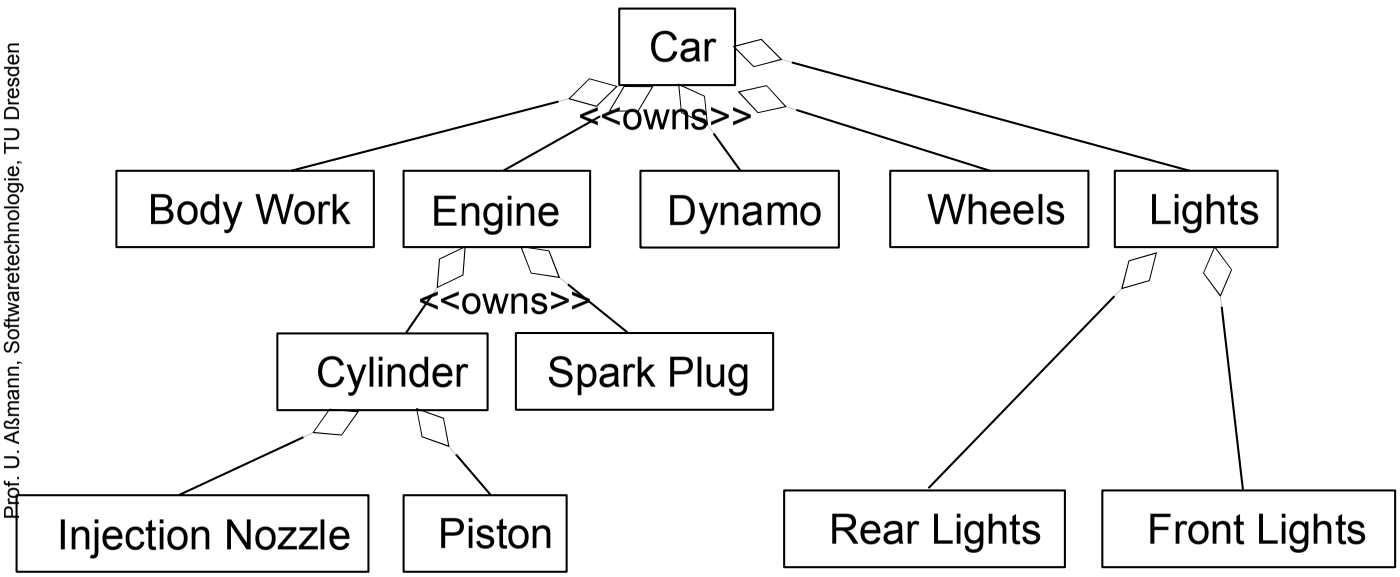
Geschichtete Systemzerlegung mit nicht-privaten Teilen (<<has-a>>)

- Das Teilen von Teilen erzeugt gerichtete azyklische Graphen, die schichtbar sind (*has-a* Relation)



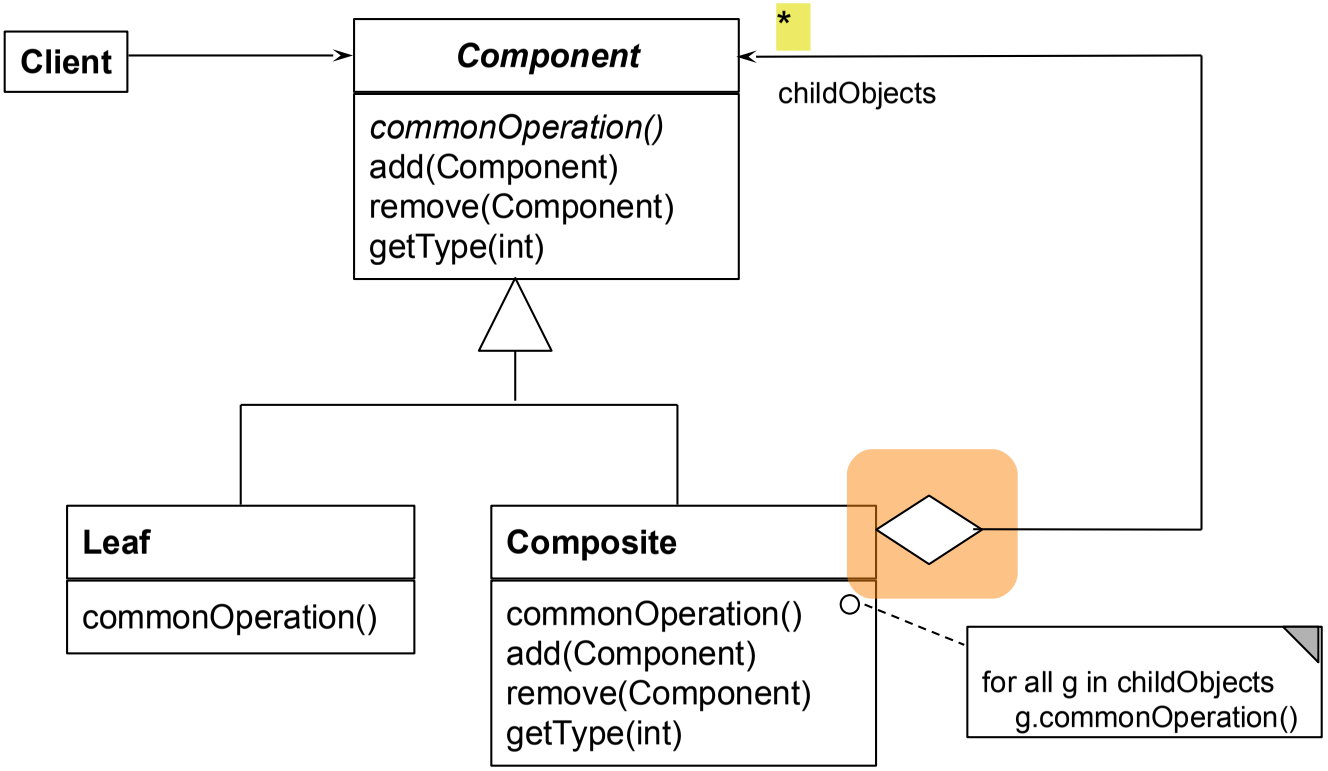
Teile-Verfeinerung durch hierarchische Systemzerlegung mit privaten Teilen

- **Teile-Verfeinerung** beginnt mit den komplexen Ganzen
  - .. und findet Schritt für Schritt neue Teile



Achtung: Welche Aggregation steht im Composite?

- Welchen Unterschied macht es, ob die Kinder eines Composite-Knoten aggregiert, komponiert, oder rollenspielend sind?



Folgen für Informationssysteme

- Materialien in Informationssystemen sind hierarchisch strukturiert

Die genaue Analyse der Ganz/Teile-Beziehung ermöglicht es, Aussagen über die Lebenszeit von Teilen in komplexen Geschäftsobjekten zu treffen.

Was haben wir gelernt?

- Es gibt komplexe Objekte, die so groß sind, dass sie aus Kern und Unterobjekten (Satelliten) bestehen
  - Komplexe Objekte sind immer hierarchisch oder azyklisch und verwenden Endo-Assoziationen
  - Der Kern bildet meist eine Facade für die Unterobjekte
  - Unterobjekte sind typisch einer Kategorie zugeordnet (Rollen, Teile, Facetten, Phasen)
- Objektanreicherung besteht darin, ein komplexes Objekt aus dem Analysemodell durch weitere Unterobjekte im Entwurf und in der Implementierung anzureichern
- Informationssysteme nutzen für die Repräsentation von Geschäftsobjekten Rollen und Hierarchien
- Die objektorientierte Systementwicklung nutzt als hauptsächliches Mittel die Objektanreicherung, um vom Problem des Kunden zum Analysemodell, dann über das Entwurfs- und Implementierungsmodell zum Programm zu kommen.

Ende des obligatorischen Materials

32.A.1 Facettenklassifikation und Facetten-Unterobjekte

65

(optional, nicht klausurrelevant)

Facettenklassifikation

66

- Manchmal kann ein Objekt mehrfach klassifiziert werden
  - Person: (Raucher / Nichtraucher), (Gourmet/Gourmand), (Vegetarier/AllesEsser)
- Im Allgemeinen gilt: Eine *Facette* ist eine Klassifikationsdimension eines Objektes
  - Jede Facette hat ein eigenes Klassendiagramm
  - Die Facetten sind unabhängig von einander
  - Die Facetten bilden einen zusammengesetzten Typ, den *powertype*
- Eine Facette ist ein rigider Typ
- Im Speziellen ist eine Facette ein rigides Unterobjekt, das eine Typdimension eines komplexen repräsentiert.
  - Es kann seinen Typ unabhängig von allen anderen Facetten-Unterobjekten wechseln

http://en.wikipedia.org/wiki/Faceted_classification

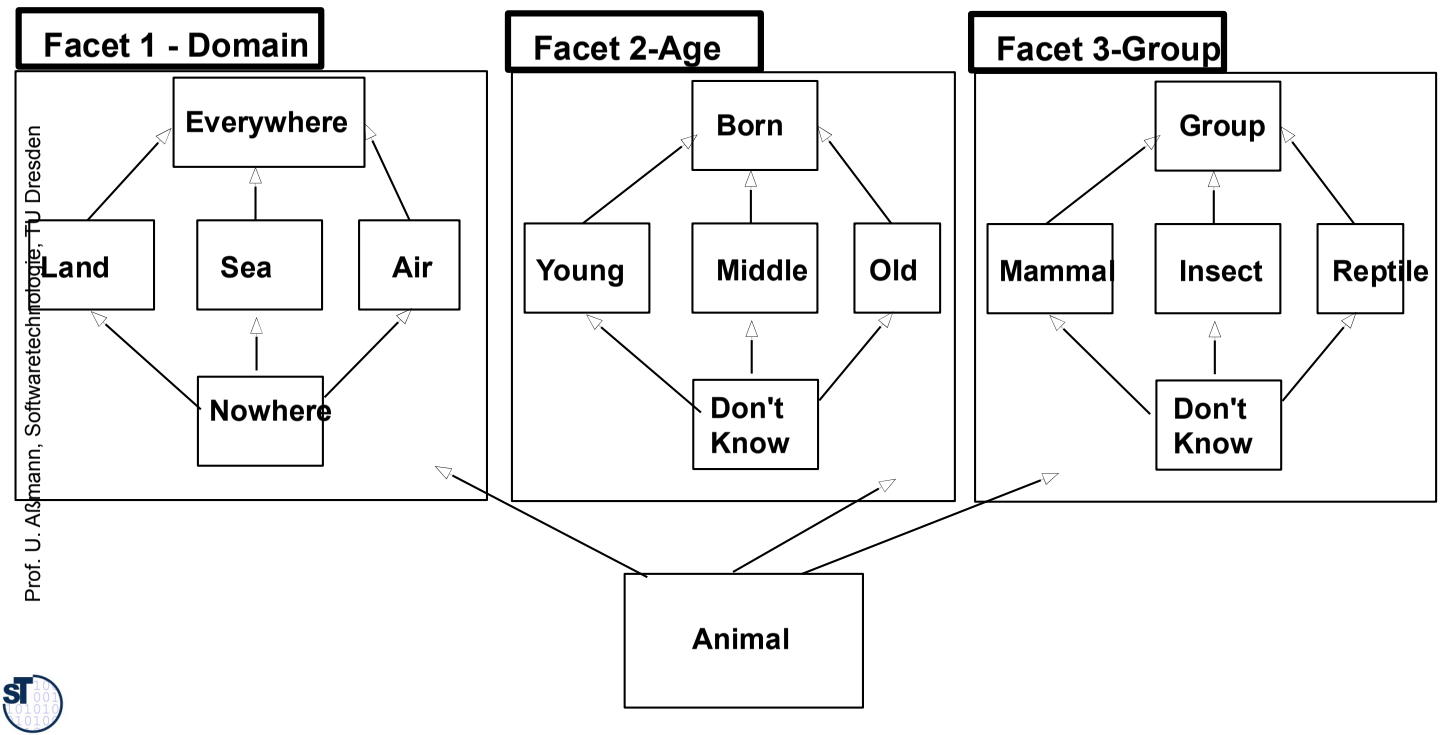
http://www.webdesignpractices.com/navigation/facets.html

Facetten von Lebewesen

- Das folgende Modell von Lebewesen hat 3 Facetten:
  - Lebensbereich
  - Alter
  - Biologische Gruppe
- Ein Tier hat also 3 Facettenunterobjekte

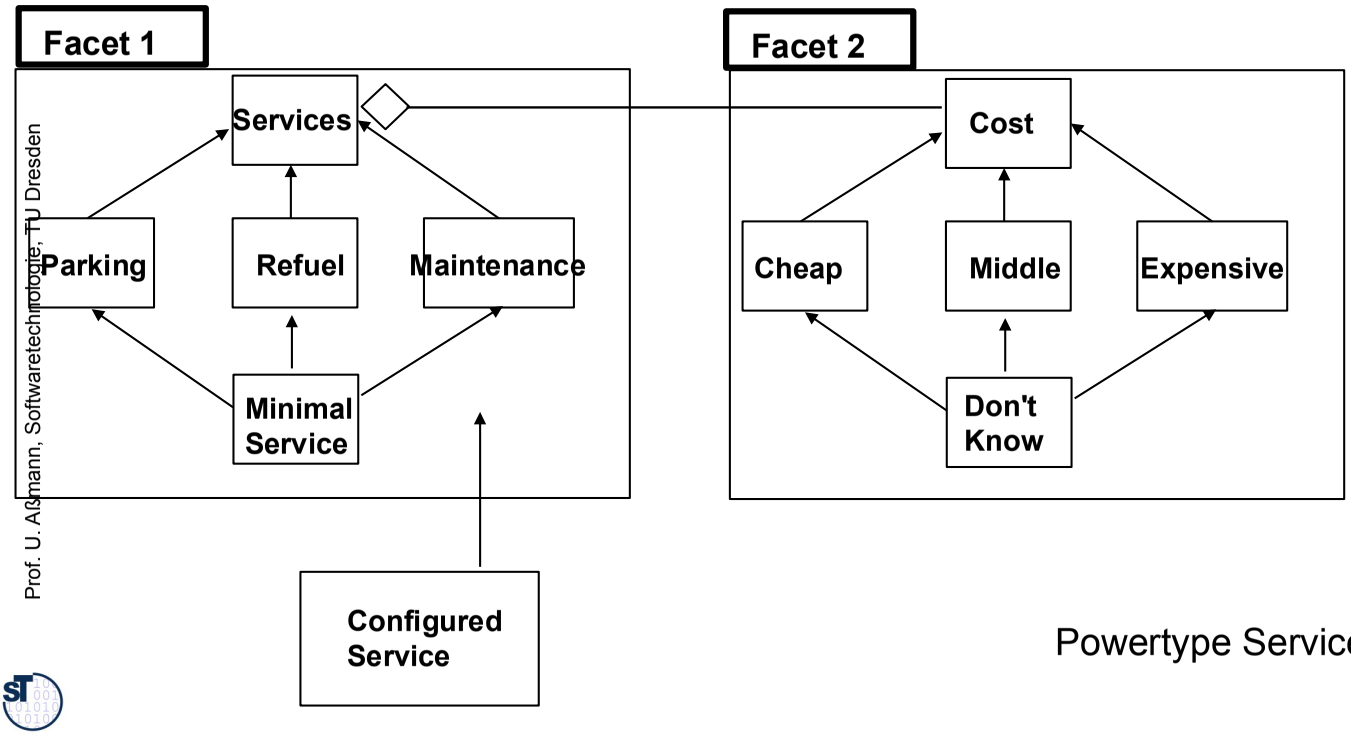
Facetten faktorisieren aus

- Eine Facettenklassifikation ist i.A. einfacher als eine ausmultiplizierte Vererbungshierarchie
  - Bei 3 Facetten braucht ein solches 3^n Klassen



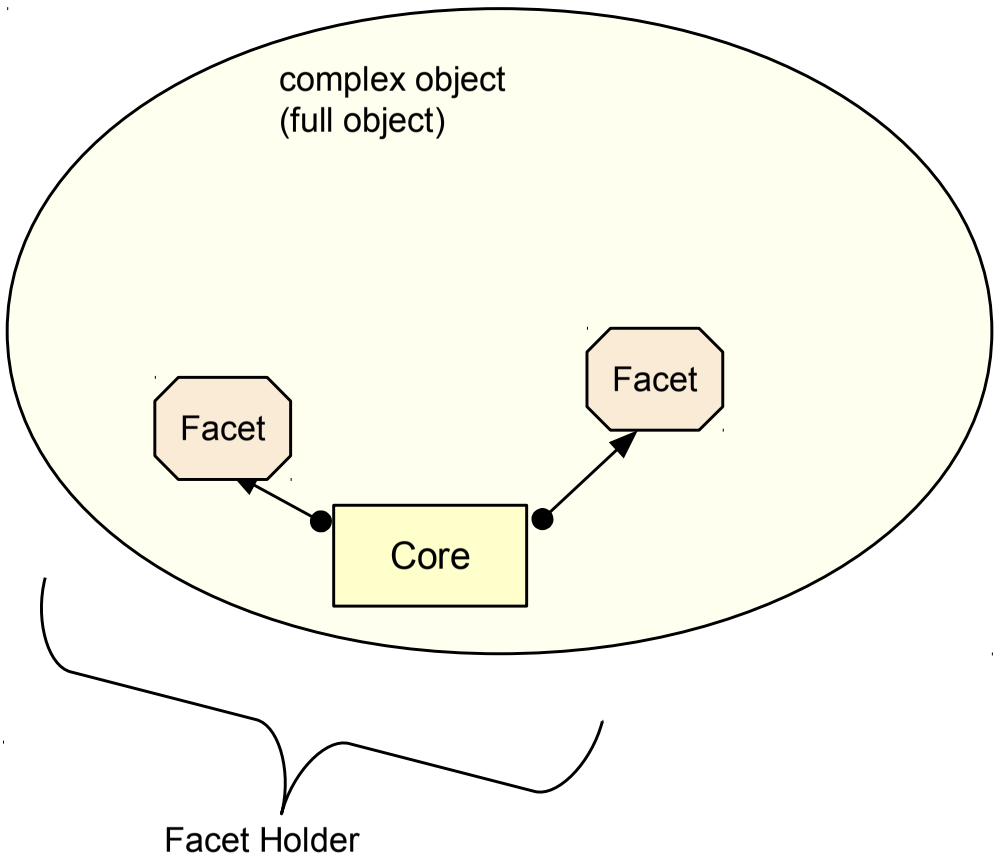
Einfache Realisierung durch Delegation

- Eine zentrale Facette, die anderen angekoppelt durch Aggregation (Delegation)



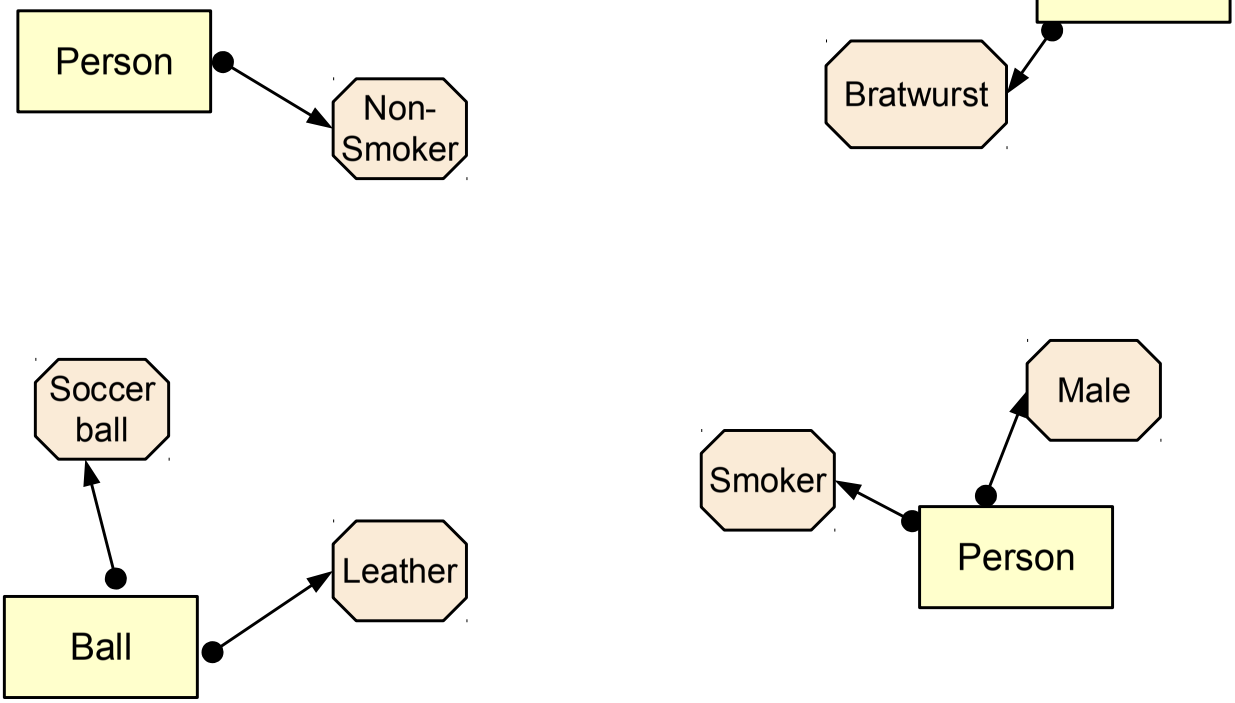
Powertype Service

Facetten, repräsentiert als Unterobjekte



Facetten, repräsentiert als Unterobjekte

- non-founded; rigid



32.A.2 Phasen als Typen

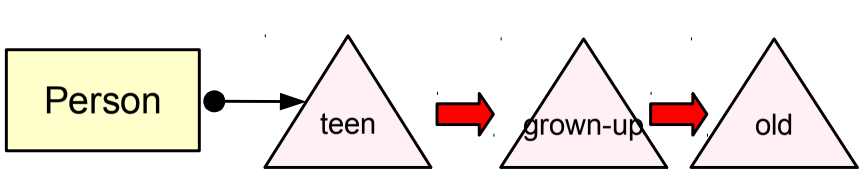
(optional)

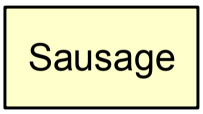
Phasentypen

- Ein **Phasenobjekt** ist ein Unterobjekt, das eine Lebensphase (Zustand) eines komplexen Objektes beschreibt
- Ein **Phasentyp** charakterisiert die Lebensphase eines Objektes in seinem Lebenszyklus
  - Ein Phasentyp ist nicht-rigide, da er sich ändert im Laufe des Lebens

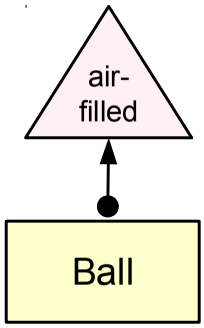
Was sind Phasen?

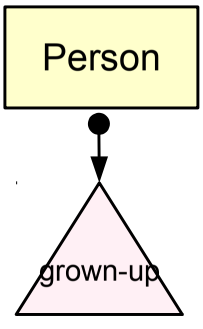
- Phasen sind nicht-fundiert, nicht-rigide Typen



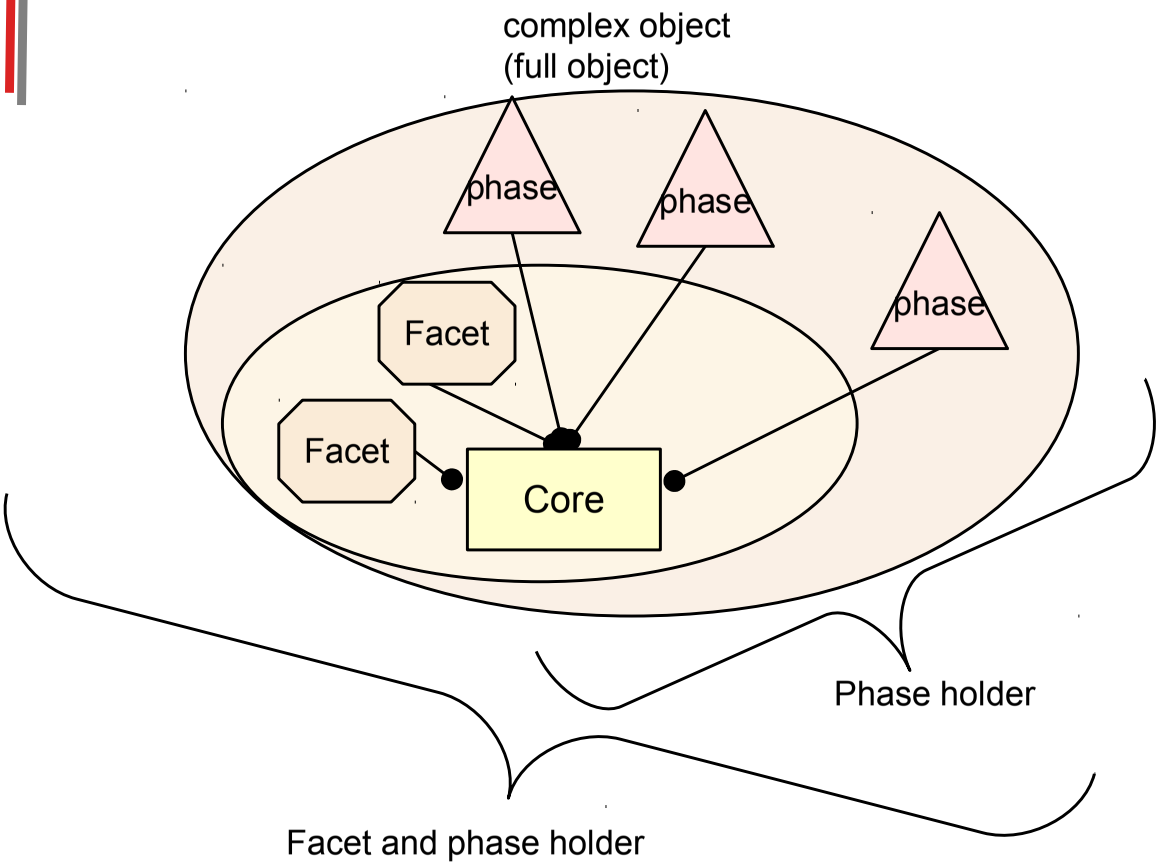


States in a lifecycle





Komplexes Objekt mit Facetten und Phasen



The End