32. Strukturelle Analyse von Verbundobjekten

Die Verwaltung von Geschäftsobjekten in Informationssystemen - Mixin-Anreicherung von Verbundobjekten durch Mehrfach-Brücken (Multi-Bridge)

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 21-1.1, 24.07.21

1) Komplexe Objekte in Informationssystemen
2) Einführung in die Schichten eines Systems
3) Modellierung von komplexen Objekten
   1) Komplexe Objekte und Stücklisten
      - Private und essentielle Teile
   2) Natürliche Typen und Rollen
   3) Realisierung von Rollen und Teilen durch Bridge
4) Mixin-Anreicherung mit Multi-Bridge
5) Komponenten mit Ports
6) Anhang

Obligatorische Literatur

- Störrle 5.3, 5.4
- Weitere nicht-obligatorische Literatur:
  - Bill Lazar. IBM's San Francisco Project. Dr. Dobbs Journal. http://www.drdobbs.com/ibms-san-francisco-project/184415597
  - Vincent D. Arnold ; Rebecca J. Bosch ; Eugene F. Dumstorff ; Paula J. Helfrich ; Timothy C. Hung ; Verlyn M. Johnson ; Ronald F. Persik ; Paul D. Whidden. IBM Business Frameworks: San Francisco project technical overview [Technical forum]. IBM Systems Journal, Volume: 36 Issue: 3
    - https://ieeexplore.ieee.org/document/5387159/
  - 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.
  - [Steimann] Friedrich Steimann. On the representation of roles in object-oriented and conceptual modelling. Data Knowl. Eng, 35(1):83-106, 2000.
  - Beispiel für ein ERP System: Odoo https://www.odoo.com/

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

Ziele

- Die Analyse von Verbundobjekten (komplexen Objekten, Großobjekten, compound objects) spielt eine große Rolle in der OOA
- Verbundobjekte sind strukturiert in Kerne und Mixins (Unterobjekte wie Rollen, Teile, Facetten, Phasen), die durch Endo-Assoziationen zusammengefügt werden
- Verstehe die Natur von Informationssystemen, die Geschäftsobjekte verwalten
  - Beispiel: SAP, Peopleware, ...

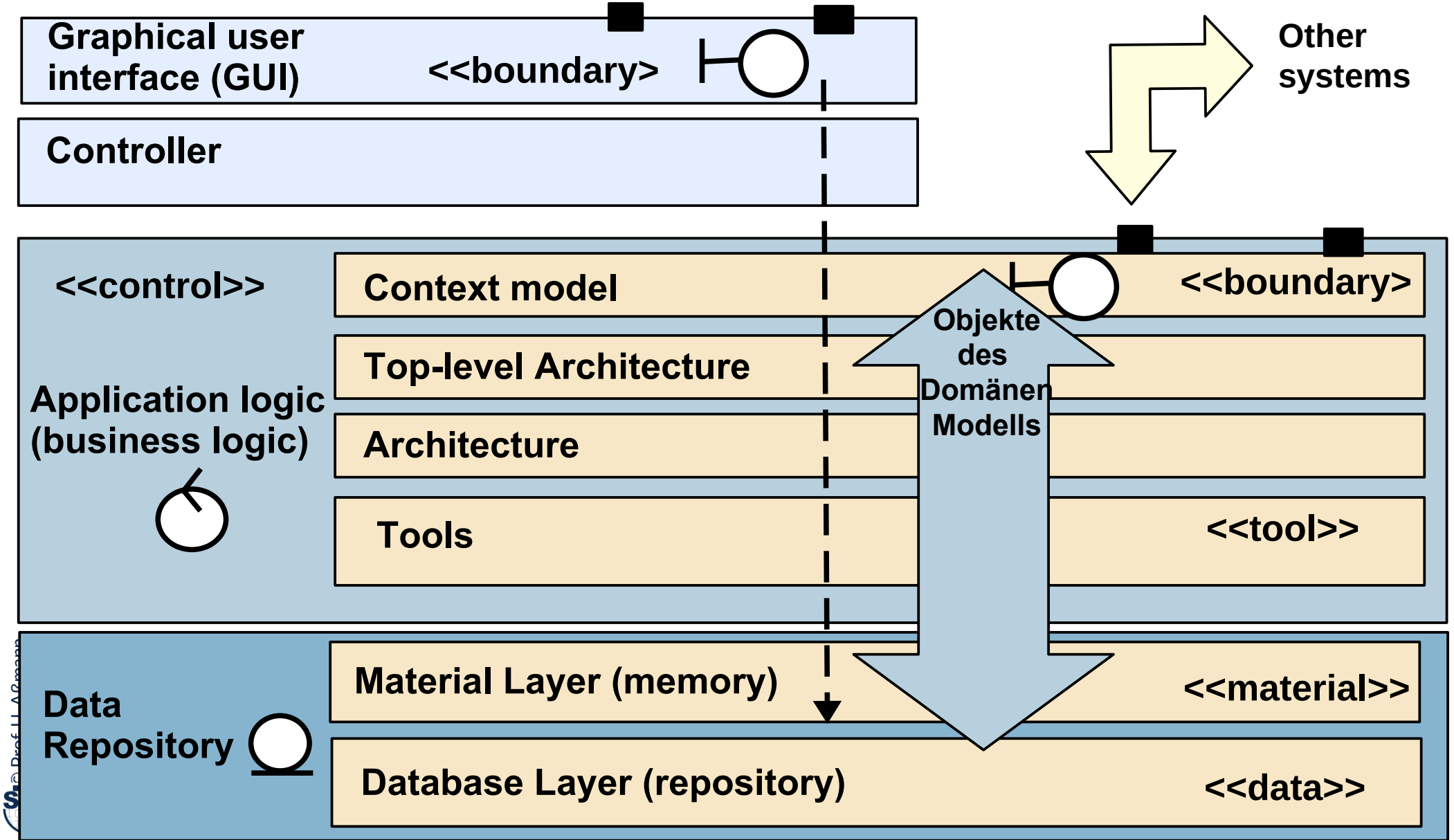
32.1 Einführung in die Schichten eines Softwaresystems

Softwaresysteme bestehen i.d.R. aus Schichten.

Manche Objekte begrenzen sich auf eine Schicht, aber komplexe Objekte kreuzen alle Schichten. Meist wird daher ein komplexes Objekt als *Verbund* über mehrere Schichten angelegt

Q7: Verfeinerte BCED-Schichtung eines Systems

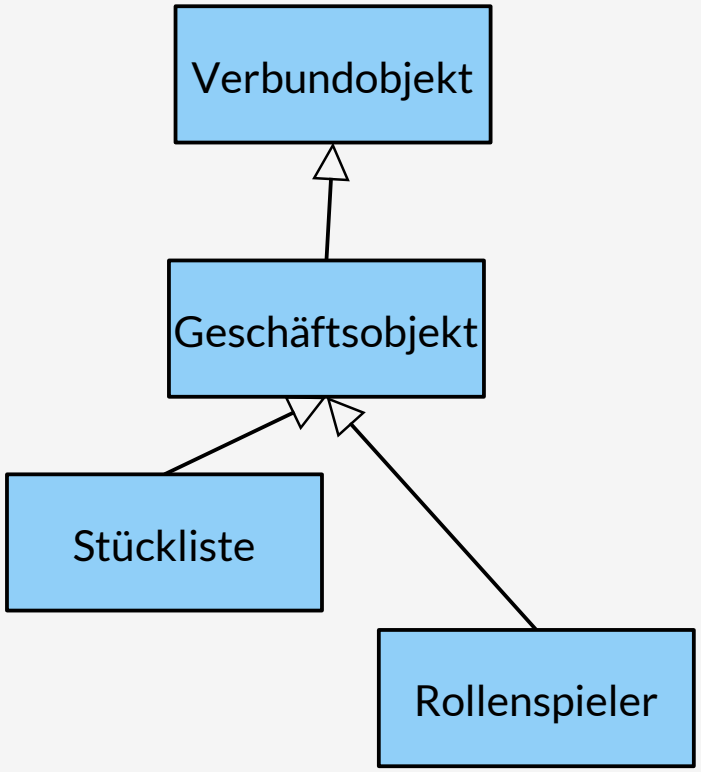
- Im Teil III+IV verwenden wir 7 Schichten in 3 Gruppen:



32.2 Verbundobjekte in Informationssystemen

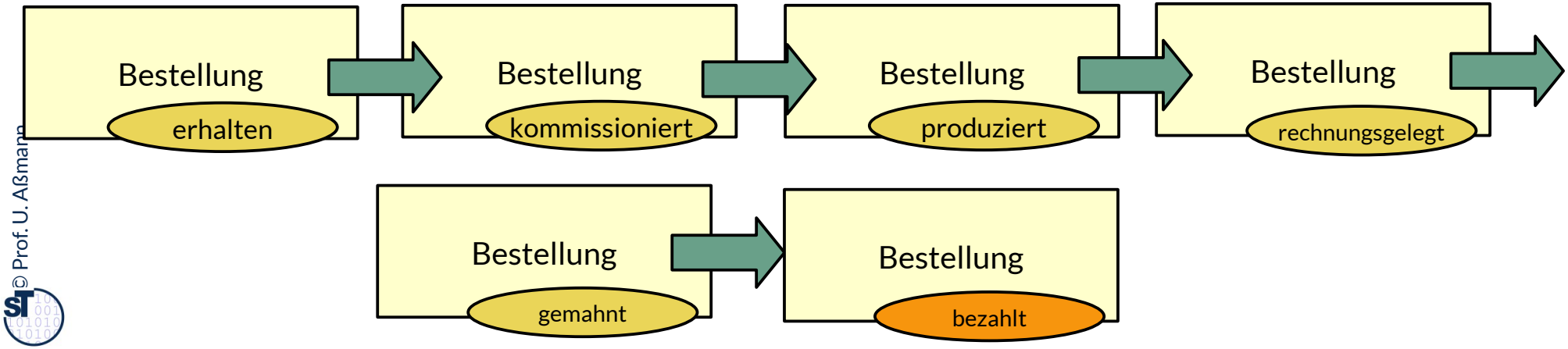
Wir haben in Kap. 31 (Strukturelle Modellierung) bereits die Analyse von Verbundobjekten (komplexen Objekten) kennengelernt. Hier folgen mehr Details:

- Komplexe Objekte mit vielen Teilen
- Rollen als spezielle Teile
- Ports als Fassaden-Mixins



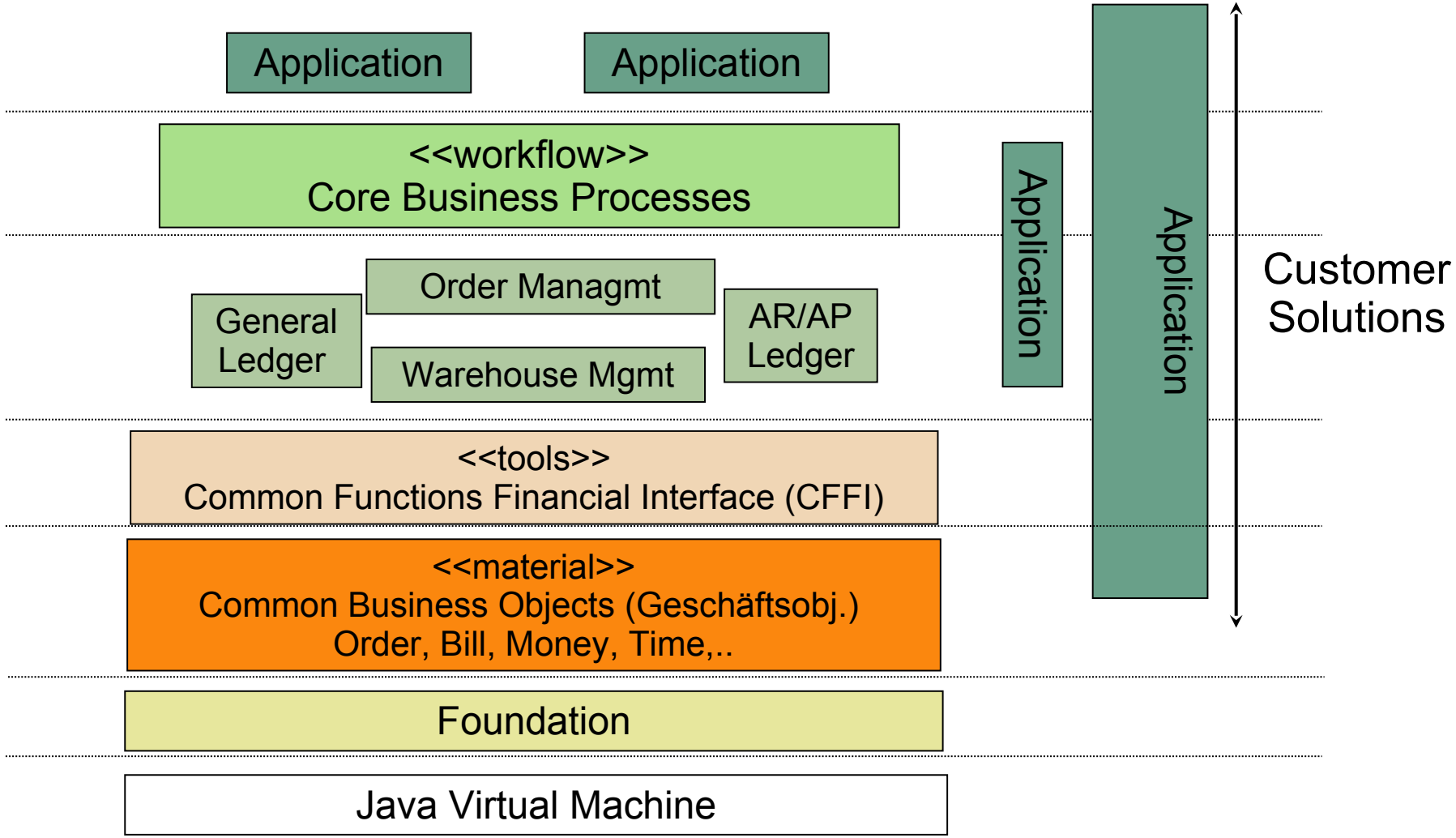
Geschäftsobjekte (Business Objects)

- In großen objektorientierten Frameworks (wie SAP) werden die Objekte sehr komplex
- Beispiel: `Bestellung`, ein *Geschäftsobjekt (business object)* in Geschäftssoftware (Enterprise Resource Planning, ERP):
  - 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 der Klasse `Bestellung` nötig, da die Bestellung verschiedene Phasen durchläuft und daher viele verschiedene Rollen spielt
  - 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: Komplexe Geschäftsobjekte in der SanFrancisco Bibliothek

- Wertobjekte (value objects):
  - 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

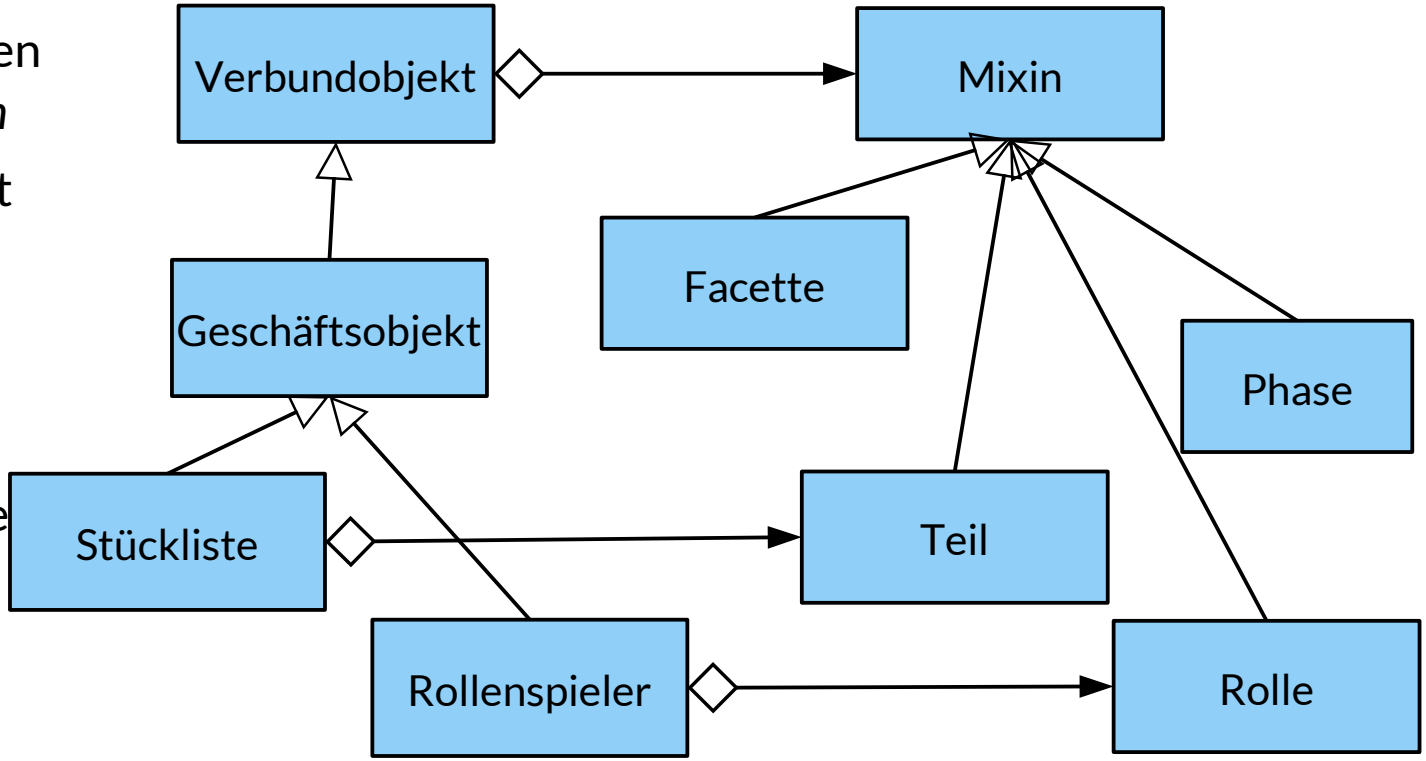
Typische Probleme in Geschäftssoftware

- Start des Projekts: 1995, aber IBM stellte 1999 sein Framework wieder ein
- Technische Gründe liegen in Java:
  - Einfache Vererbung von Java erschwert die Wiederverwendung von Code
  - Komplexe Objekte (Geschäftsobjekte) können in Java schlecht repräsentiert werden (keine Endorelationen)
  - Dynamisch sich wandelnde komplexe Objekte können schlecht repräsentiert werden:
    - Dynamische Teile wie Phasen und Rollen können nicht einfach abgebildet werden. Dynamische Anpassung ist zu komplex
  - In Wirklichkeit hat Java Probleme, komplexe *und* dynamische Objekte zu beschreiben. Es braucht dazu Entwurfsmuster, aber auch Mechanismen für große Objekte
  - Java unterstützt keine Komponenten zur besseren Austauschbarkeit
  - Mixin-Anreicherung wird nur über Entwurfsmuster (Multi-Bridge) unterstützt
- Das Folgende stellt Modellierungsmethoden für komplexe Verbundobjekte vor

Verbundobjekte

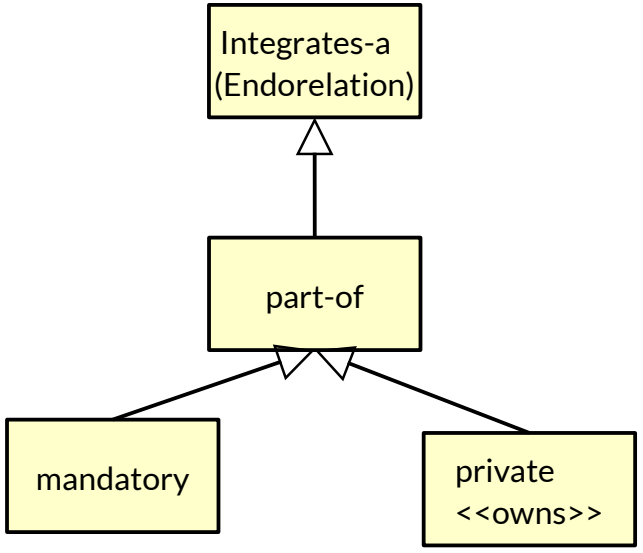
Ein **Verbundobjekt (komplexes Objekt, Subjekt, complex object, compound object)**
ist ein (logisch zusammenhängedes) Objekt,
das auf Programmierniveau
wegen seiner Komplexität durch mehrere (physische) Objekte dargestellt wird.
Seine innere Struktur ist meist hierarchisch, immer aber azyklisch angelegt.
Oft kreuzen Verbundobjekte mehrere Schichten des Systems.

- Verbundobjekte definieren also eine *Endo-Assoziation*
- Verbundobjekte leben oft parallel zueinander und kommunzieren oft über Ströme, Senken, Kanäle
- **Stücklisten** haben Teile; **Rollenspieler** haben Rolle **Geschäftsobjekte** haben beides.



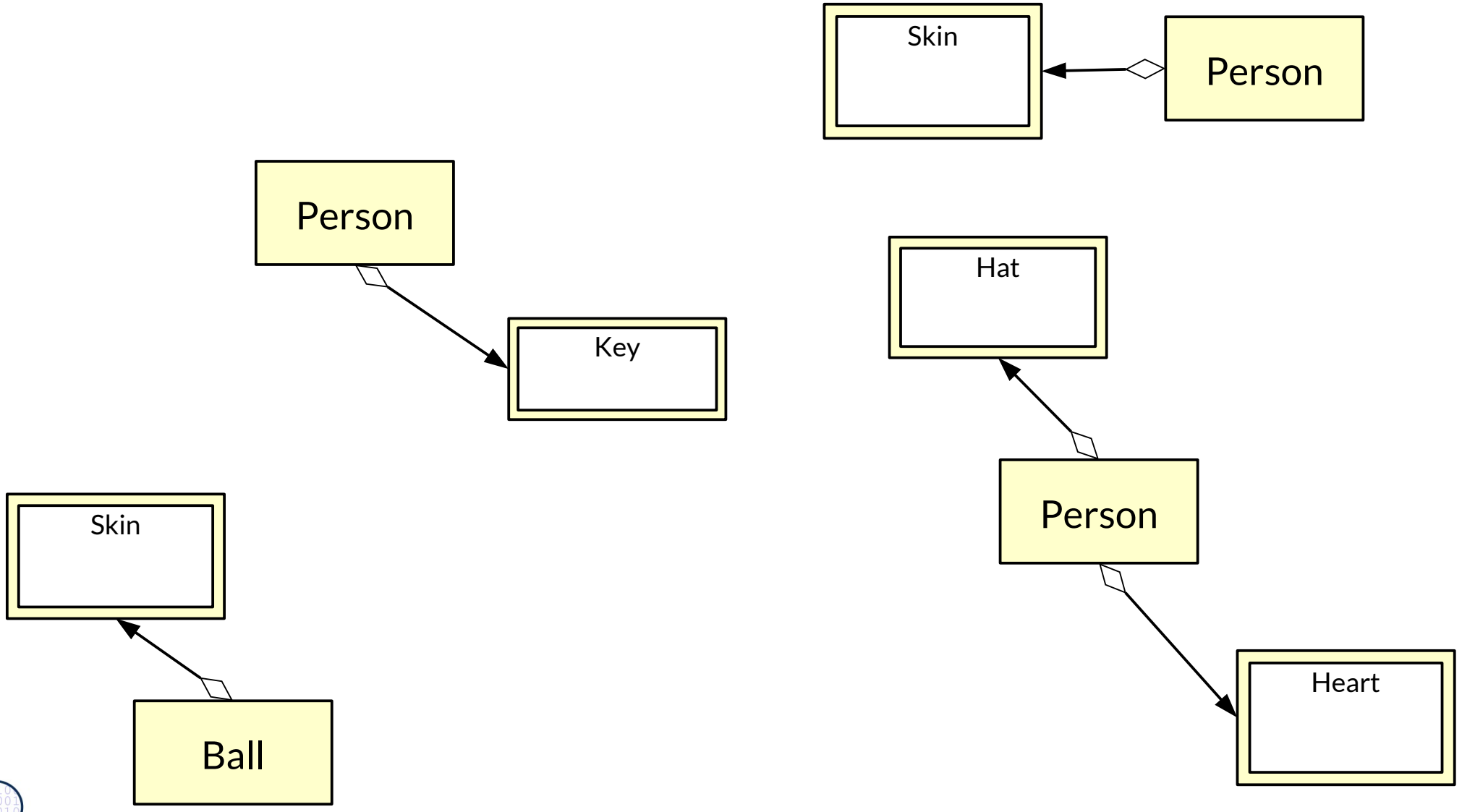
32.2.1 Modellierung von Stücklisten mit privaten und essentielle Teilen von komplexen Objekten

- z.B. in SAP-Systemen
- “Mereologie” https://de.wikipedia.org/wiki/Mereologie



Beispiel: Private und essentielle Teile

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



Private Teile für komposite Objekte

Geteilte Teile

- Ein geteiltes Teil (shared part) ist in mehrere Ganze eingebettet und gehört zu mehreren Kernobjekten. <<shared>>

Private Teile (Eigentumsbeziehung):

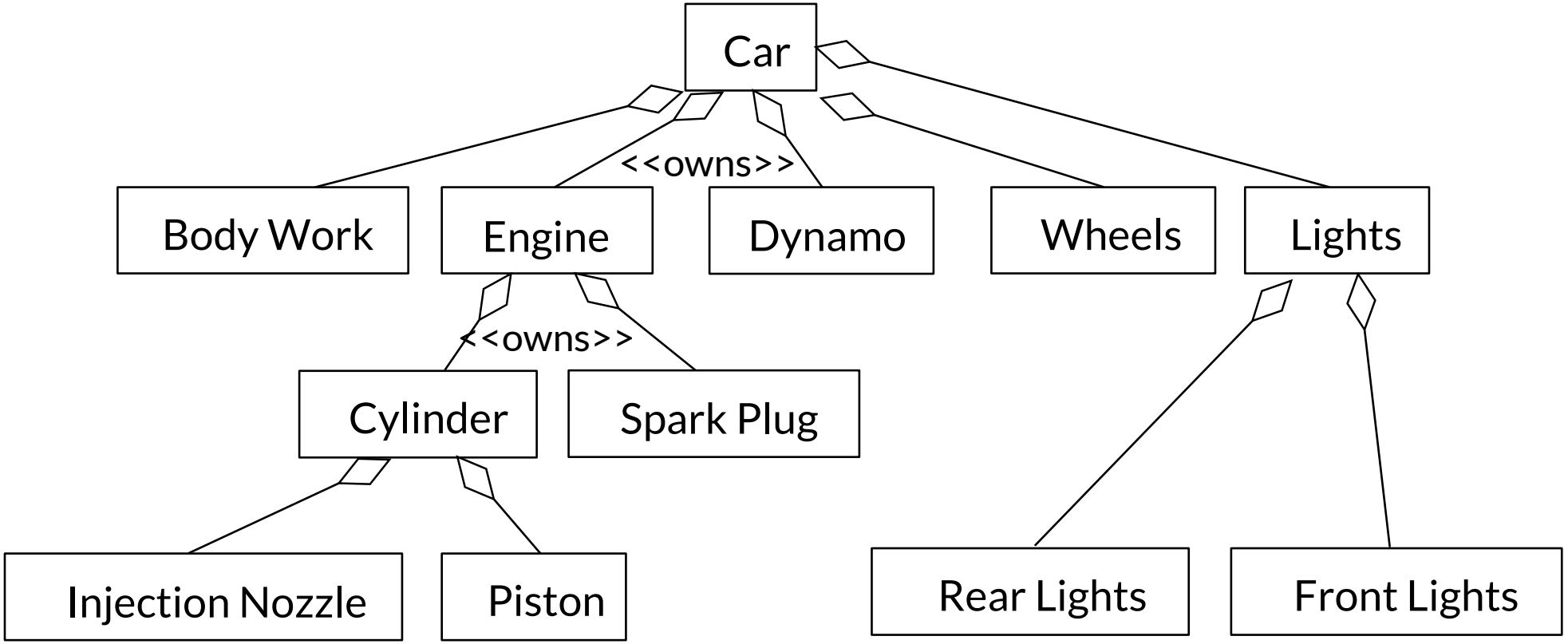
- Ein **privates Teil (owned part)** ist ein nicht-fundiertes Unterobjekt, das ausschließlich zu einem Kernobjekt gehört
  - Beispiel: Eine Person hat einen Hut
  - Zu einer Zeit hat das Teil genau einen Eigner (alias-freie Ganz/Teile-Beziehung), aber das Teil kann das Ganze wechseln
  - Stereotyp <<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>>

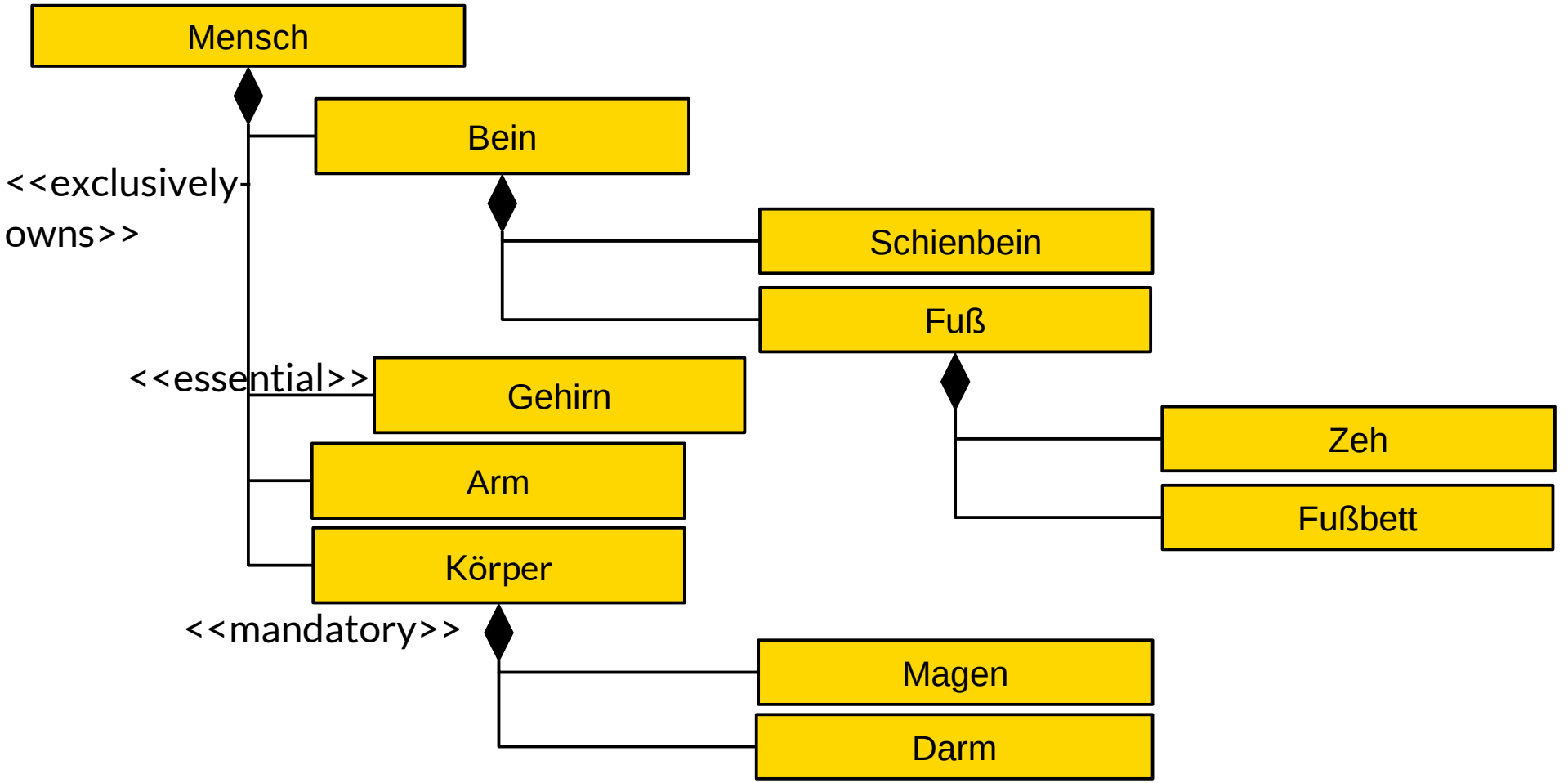
Hierarchische Systemzerlegung mit privaten Teilen (Endo-Assoziation <<owns>>)

- Bei privaten Teilen (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

- <<exclusively-owns>> kann durch <<essential>> und <<mandatory>> spezialisiert werden
- Aggregationshierarchien können als Zeilenhierarchie dargestellt werden

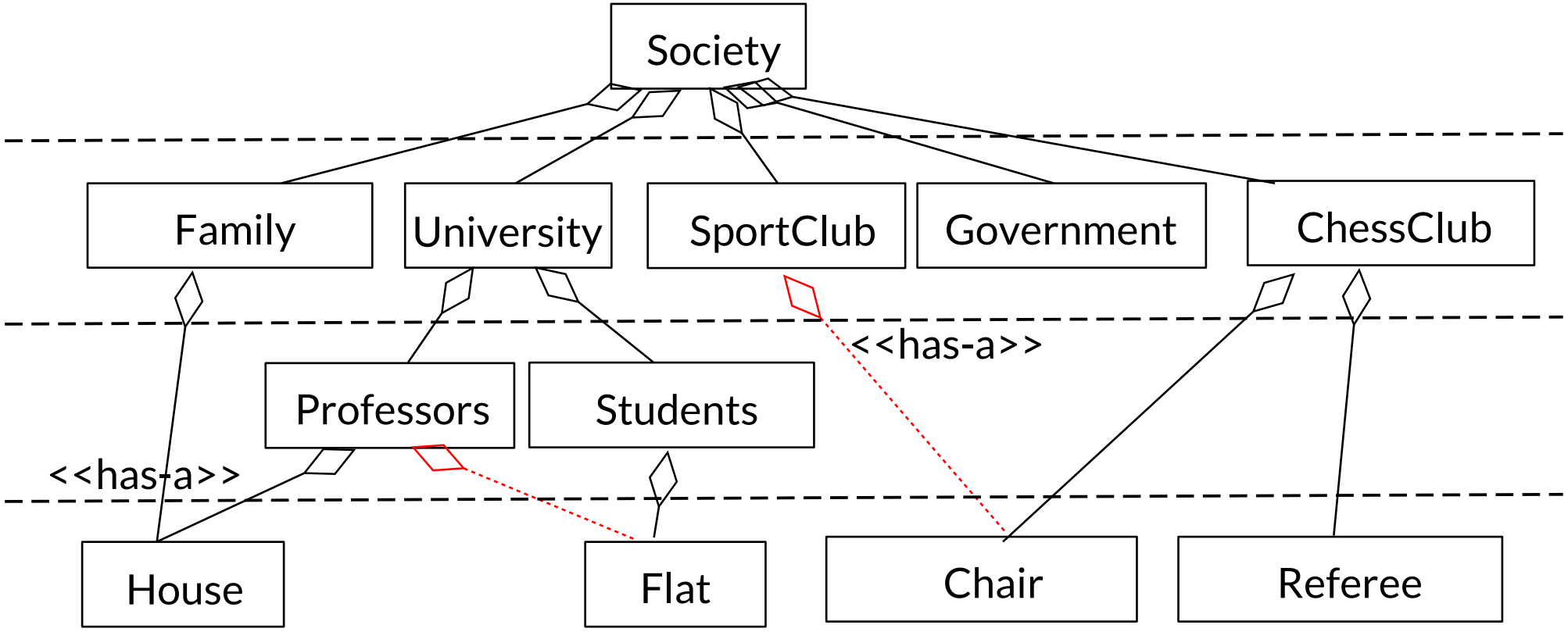


Weitere Ganz/Teile-Beziehungen (Whole-Part Relationships)

- **Eigentumsbeziehungen** bilden baumförmige Relationen (***Stückliste***)
  - *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

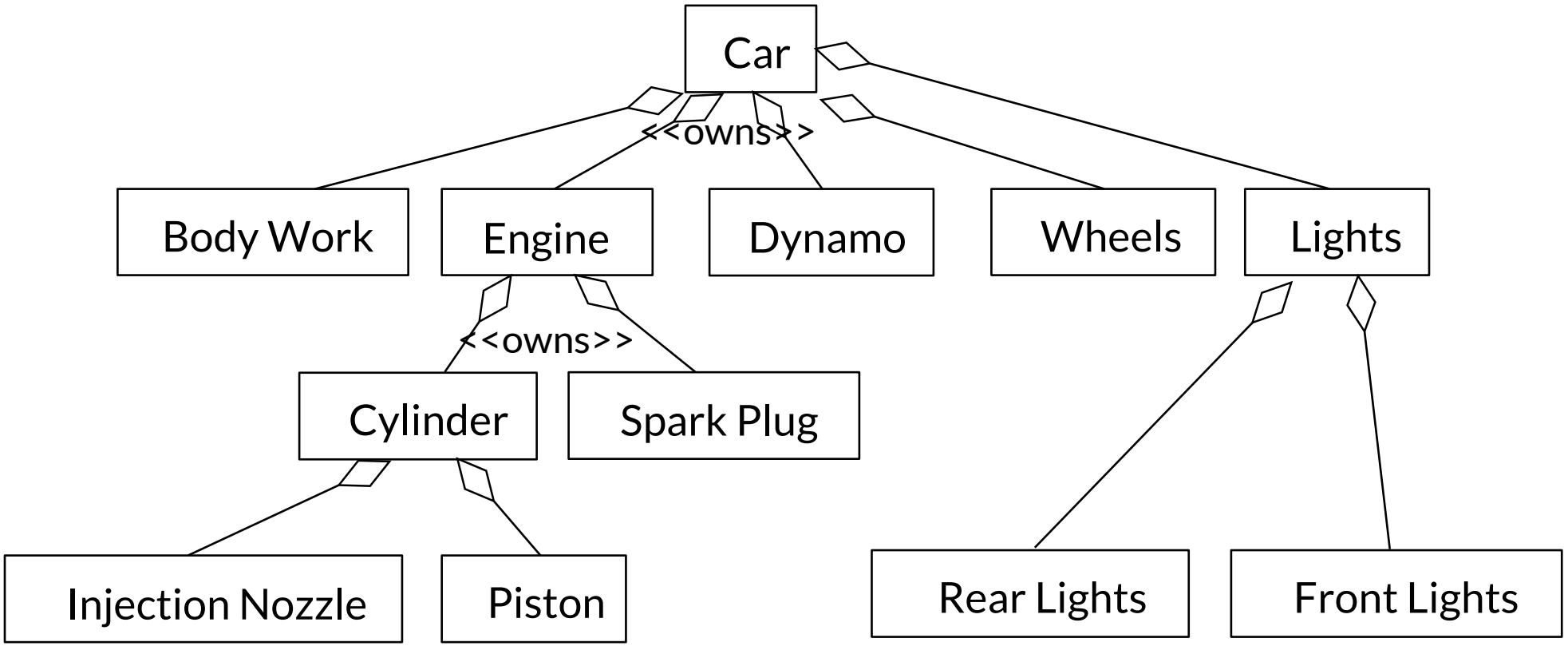
Geschichtete Systemzerlegung mit nicht-privaten, geteilten Teilen (<<has-a>>)

- Das Teilen (*sharing*) von Teilen (parts) erzeugt gerichtete azyklische Graphen (directed acyclic graphs, DAGs), die schichtbar sind (*has-a* Relation)



Teile-Verfeinerung durch hierarchische Systemzerlegung mit statisch bekannter Anzahl von privaten Teilen

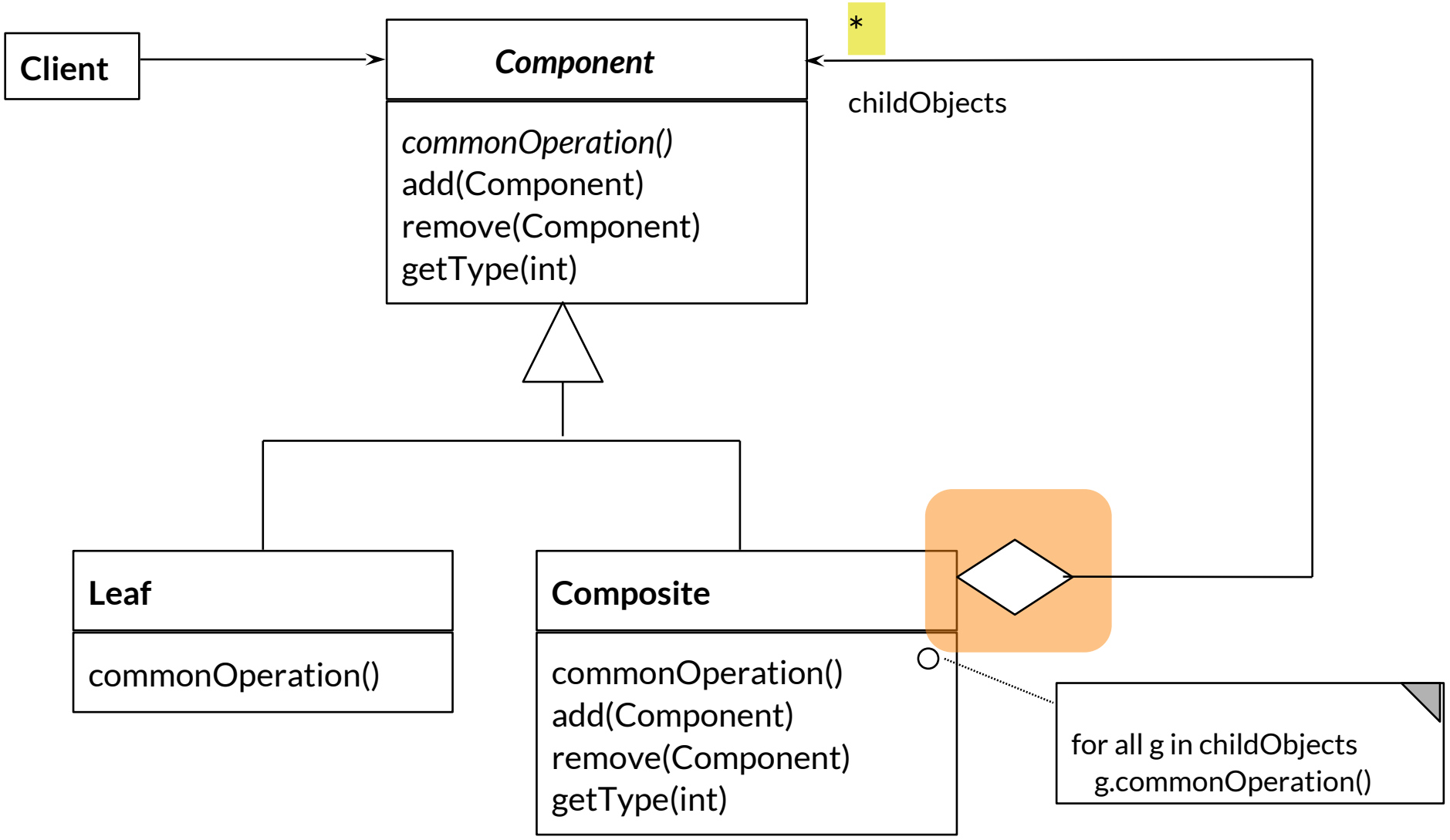
- **Teile-Verfeinerung** beginnt mit den komplexen Ganzen
  - .. und findet Schritt für Schritt neue Teile
- Resultat: Stückliste



https://de.wikipedia.org/wiki/St%C3%BCckliste

Unbekannt viele 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

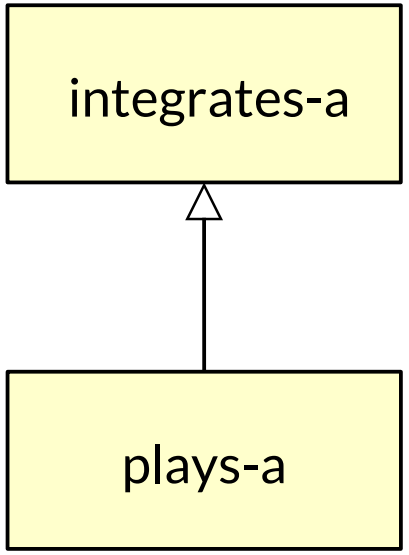
- Materialien in Informationssystemen sind hierarchisch strukturiert

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

Die Verwaltung von Stücklisten ist eine wesentliche Aufgabe von Informationsystemen.

32.2.2 Natürliche Typen und Rollen zur Modellierung von Rollenspielern (und Geschäftsobjekten)

- Geschäftsobjekte in Informationssystemen enthalten Teile und Rollen



DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Verschiedene Arten von integrierten Unterobjekten

- Die verschiedene Arten von Unterobjekten ergeben sich aus einer Matrix von Typqualitäten
  - Hier: Überblick über das Folgende
- Aus allen Arten können Stereotypen abgeleitet werden (Profil “Mixins/Satelliten”)

	Nicht-Fundiert	Fundiert
Rigide	Natürlicher Typ <<natural>> Kern <<core>> Facette <<facet>> Zugeeignetes Teil <<obligatory part>>	Geteiltes Teil <<shared part>>
Nicht-rigide	Phase <<phase>> Privates Teil <<private part>>	Rolle <<role>>

Rigide Typen

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

- Beispiele:
  - *Buch* ist ein rigider Typ; *Leser* ist ein nicht-rigider Typ
  - .. denn 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

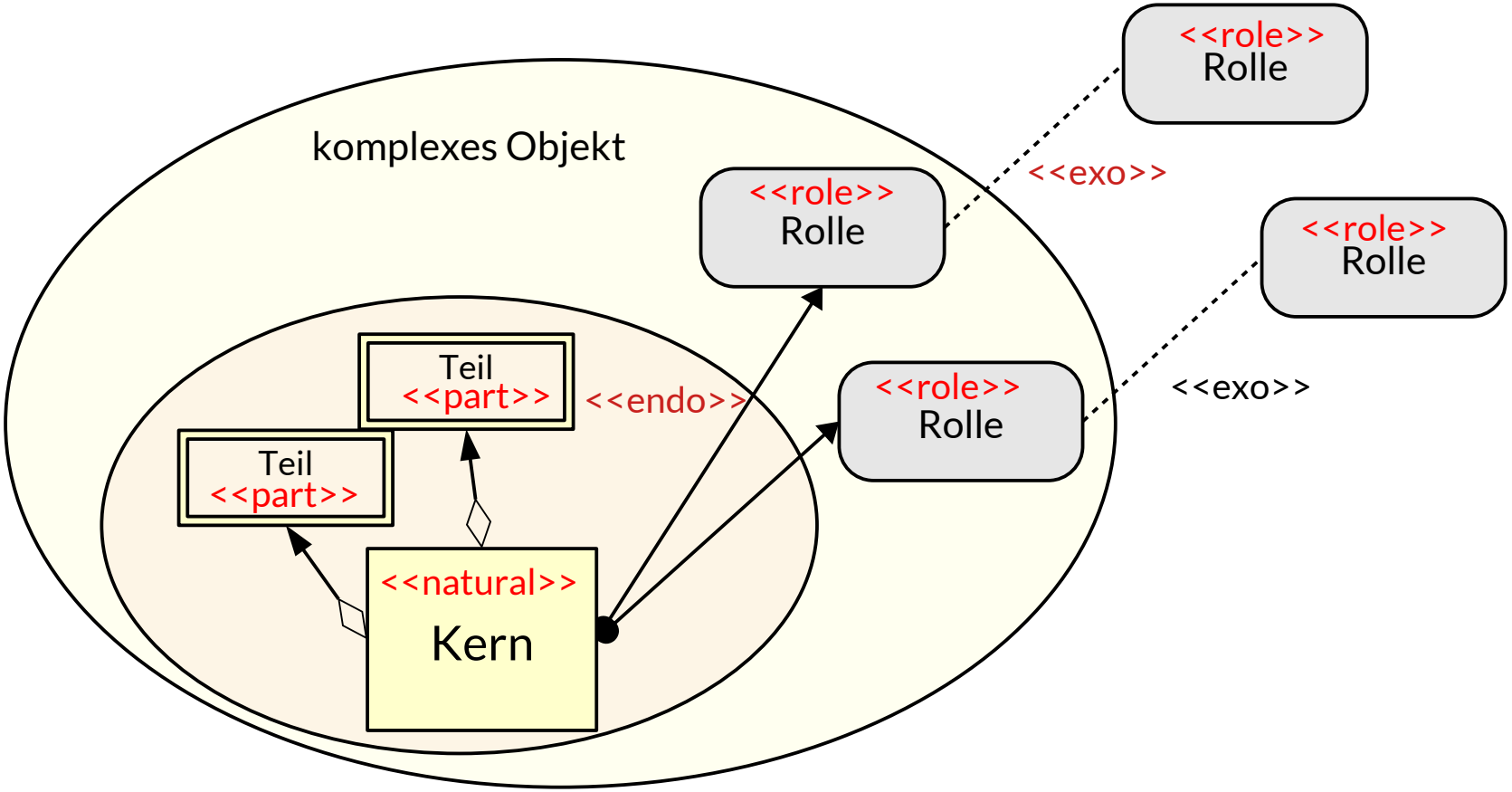
Ein ***natürlicher Typ*** ist ein nicht-fundierter und rigider Typ.

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."
- Unterscheide natürlichen Typen von Rollentypen:
  - *Nicht-rigide Typen* lassen sich immer durch ein Partizip Präsens aktiv beschreiben (im Englischen durch "object" ergänzen):
    - Leser == Lesender, Reader = Reading Object,
    - Forscher == Forschender, Researcher == Researching Object
  - *Rigide Typen* dagegen nicht:
    - Car == ?; Person = ?; Chicken == ?

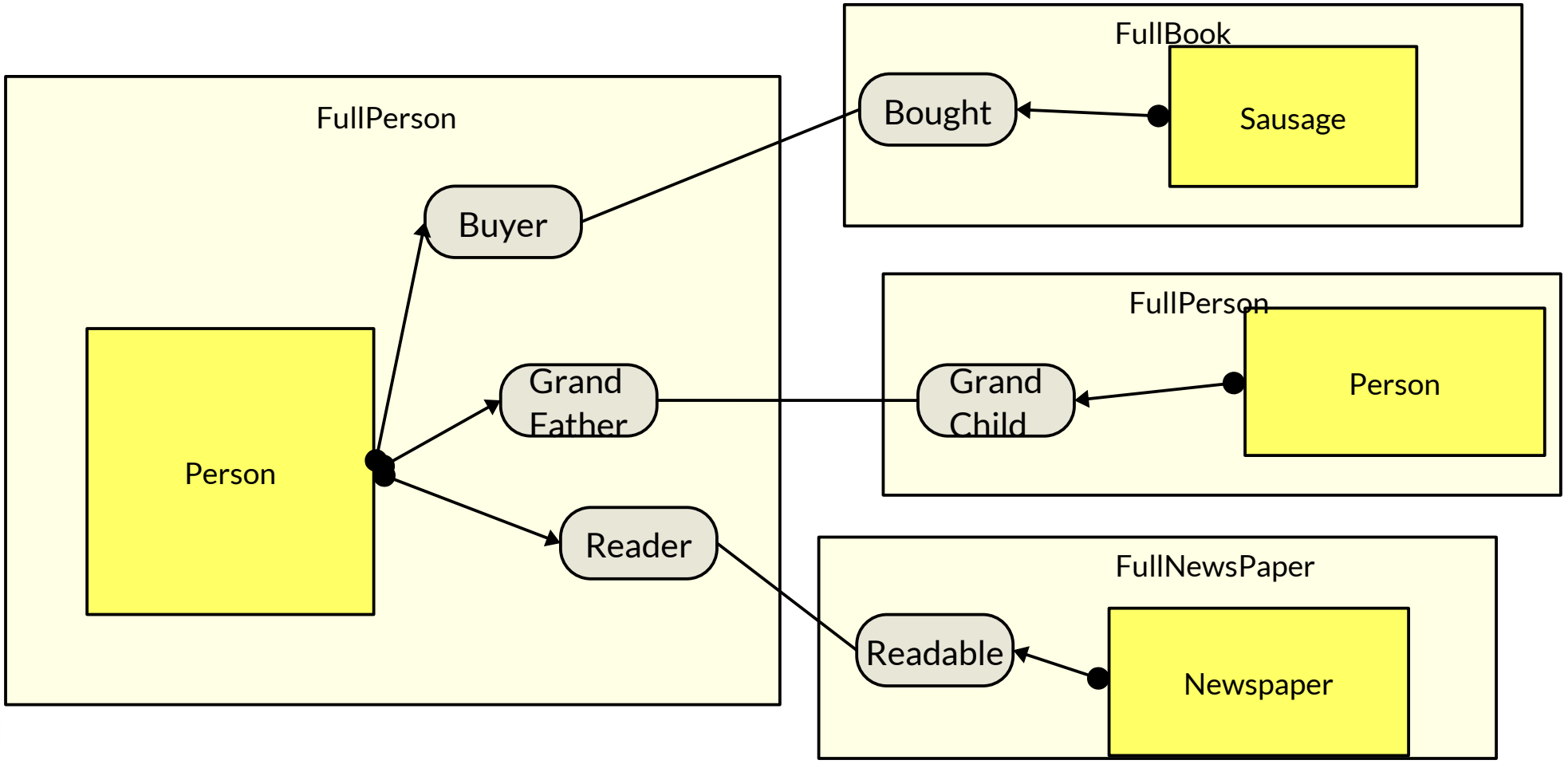
Komplexe Objekte mit Rollen

- Man sagt: ein Objekt *spielt eine Rolle (object plays a role)*
  - Ein Rollentyp (Fähigkeit) wird zur Laufzeit durch ein kontextbezogenes Unterobjekt (RollenMixin) repräsentiert, das in Abhängigkeit von einem Kooperationspartner existiert
  - Rollen sind abhängige Teile, d.h. sie leben nur solange wie das Kernobjekt



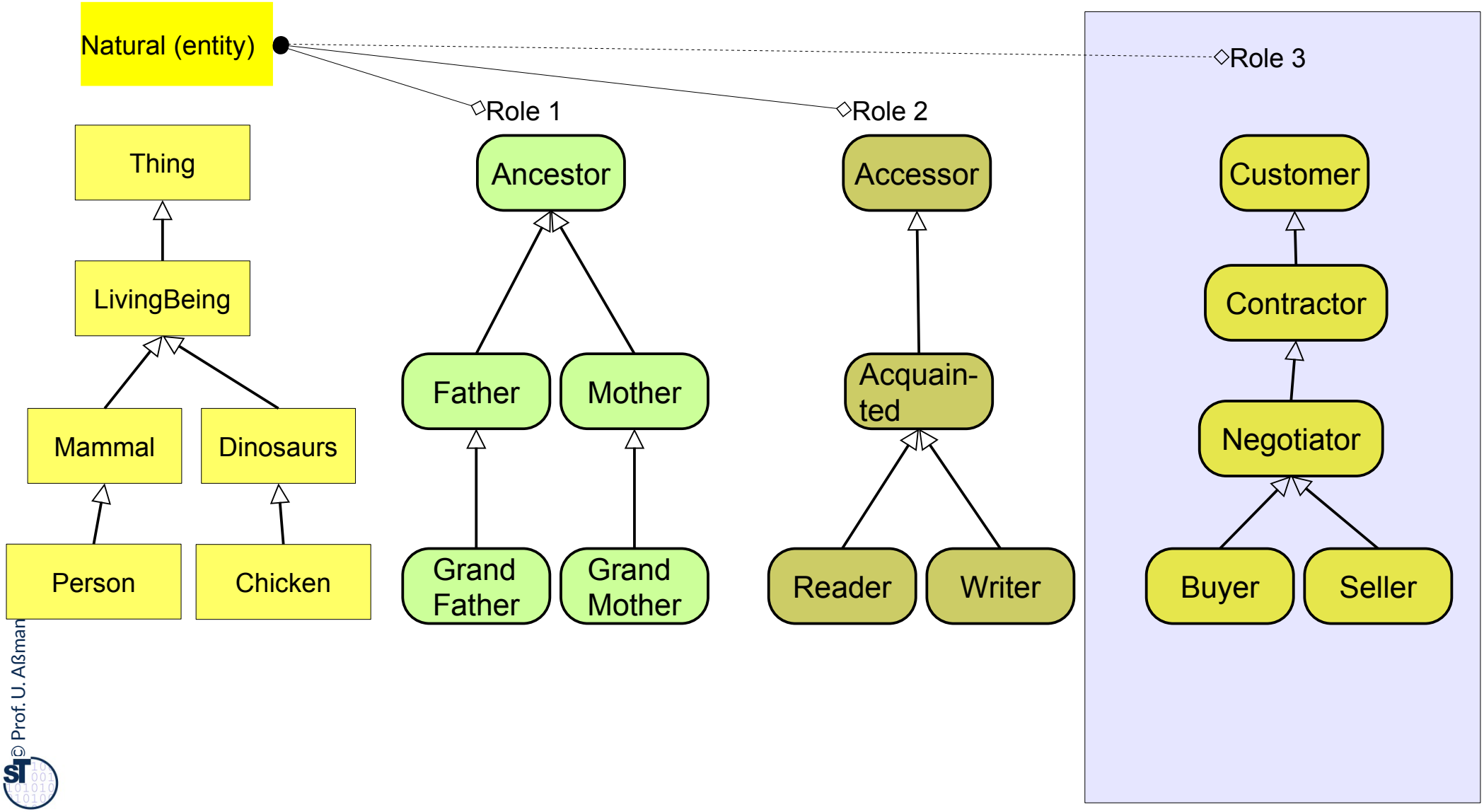
Die Steimannsche Rollen-Faktorisierung [Steiman]

- 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, ..)

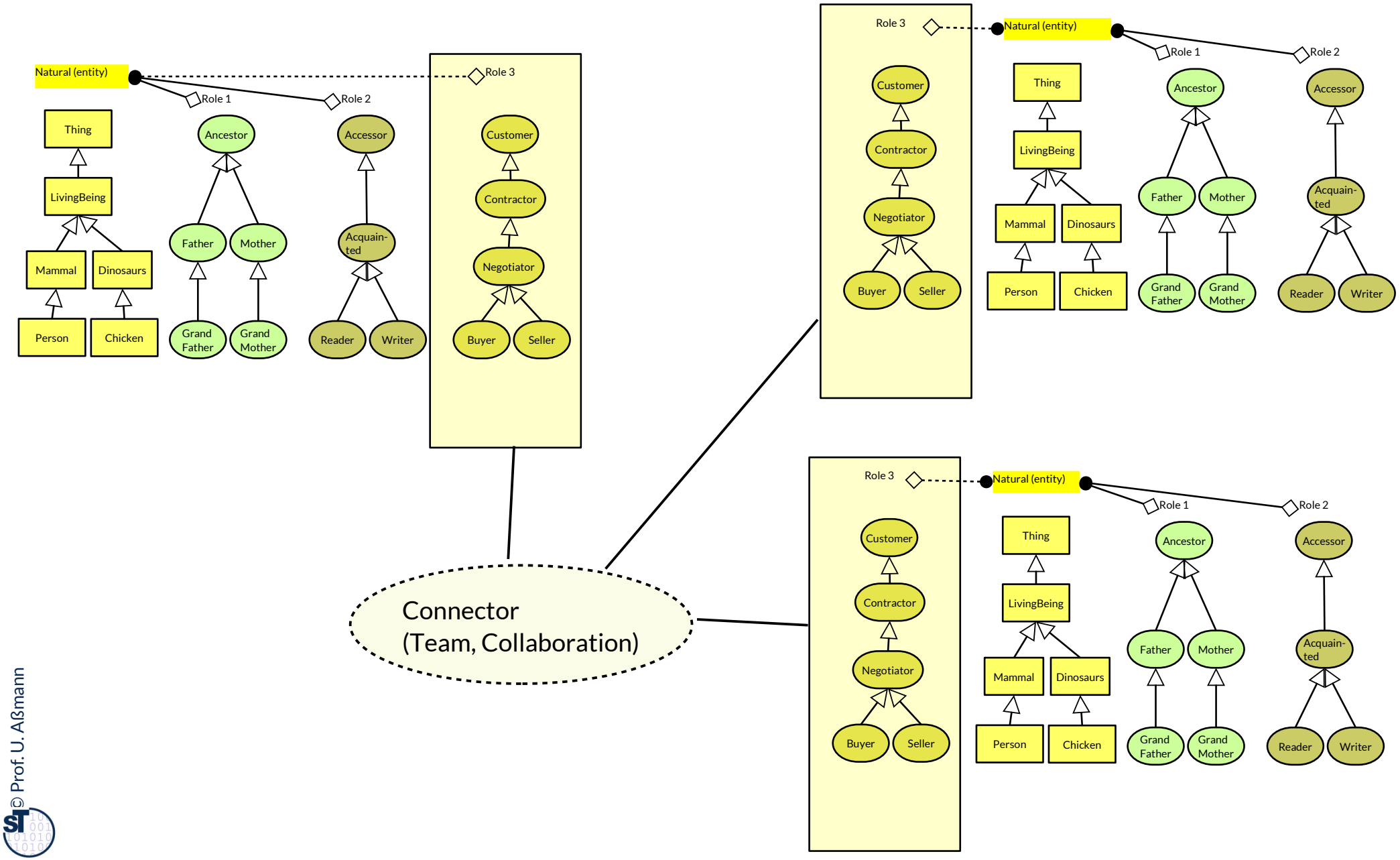


Multidimensionaler Typ für Kern und Mixins

- A new role relationship extends the product lattice by another dimension.

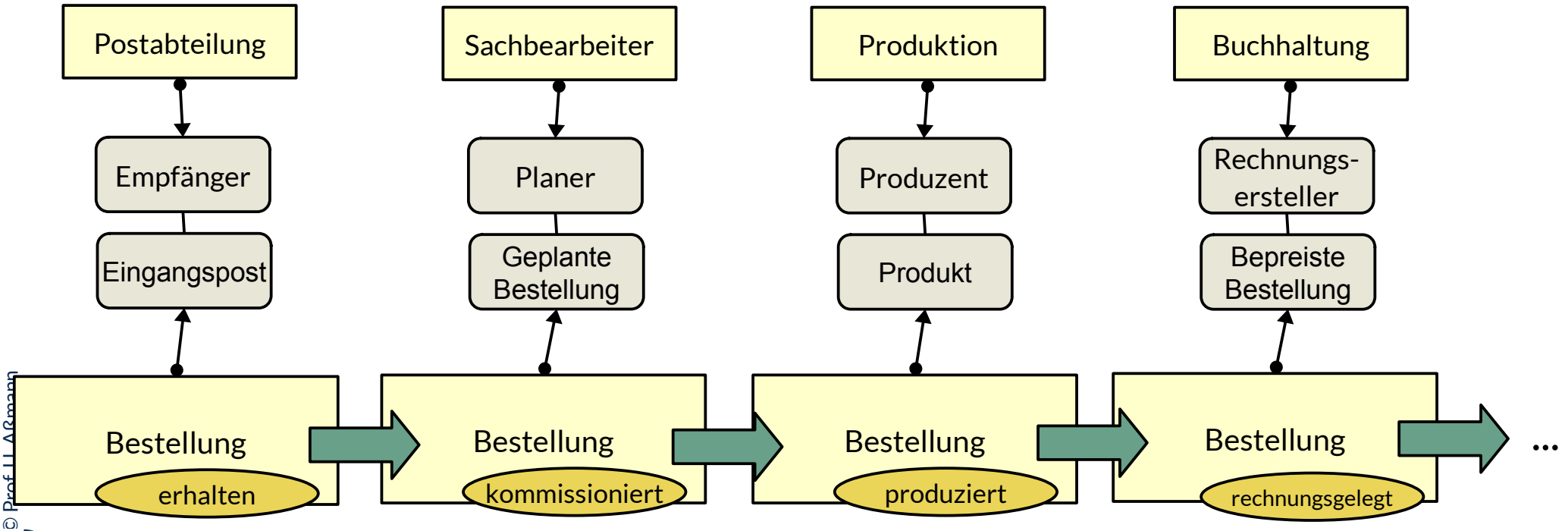


Teams erweitern die Multidimensionalen Typen aller involvierten Verbundobjekte (Anlagerung von Rollen-Mixins)

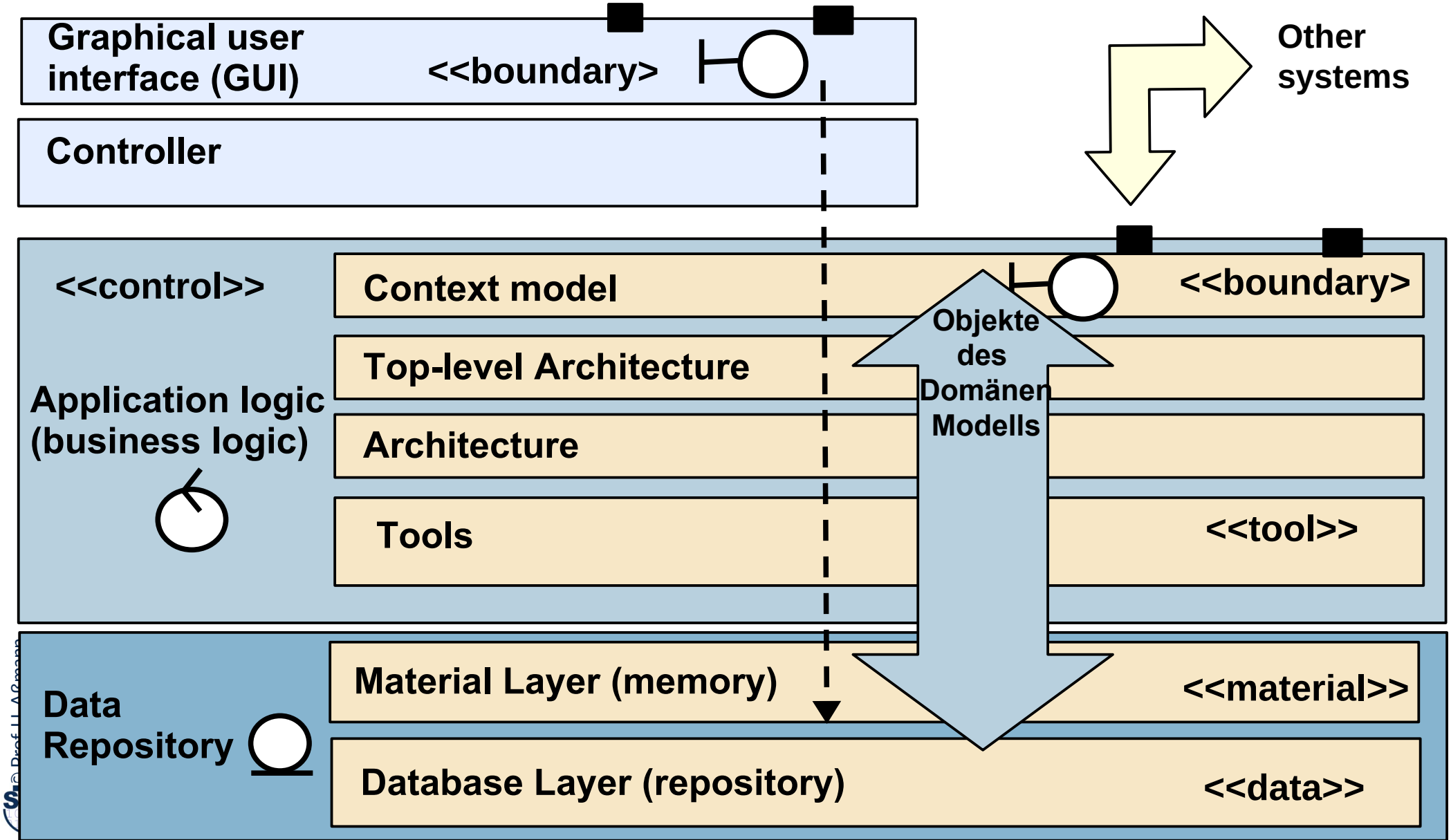


Rollen in Geschäftsobjekten (Business Objects)

- In Geschäftsobjekten kommen immer Rollen- und Teile-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:



Warum komplexe Objekte des Domänenmodells alle Schichten kreuzen



32.3 Realisierung von Teilen und Rollen in Geschäftsobjekten in Informationssystemen

.. mit dem Bridge-Muster

Polymorphie ergibt sich aus dem Wechsel von Rollen-Unterobjekten

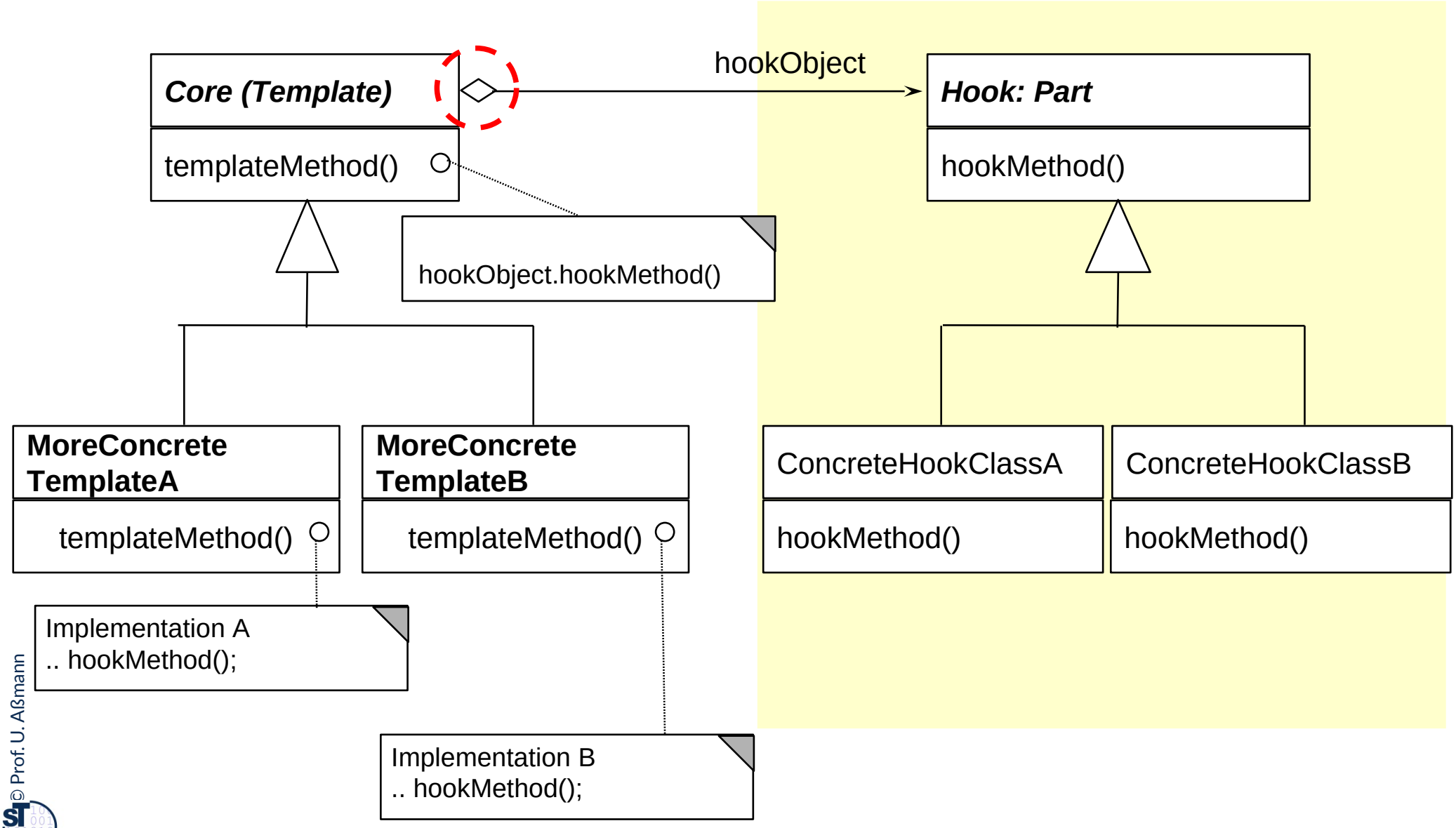
Programm PersonRoles.java

Klassische Realisierung: Unterobjekte mit Bridge-Muster

- Das Bridge-Muster bildet ein Großobjekt ab
  - Abstraction layer (Kern)
  - Implementation layer (Unterobjekt)
  - Es gibt aber Varianten der Bridge, die mit der Teilerelation zwischen Abstraction und Implementation zusammenhängen
- Das ***Multi-Bridge-Muster*** enthält weitere Unterobjekte, d.h. mehrere Bridge
- Welche Arten von Unterobjekten gibt es?

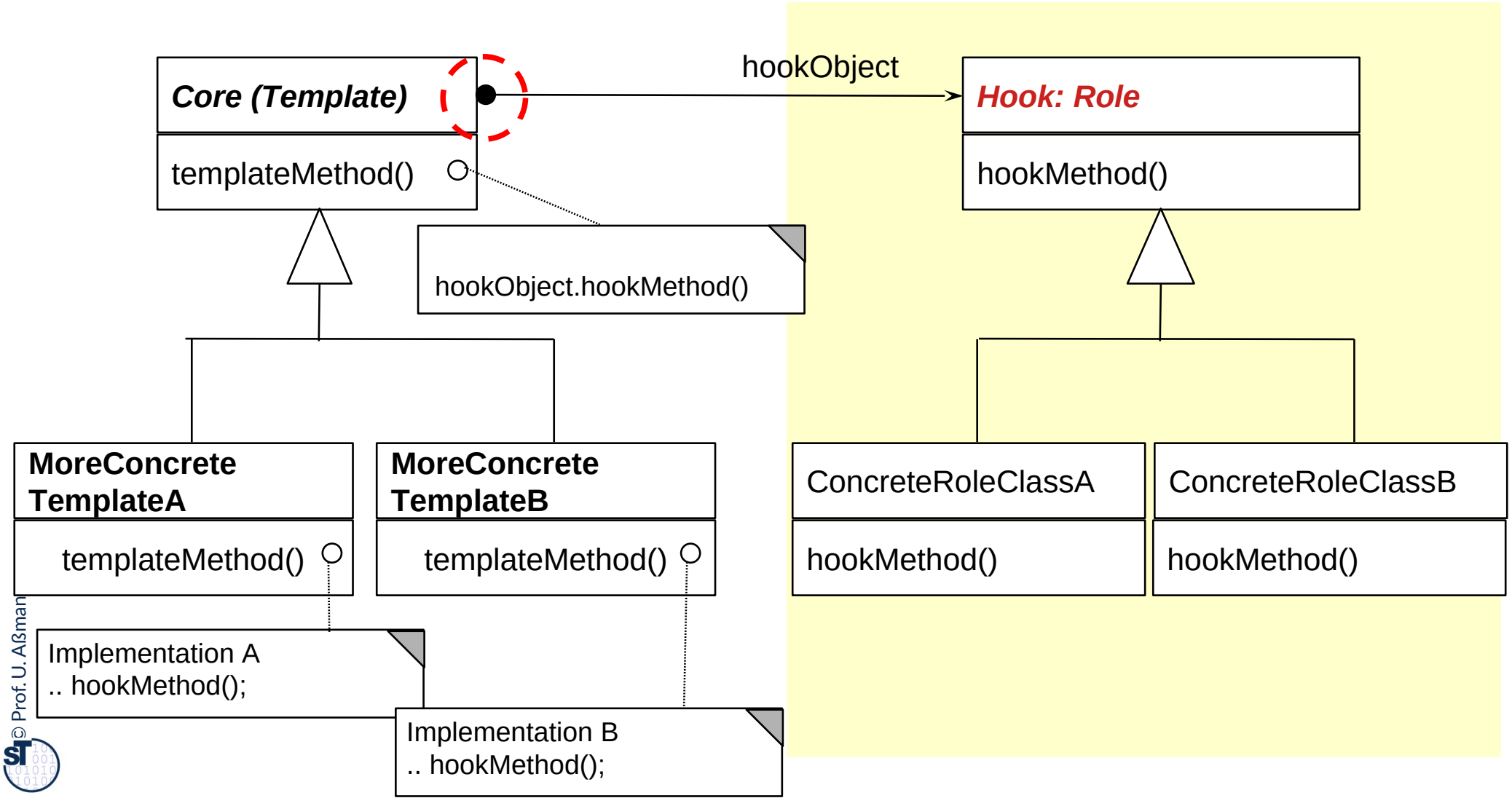
Rept.: Bridge and Dimensional Class Hierarchies

- Bridge implementiert Teile und Rollen in der Dimension “Implementierung”



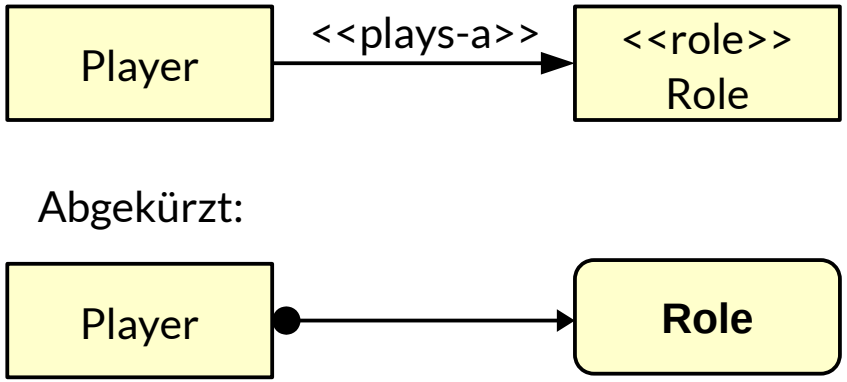
Role-Bridge

- RoleBridge implementiert Teile und Rollen in der Dimension “Implementierung”
- Meistens existieren mehrere Rollen (Multi-Role-Bridge)

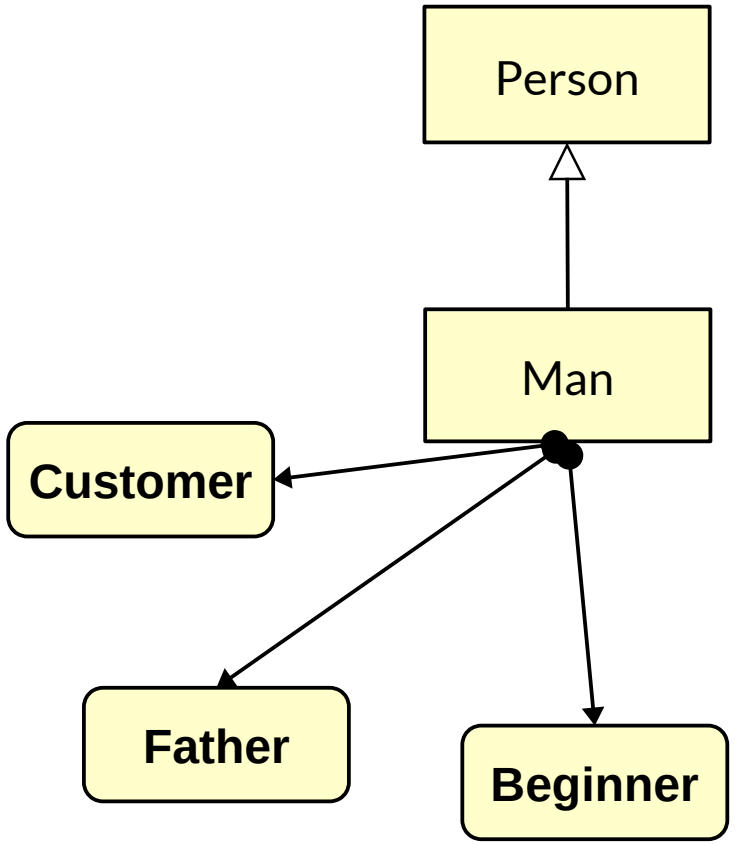


Spezielle Icons für Stereotypen von Aggregation in Brücken

- Für alle grafischen Elemente in UML können spezielle Icone vereinbart werden.
- Für Rollenklassen wählen wir ein Oval, für die Plays-a-Relation einen Pfeil mit Doppel-Kopf-Dreieck.

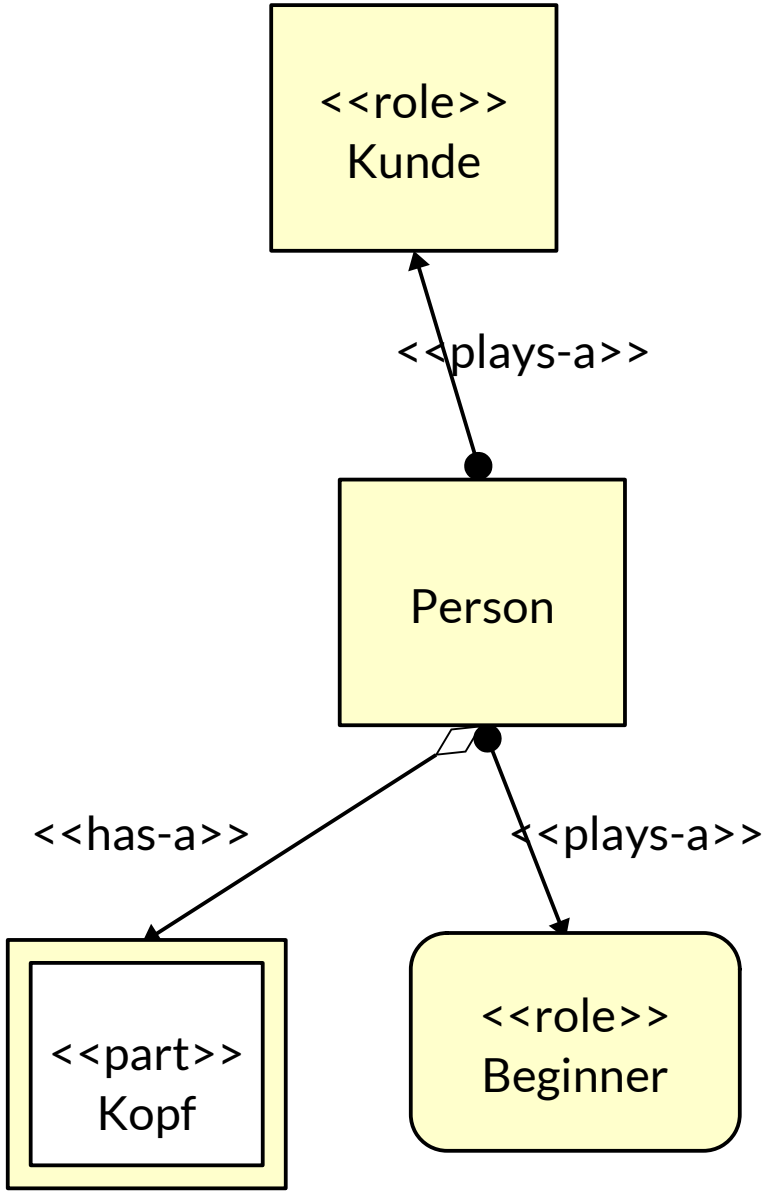


Realisierung mit Multi-Role-Bridge:



Unterobjekte (Mixins)

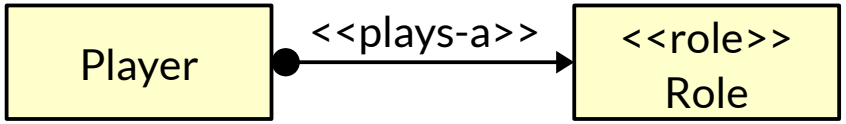
- **Verbundobjekt:** Manche Objekte sind sehr groß und leben lang. Sie werden daher in Unterobjekte gegliedert.
- Ein **Unterobjekt (Mixin, Satellit)** gehört semantisch zu einem **Kernobjekt**
  - teilt also die Identität des Kernobjektes
  - muss diese auf die Frage "Wer bist du?" melden und *nicht den eigenen Identifikator*
  - Hier:
    - ein `Kopf` muss den Identifikator von `Person` melden, *nicht seinen eigenen*
    - ein `Beginner` muss den Identifikator von `Person` melden, *nicht seinen eigenen*

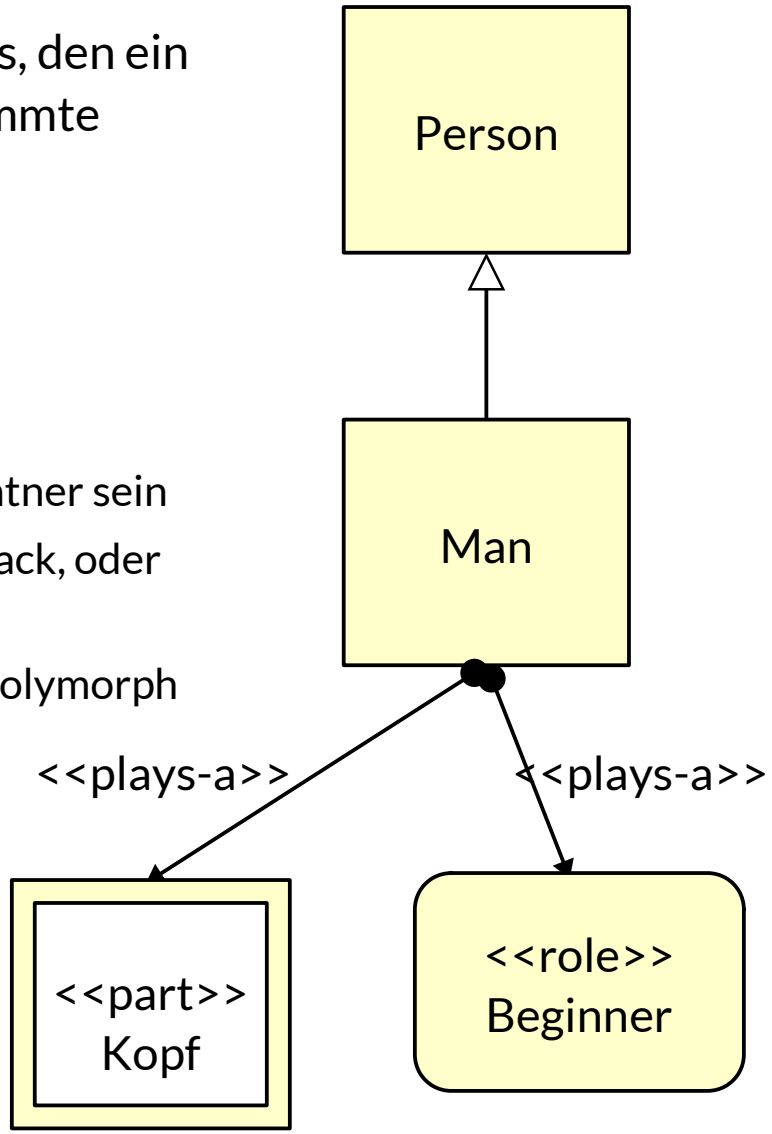


Bug corrected

Rollenwechsel als Grund für Rollen-Polymorphie

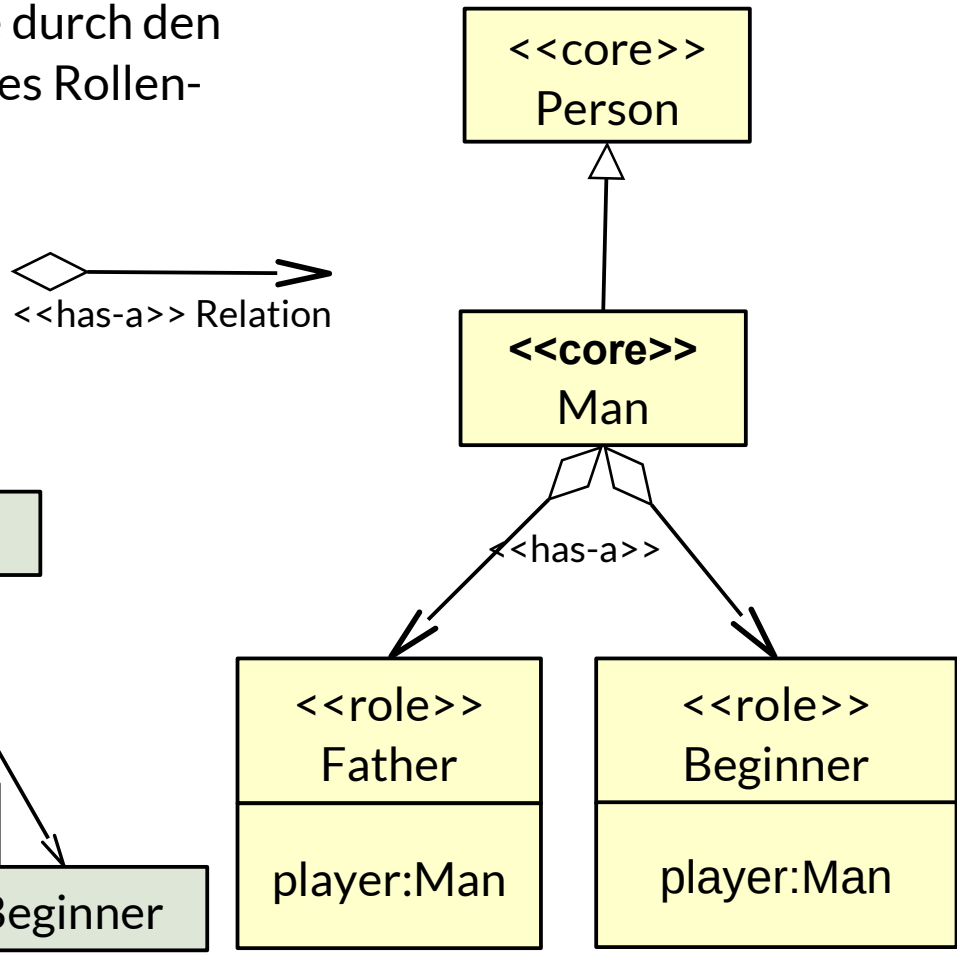
- Ein Spezialfall der Polymorphie ergibt sich, wenn Objekte temporär **Rollen spielen** und ihre Rollen **wechseln**
- Eine **Rolle** ist ein dynamischer Zustand eines Objekts, den ein Objekt für einige Zeit einnimmt und in dem es bestimmte Dienste anbietet
  - Akademiker spielen verschiedene Rollen bez. ihres Studentenlebens (vgl. unser Beispiel)
  - Personen können ledig, verheiratet, geschieden, wiederverheiratet sein
  - Menschen können Kinder, Teenager, Erwachsene, Rentner sein
  - Autos können fabrikneu, Jahreswagen, gebraucht, Wrack, oder Schrott sein
- Weil ein Objekt verschiedene Rollen spielen will, muss es polymorph benutzbar sein!



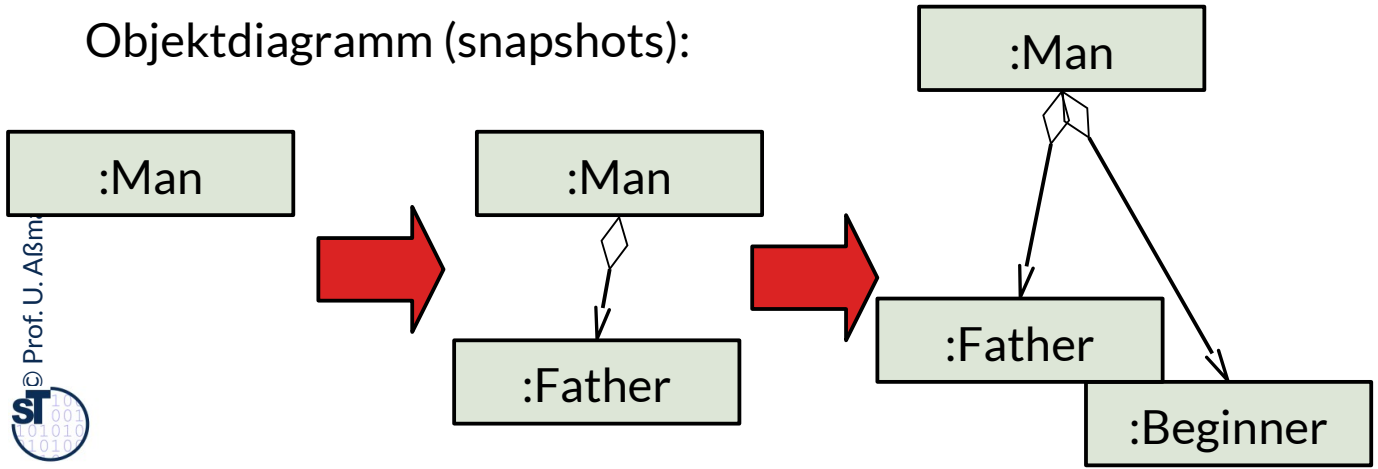


Implementierungsmuster: “Rollen mit Bridge”

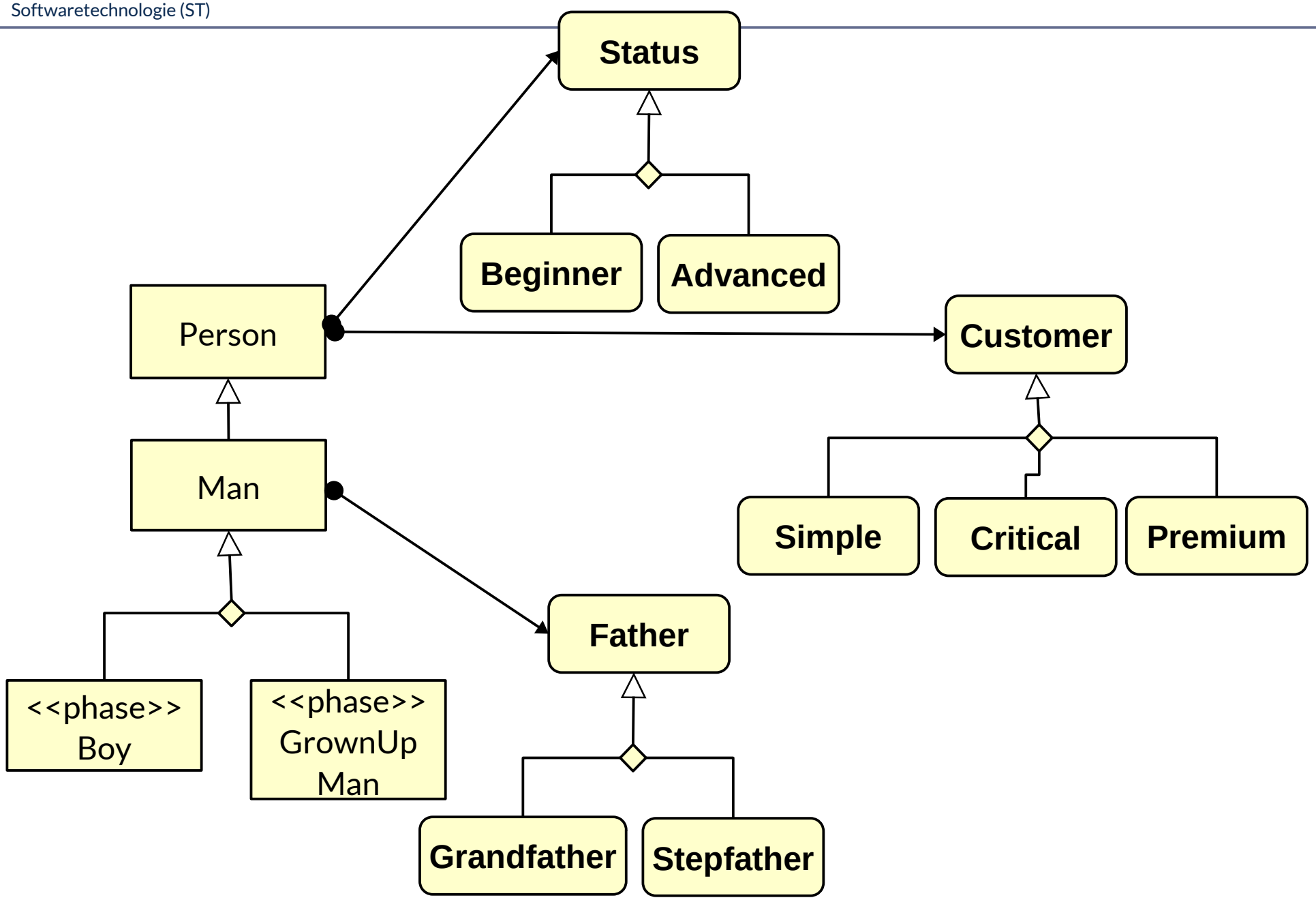
- In Java gibt es leider kein Sprachkonzept “Rolle”; statt dessen können Rollen durch ***Teile*** eines ***Kernobjektes*** modelliert werden (Bridge-Muster, Implementierung von *plays-a* durch *has-a*, Aggregation)
  - Vorteil: Kernobjekt kann viele Rollen spielen
  - Vorteil: Referenzen auf Kernobjekt bleiben bei Rollenwechsel erhalten!
- Man implementiert den Wechsel einer Rolle durch den Wechsel der entsprechenden Unterklasse des Rollen-Unterobjektes durch
  - Alloziere und fülle neues Rollen-Unterobjekt
  - Setze Variable um auf neues Objekt
  - Dealloziere altes Rollen-Unterobjekt



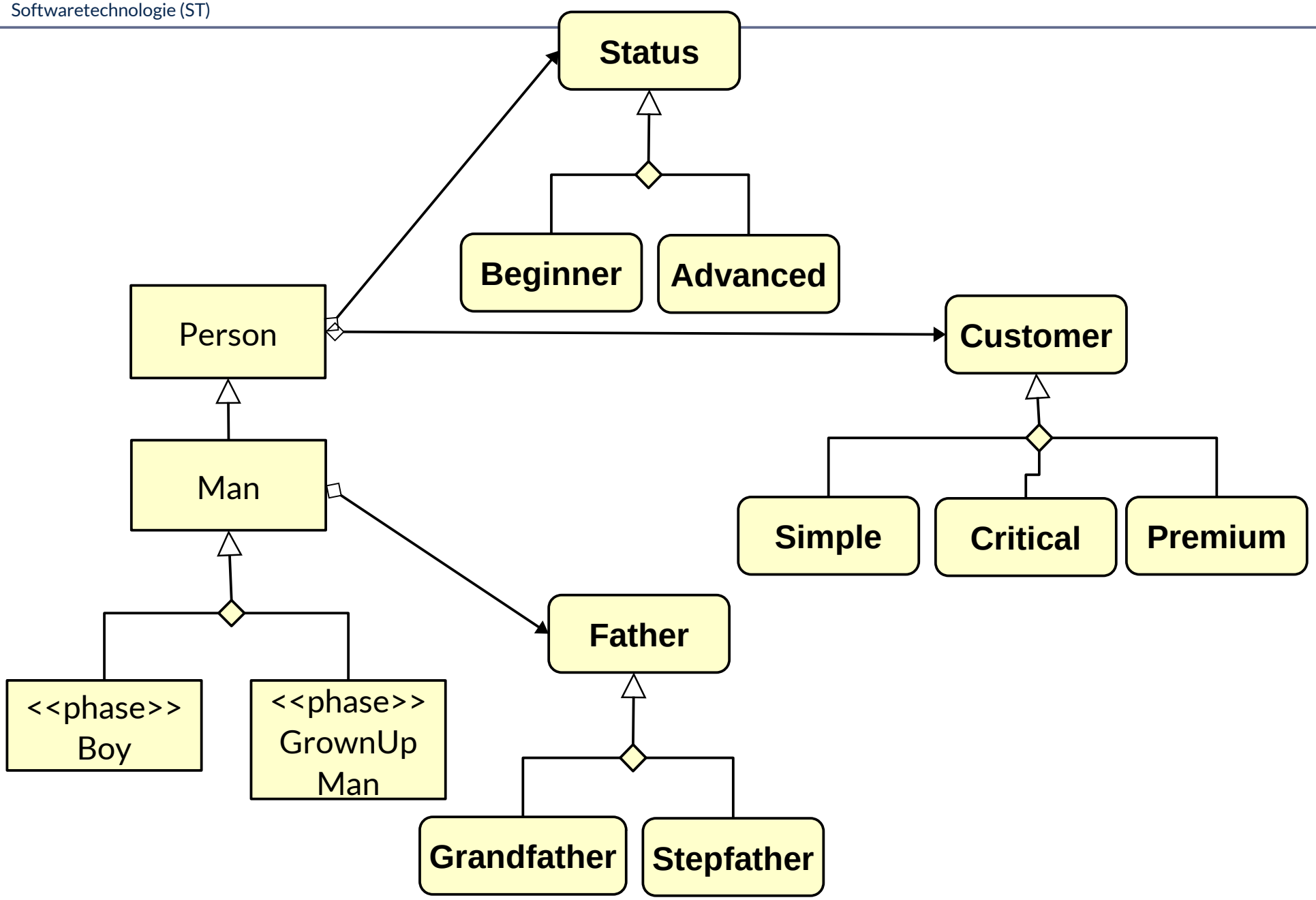
Objektdiagramm (snapshots):



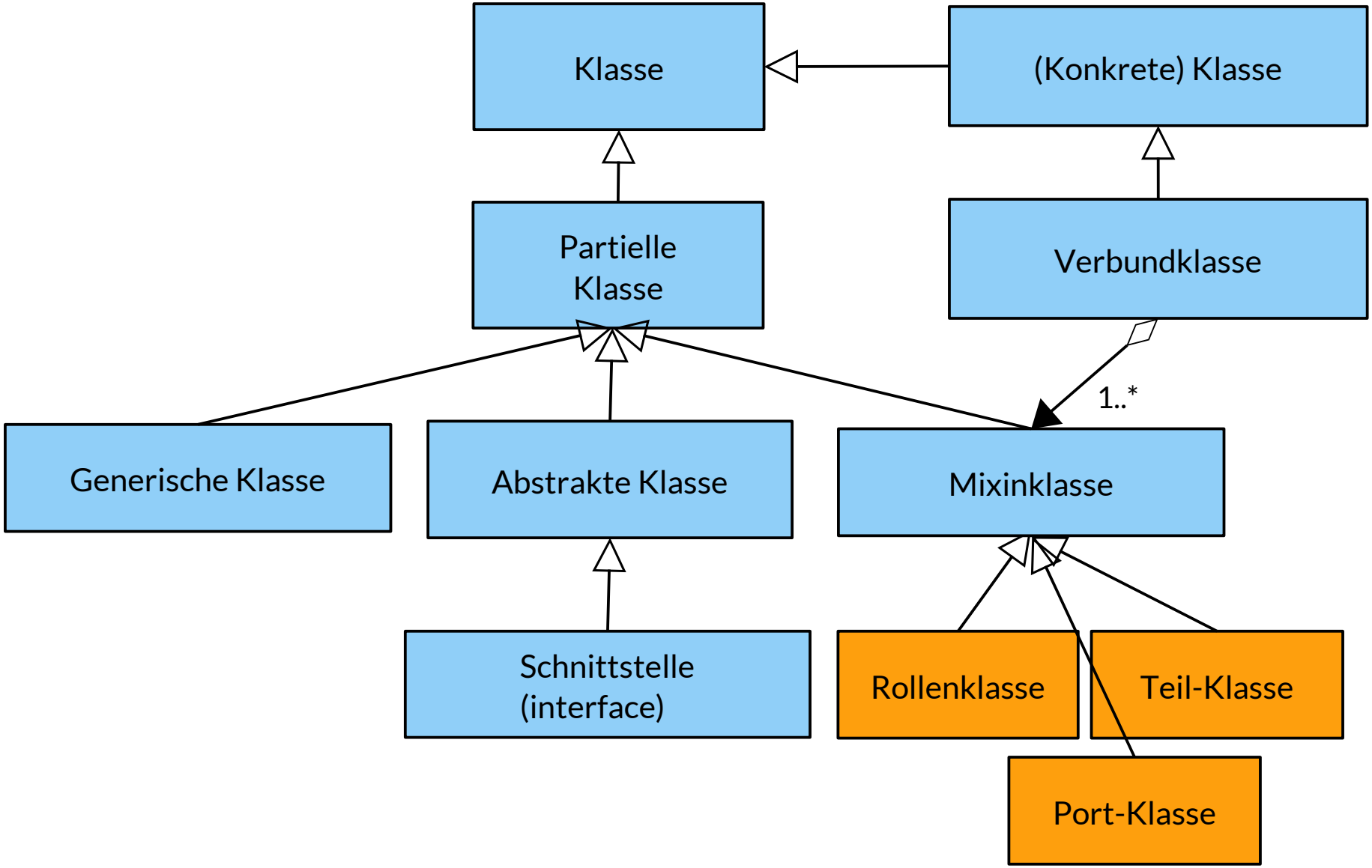
Variation von Rollen mit Multi-Role-Bridge



Implementierungsmuster: Variation mit Multi-Part-Bridge



Begriffshierarchie von Klassen (Erweiterung)

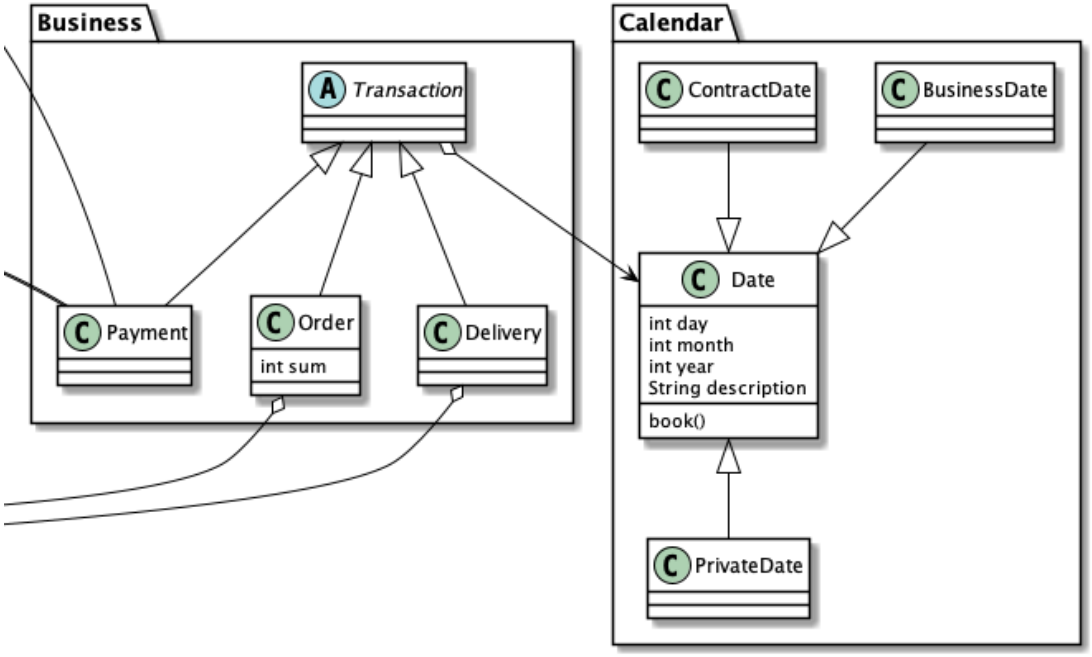


32.3.2 Das Domänenmodell des Kurses

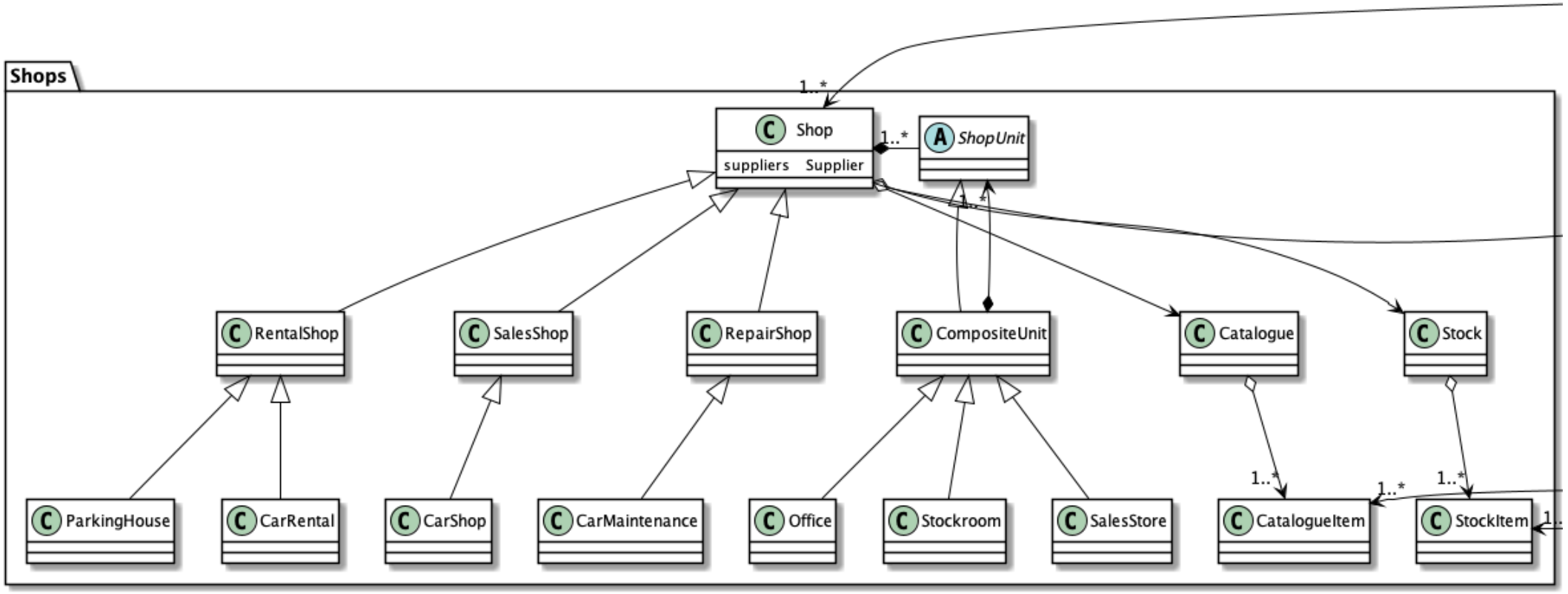
- Die Entwicklung eines Informationssystems beruht auf einem fachlichen Modell der Domäne (Domänenmodell).
- Im Folgenden gibt es einen Überblick, erstellt mit plantUML.

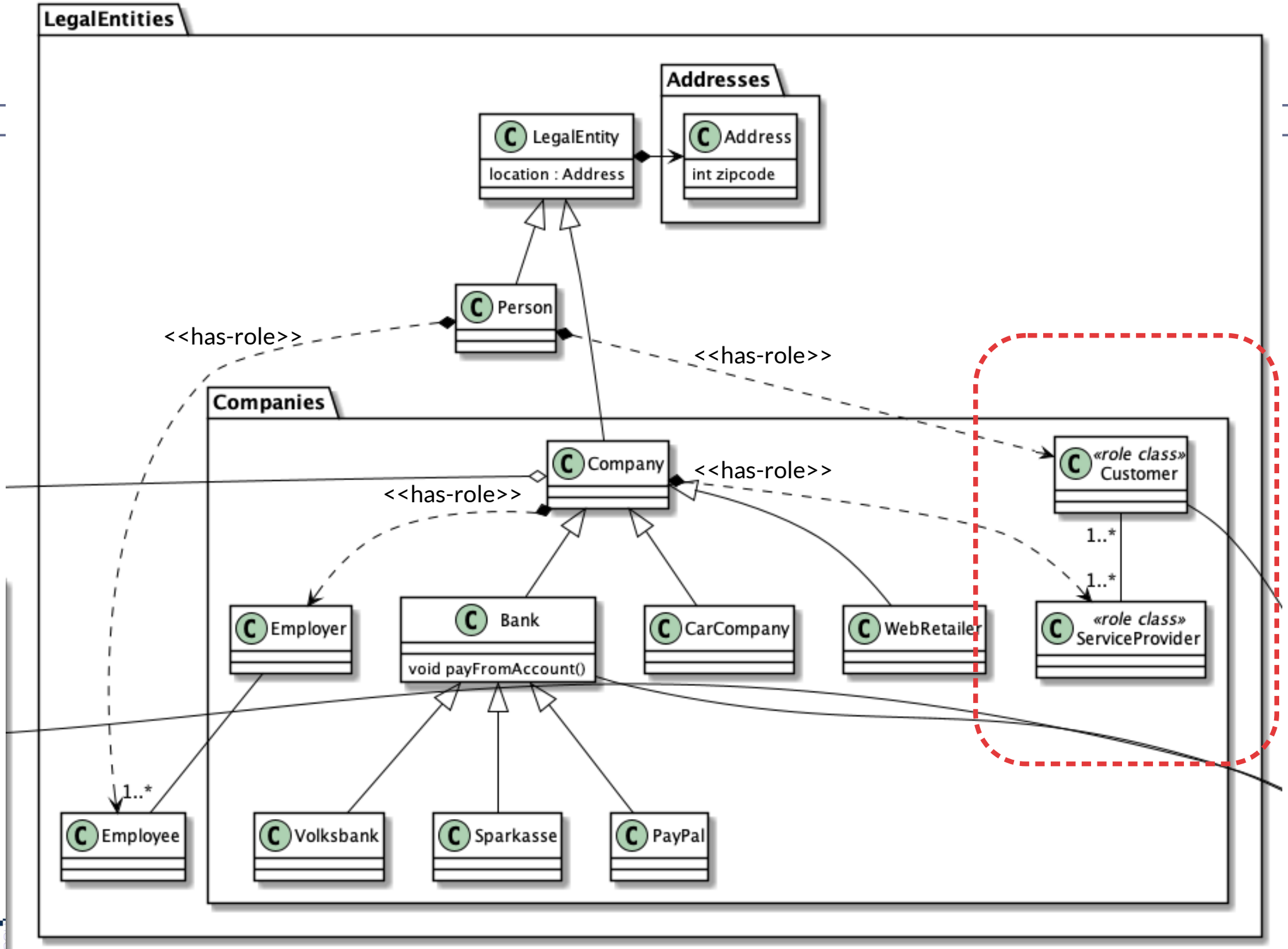
Order, Delivery, Payment

- Order, Delivery, Payment sind komplexe Geschäftsobjekte zwischen mehreren Teilnehmern
  - Assoziationsklassen
  - Kollaborationen (Teams)
- Können als Teams mit Rollen dargestellt werden
  - Hier: einfache Klassen



Shops



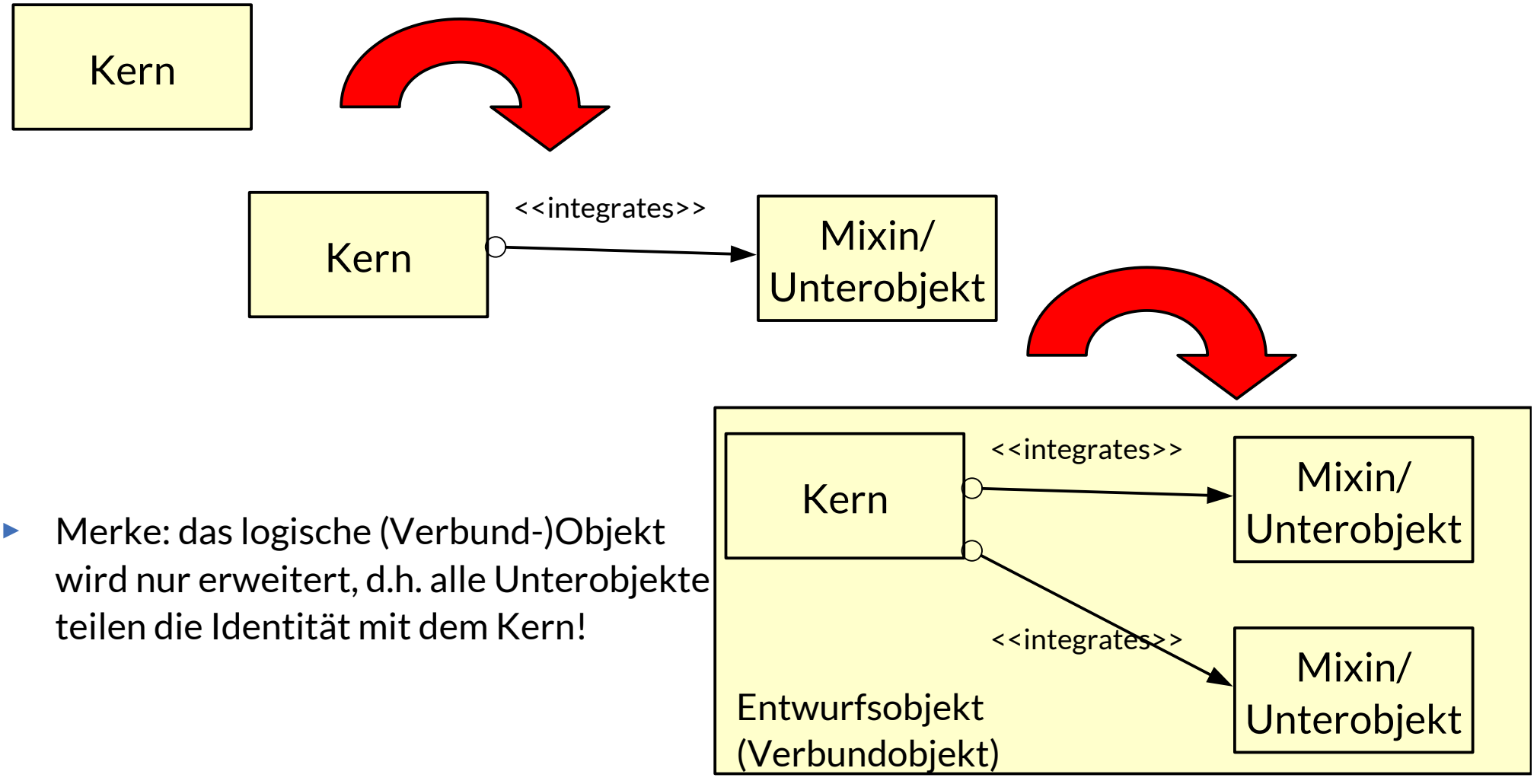


32.4 Punktweise Verfeinerung durch Mixin-Anreicherung mit Teilen und Rollen

- Die Entwicklung eines Informationssystems beruht auf Mixin-Anreicherung durch Teile und Rollen, die in Verfeinerungsoperationen an die initialen Modelle angelagert werden.

Anreicherung von Mixins in Analyseobjekte zu komplexen Objekten

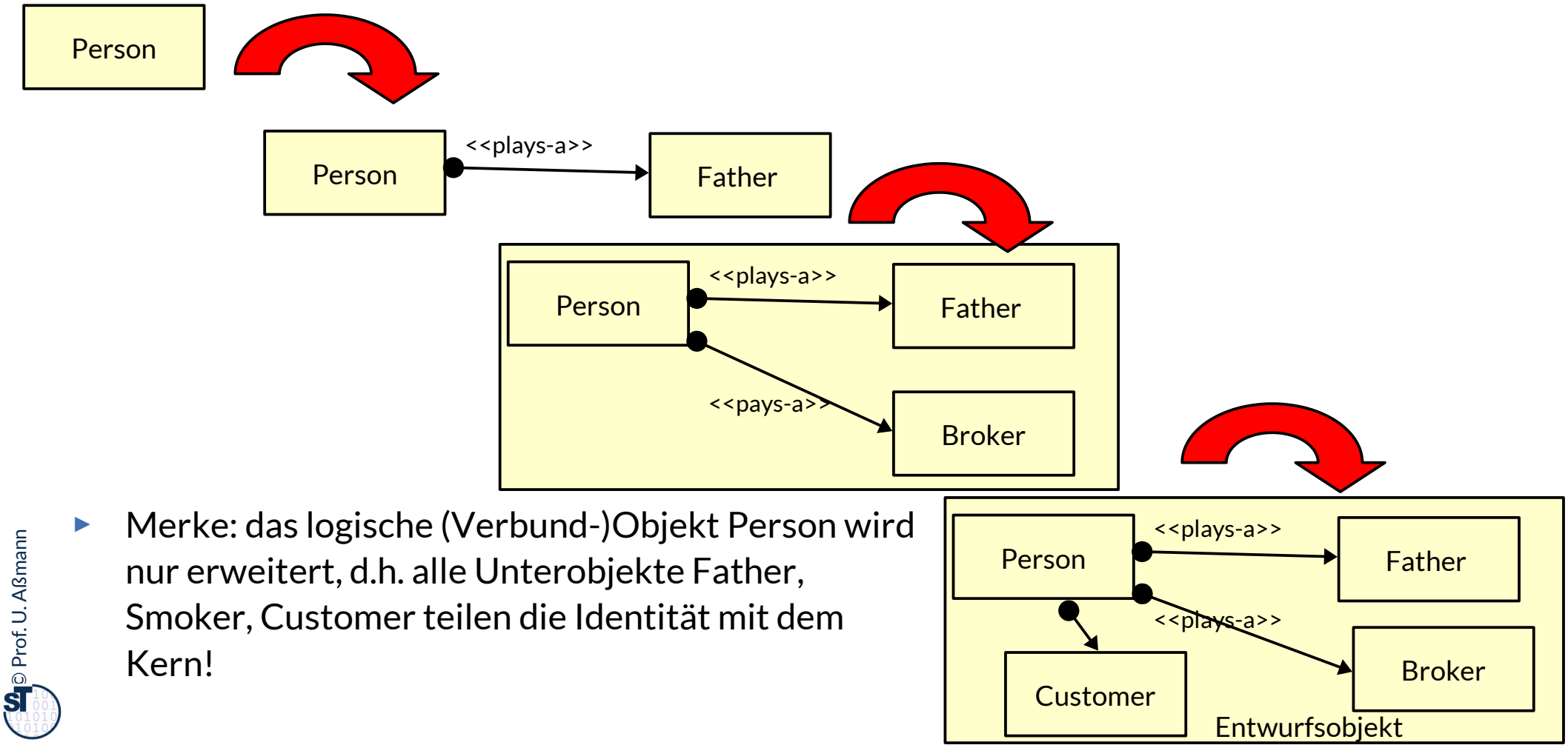
- **Integration von Mixins** besteht aus der Mixin-Anreicherung von Analyseobjekten aus dem Domänenmodell



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

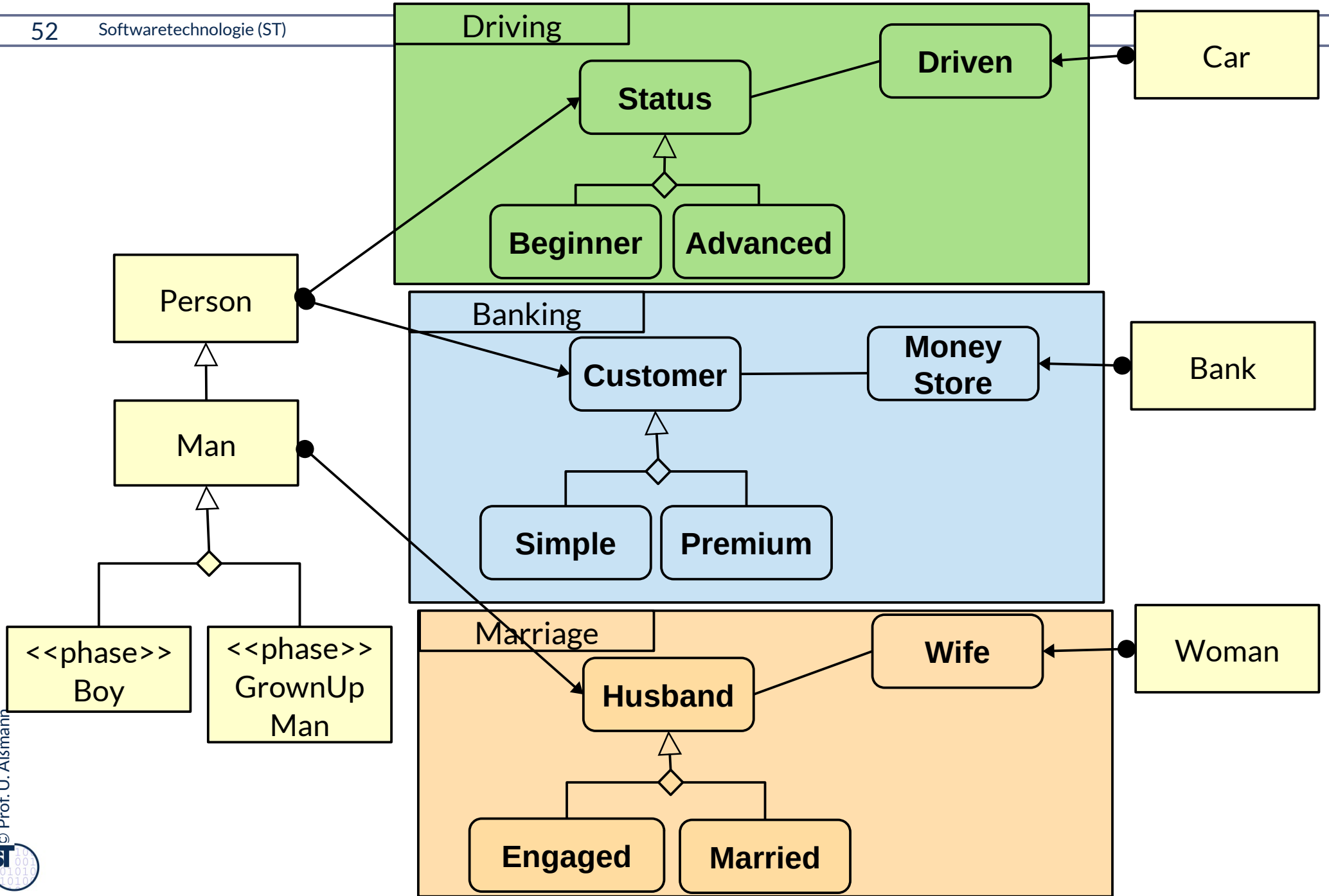
Beispiel: Integration von Mixins in Personen

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

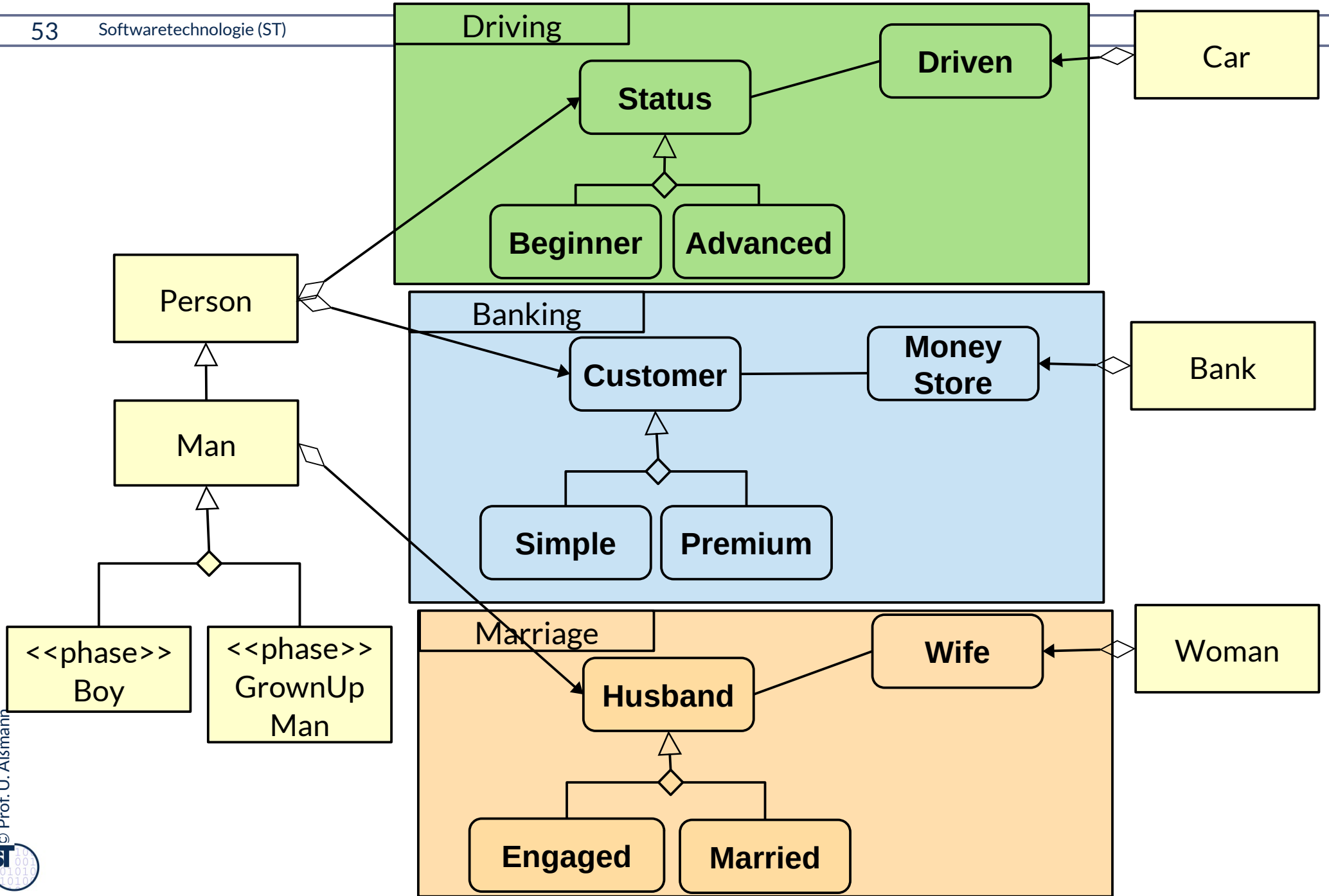


- Merke: das logische (Verbund-)Objekt Person wird nur erweitert, d.h. alle Unterobjekte Father, Smoker, Customer teilen die Identität mit dem Kern!

Beipiel: Querschneidende Erweiterung von Objekten mit Multi-Role-Bridge (Mehrfach-Mixin-Anreicherung)

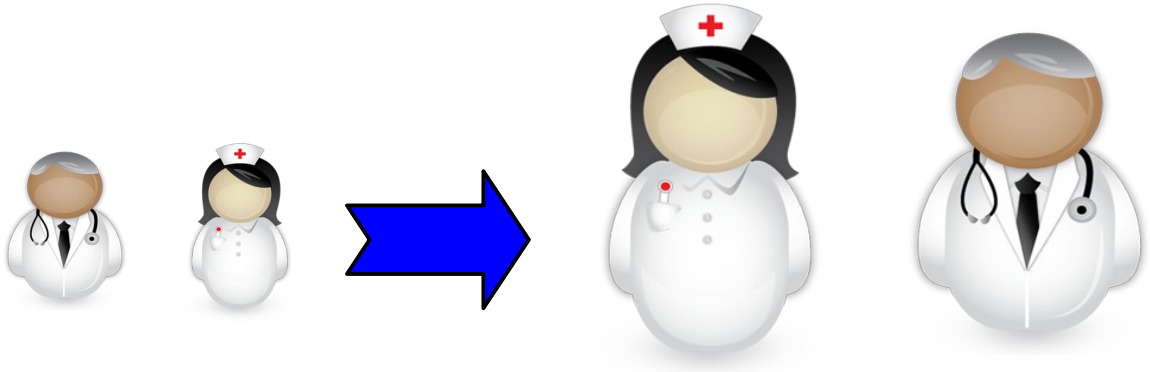


Beipiel: Querschneidende Erweiterung von Objekten mit Multi-Part-Bridge (Mehrfach-Mixin-Anreicherung)



Querscheidende Verfeinerung benötigt Mixin-Anreicherung (Object Fattening)

- **Mixin-Anreicherung (Object fattening, Objektverfettung)** ist ein Verfeinerungsprozess zur *Entwurfszeit*, der an ein *Kernobjekt aus dem Domänenmodell* Mixins 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", "Ärzte") aus dem Domänenmodell werden Stück für Stück mit Implementierungsinformation in Unterobjekten verfettet
  - Die Objekte aus dem Domänenmodell bleiben erhalten, werden aber immer dicker
- Mixin-Anreicherung geschieht mit dem Entwurfsmuster **Bridge**. Jede Anreicherung erzeugt eine Brücke, insgesamt ergibt sich eine **Multi-Bridge**



Arten von Verfeinerung durch Integration von Unterobjekten (Object fattening)

Bei der Mixin-Anreicherung können verschiedene Arten von Unterobjekten zur Verfeinerung benutzt werden:

- **Teileverfeinerung (Teileanlagerung):**
  - Finden von privaten Teilen von komplexen Analyseobjekten, die integriert werden
  - Schichtung ihrer Lebenszyklen
- **Rollenverfeinerung (Rollenanlagerung)**:
  - Finden von **Konnektoren (teams, collaborations)** zwischen Anwendungsobjekten
- **Fassadenanlagerung (port mixins)** (35.5.)
- Optional (Anhang):
  - **Facettenverfeinerung (Facettenanlagerung)**: Finden von Facetten-Unterobjekten, die Mehrfachklassifikationen ausdrücken
  - **Phasenverfeinerung (Phasenanlagerung):** Finden von Phasen-Unterobjekten, die Lebensphasen des komplexen Objektes ausdrücken

Unterobjekte (Mixins, Satelliten) gehören immer zu einer dieser speziellen Kategorien.

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

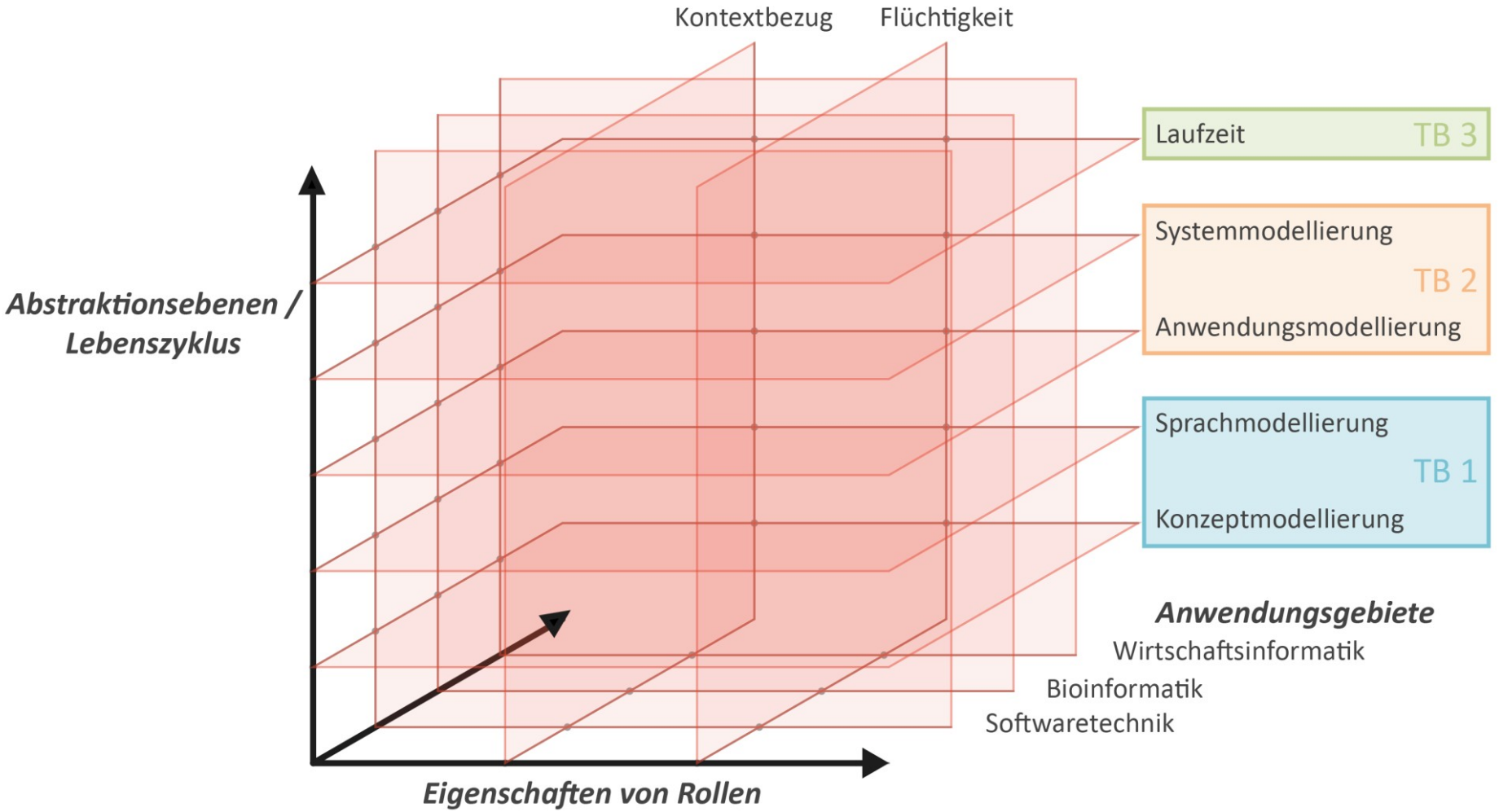
In Informationssystemen modellieren Rollen die wechselnden Kontexteigenschaften von Materialien.

Bei der Entwicklung von Informationssystemen werden ihre Tools und ihre Materialien durch Mehrfach-Brücken querschneidend verfeinert, die Mixins (z.B. Rollen und Teile) anlagern.

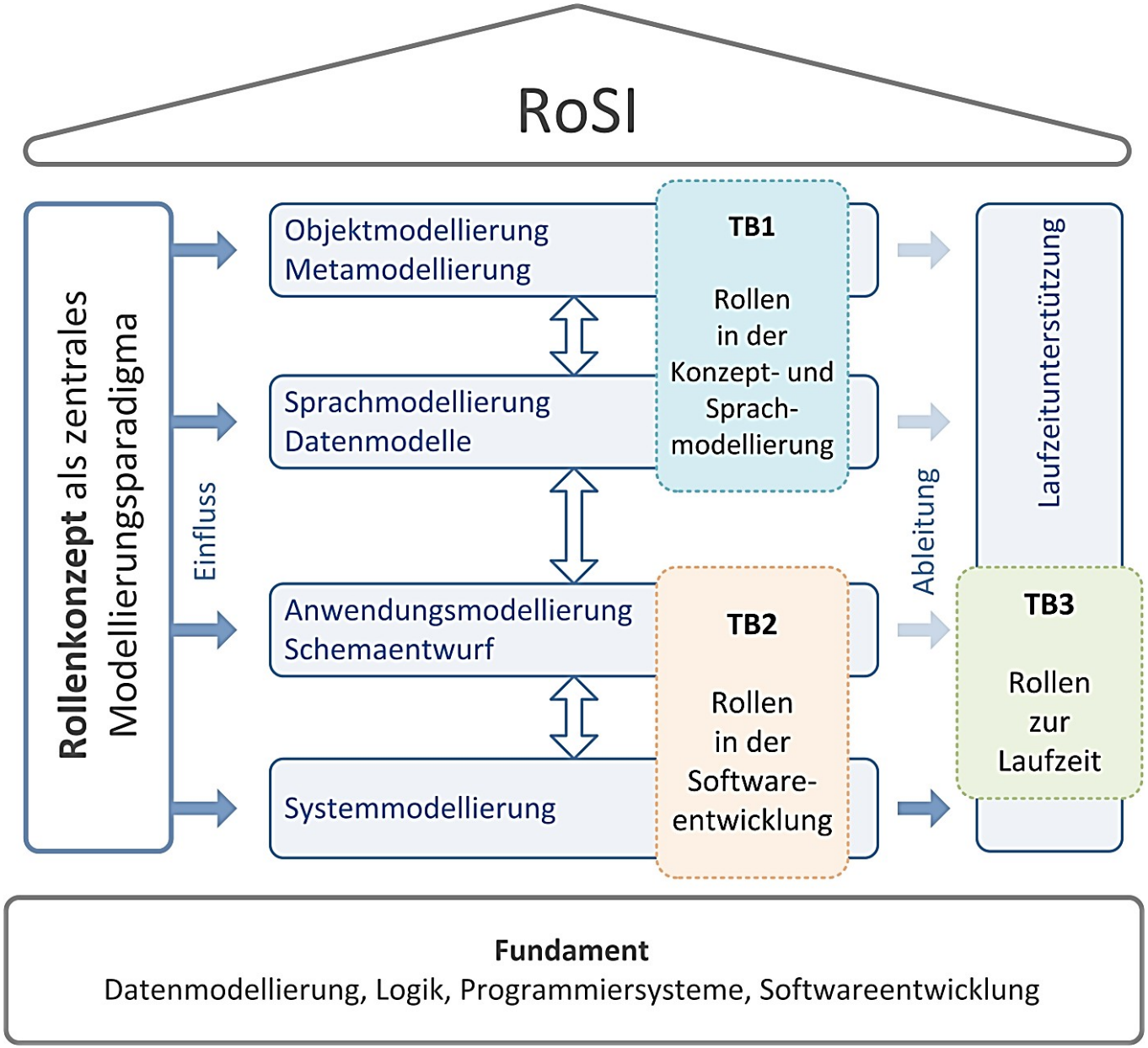
Aktuelle Forschung: Das Graduiertenkolleg “Rollenorientierte Software-Infrastrukturen (RoSI)

- https://rosi-project.org/
- Eine Speerspitze der Forschung in der Fakultät Informatik an der TU Dresden

Hypothese: Rollen sind ein Kernkonzept der Software-Entwicklung - Durchgängigkeit



Themenbereiche



Rollen, Teile, Facetten, Phasen als Forschungsthema

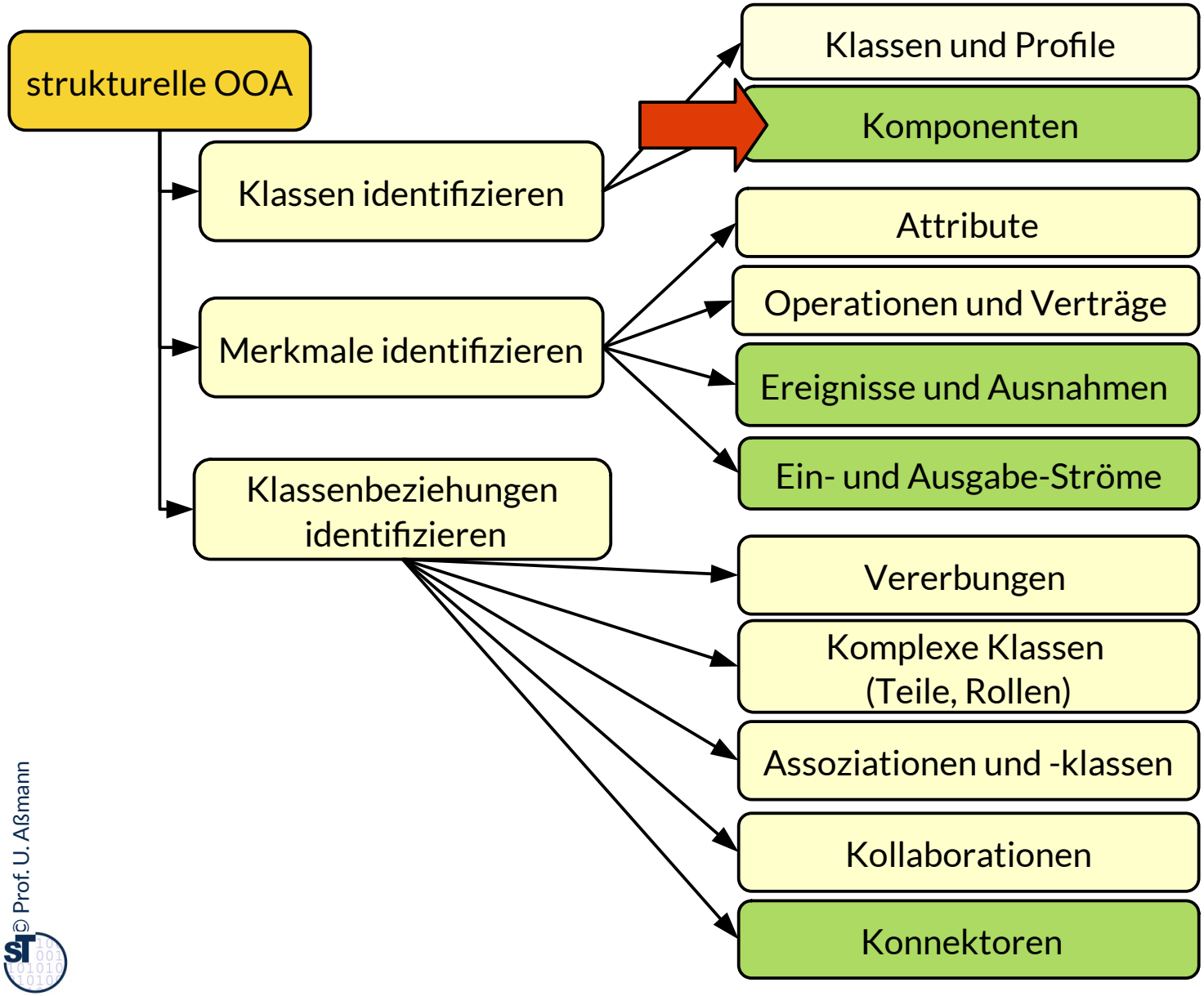
	Nicht-Fundiert	Fundiert
Rigide	(* natürlich *) Kern, <<natural>> Facette,<<facet>> privates Teil <<part>>	
Nicht rigide	Phase <<phase>>	Rolle <<role>>

32.5 Komponentenklassen als Verbundobjekte mit Fassaden-Mixins (Hierarchische komplexe Klassen)

- In der objektorientierten Systemanalyse wird das System als Verbundobjekt (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
- UML-Komponenten erleichtern die Wiederverwendung, weil sie einfacher austauschbar sind als Objekte

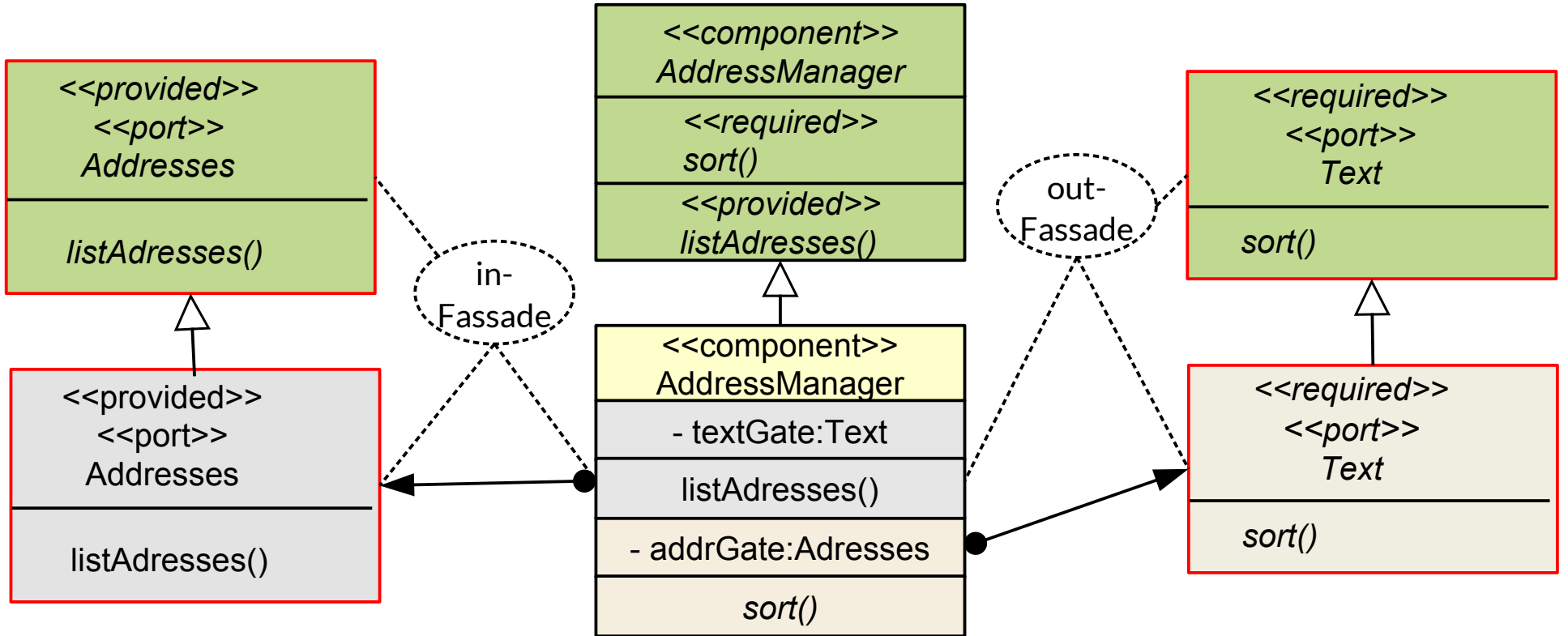
Q6: Schritte der strukturellen, metamodellgetriebenen Analyse

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



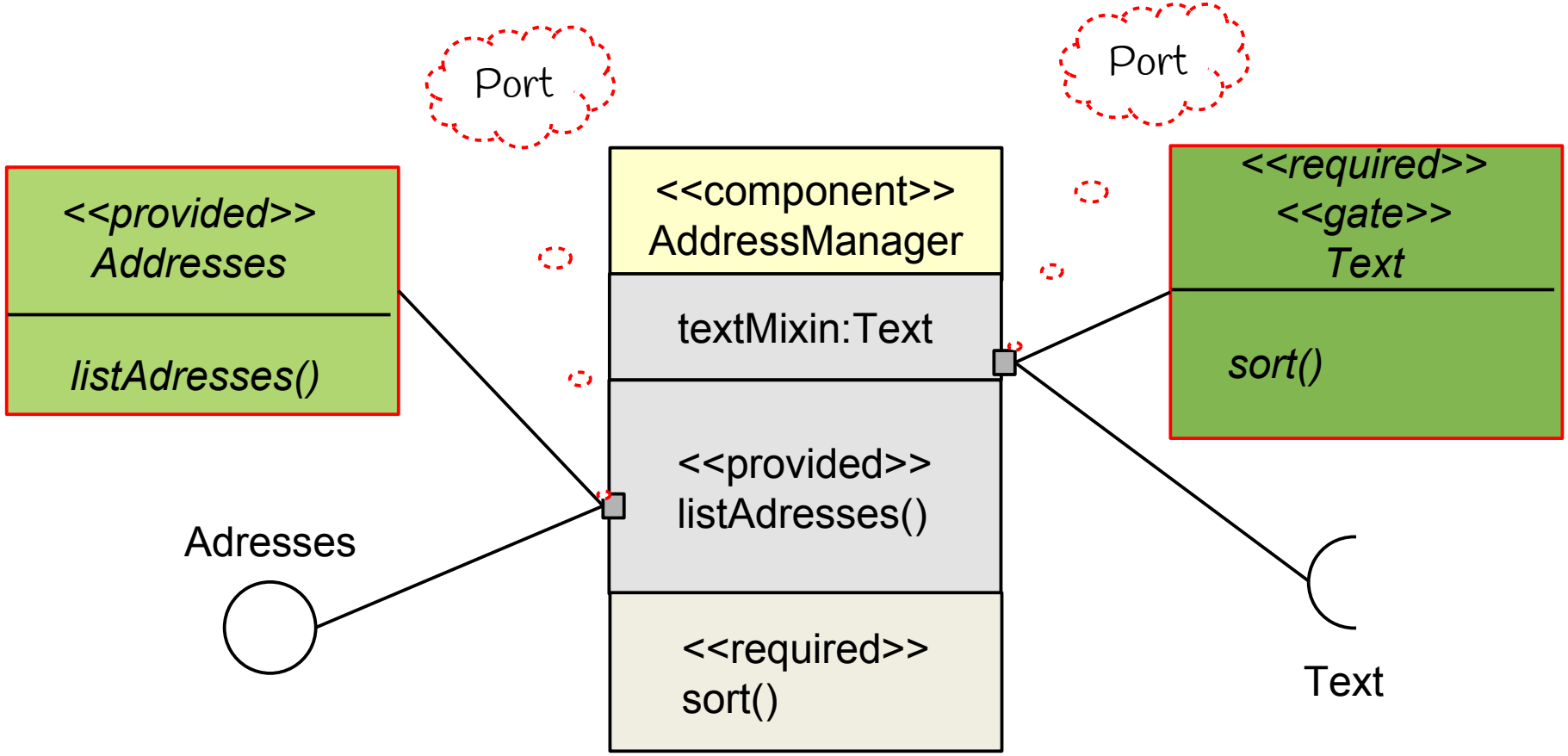
Entwurfsmuster “Component with Provided and Required Interfaces”

- Ein Verbundobjekt kann mehrere Mixins zur Kommunikation besitzen (***ports, gates, Tore, Anschlüsse***).
- Ein ***angebotenes Tor (provided port)*** ist ein Mixin, dessen öffentliche Schnittstelle (***provided interface***) einen Teil der Schnittstelle des Verbundobjektes bildet (Entwurfsmuster Fassade).
  - Das angebotene Tor bildet eine (in-)Fassade des Verbundobjektes
- Ein ***benötigtes Tor (required port)***, ist ein Mixin, das eine Teilmenge der von dem Objekt benötigten Merkmale (Dienste) sammelt. In der Regel ist seine Implementierung unspezifiziert. (out-Fassade)
  - Seine Schnittstelle heißt ***benötigte Schnittstelle (required interface).***
- Die Tor-Mixins dient der Isolation, weil die Aufrufe nach außen über das Tor gehen (Fassaden)



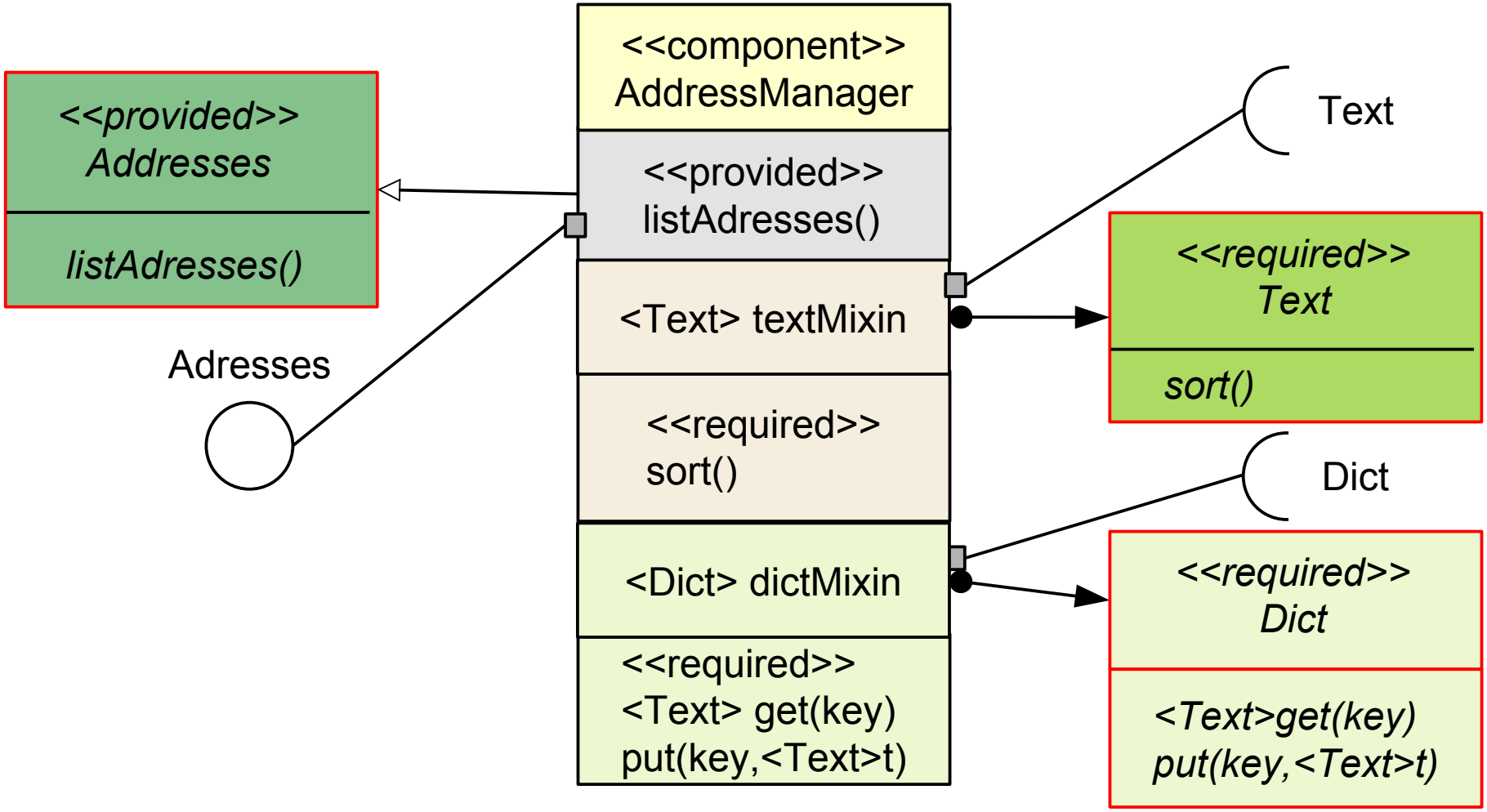
Entwurfsmuster “Component with Provided and Required Interfaces”

- Ein Verbundobjekt heißt ***Komponentenobjekt***, wenn es
  - alle angebotenen Dienste durch Port-Mixins (*provided ports*) anbietet, d.h. wenn es ganz von Fassaden-Mixins gekapselt ist
  - alle benötigten Dienste über Port-Mixins *(required ports)* aufruft
- Ein Komponentenobjekt ist gut isoliert, weil alle Aufrufe über Tor-Mixins (ports) gehen
- In einer ***Komponentenklasse*** sind alle Objekte Komponentenobjekte



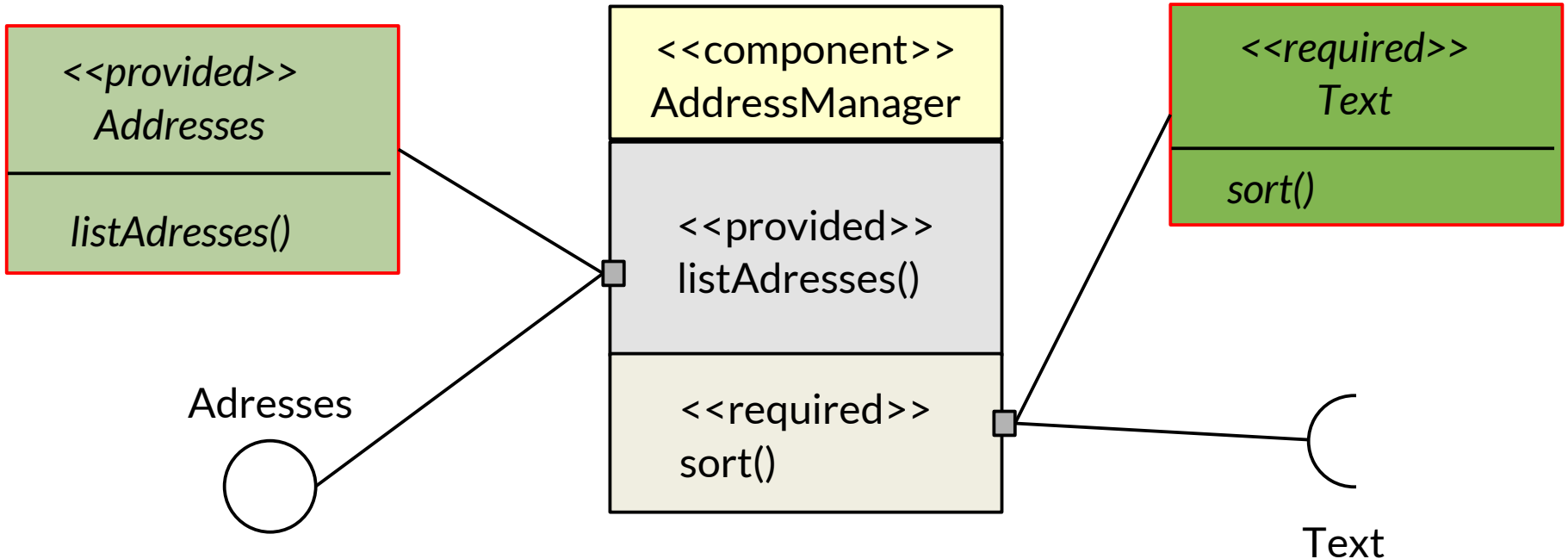
Entwurfsmuster “Component with Provided and Required Interfaces”

- Benötigte Tore (ports) sammeln Aufrufe aus der Komponente heraus nach außen
- Komponentenklassen können **mehrere** benötigte Tore (required gates) haben
- Wichtig: andere Aufrufe nach außen sind verboten!



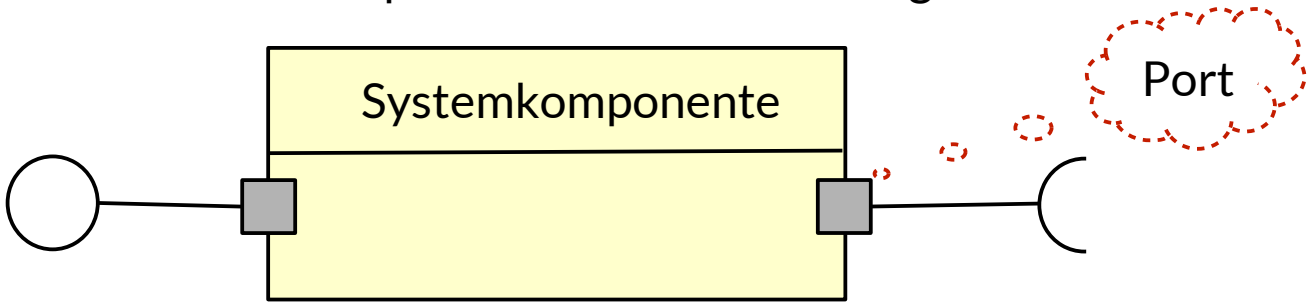
Komponenten mit Lollipops und Plugs (Balls and Sockets)

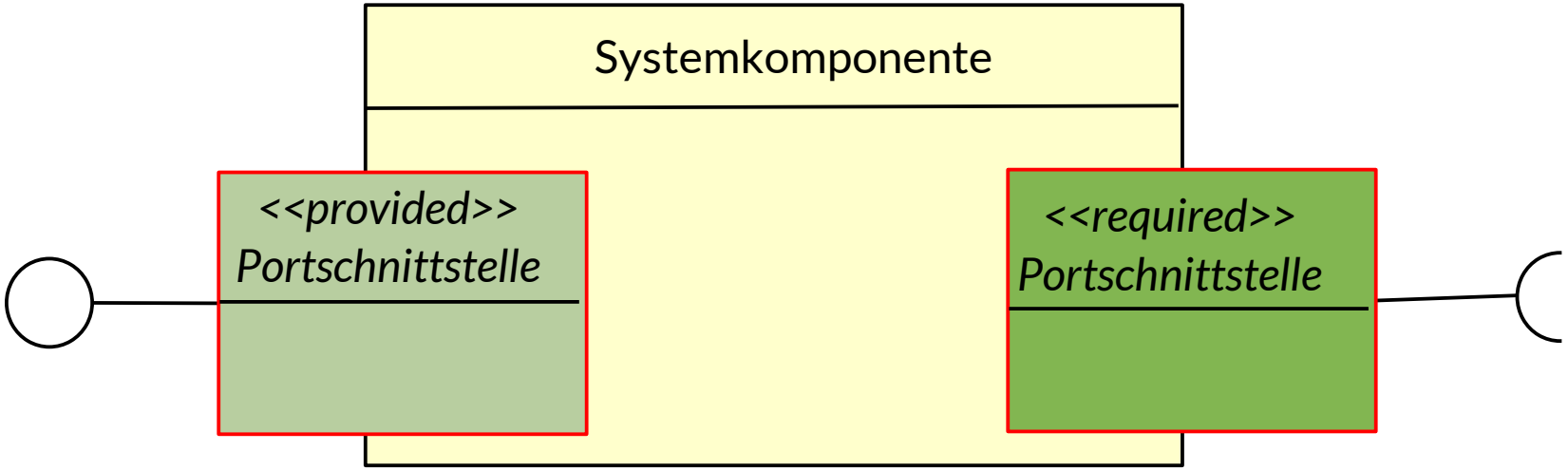
- Klassen, die *angebotene (provided)* von *benötigten (required)* Schnittstellen unterscheiden, heißen **Komponentenklassen**
  - Eine benötigte Schnittstelle spezifiziert, welche Partnerklasse die Klasse zum Ausführen benötigt (wird in separatem Compartment notiert)
    - Achtung: In Java wird *nicht spezifiziert*, sondern vom Übersetzer herausgefunden
  - I.d.R. sind UML-Komponenten Verbundobjekte 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 (Tore, Ports) gruppieren Schnittstellen

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

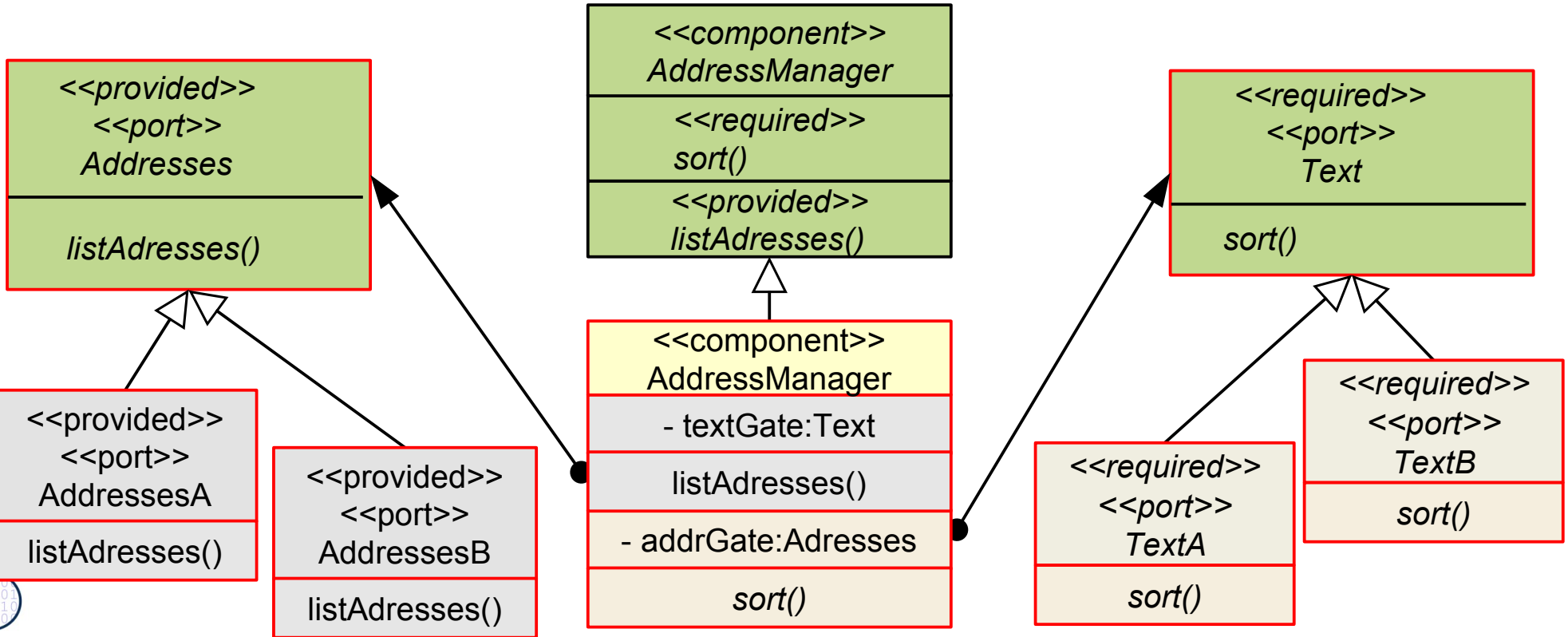




Geheimnisprinzip von Komponenten

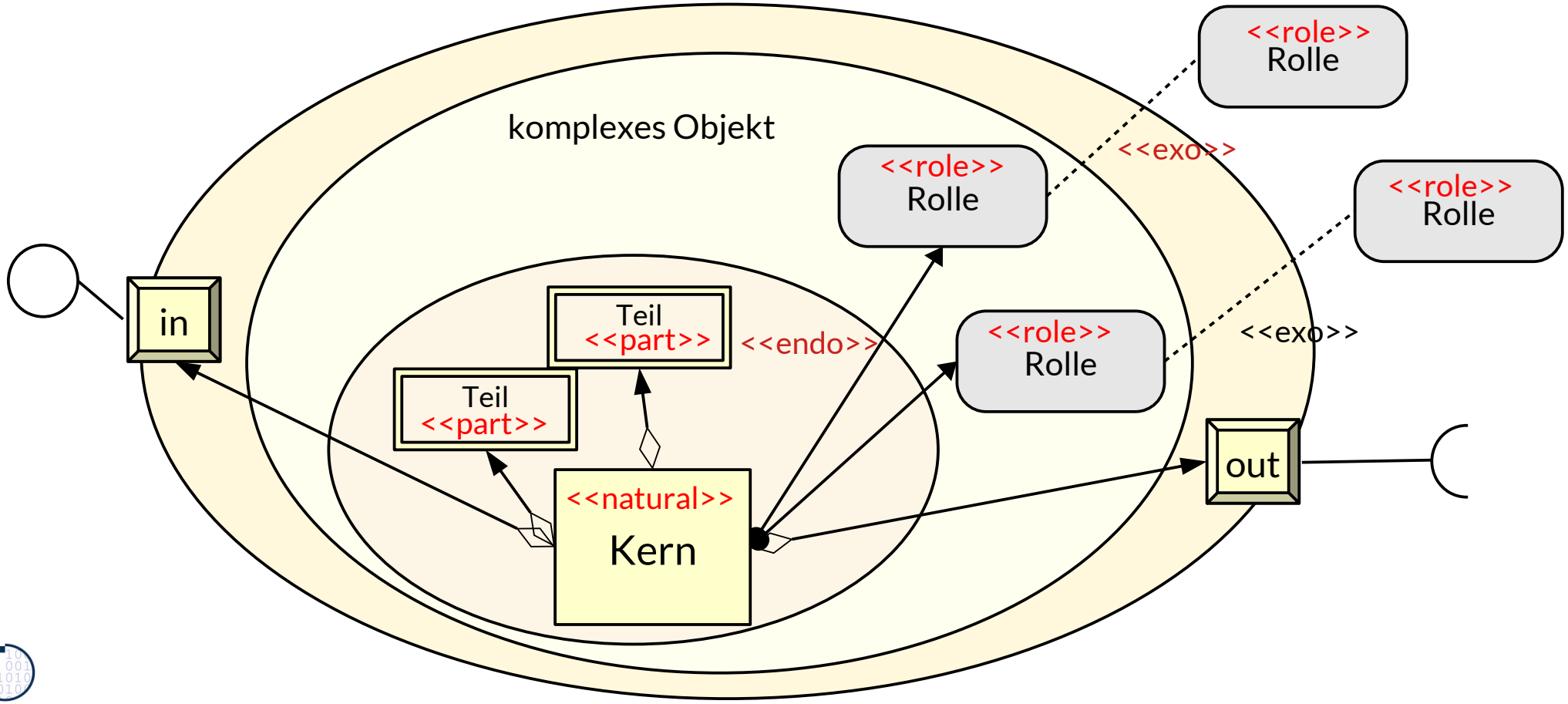
- Zum Austausch einer Komponente können die Portobjekte durch Polymorphie variiert werden (Mehrfachbrücken).
- Während Verbundobjekte verbergen, welches Mixin einen Dienst erbringt, verbergen Komponentenobjekte, welches externe Objekt einen Dienst erbringt

Komponenten erhöhen die Austauschbarkeit und Variabilität, weil sie ausschließlichüber ihre Ports kommunizieren



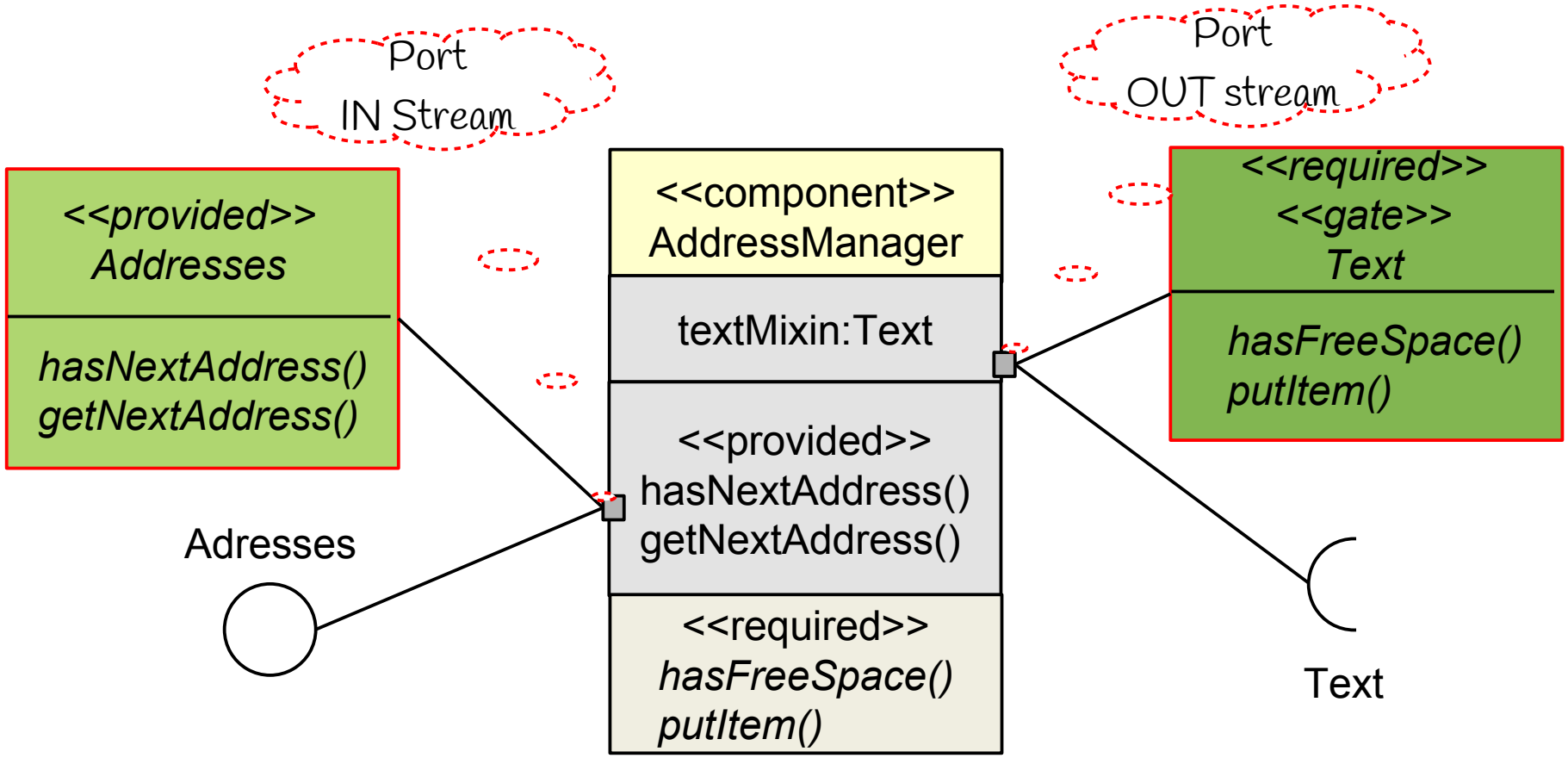
Verbundobjekte als Komponenten mit Teilen, Rollen und Ports

- Port-Mixins sind Rollen-Mixins, die die Kommunikation bestimmen
- Verbundobjekte mit Ports heißen *Komponenten*



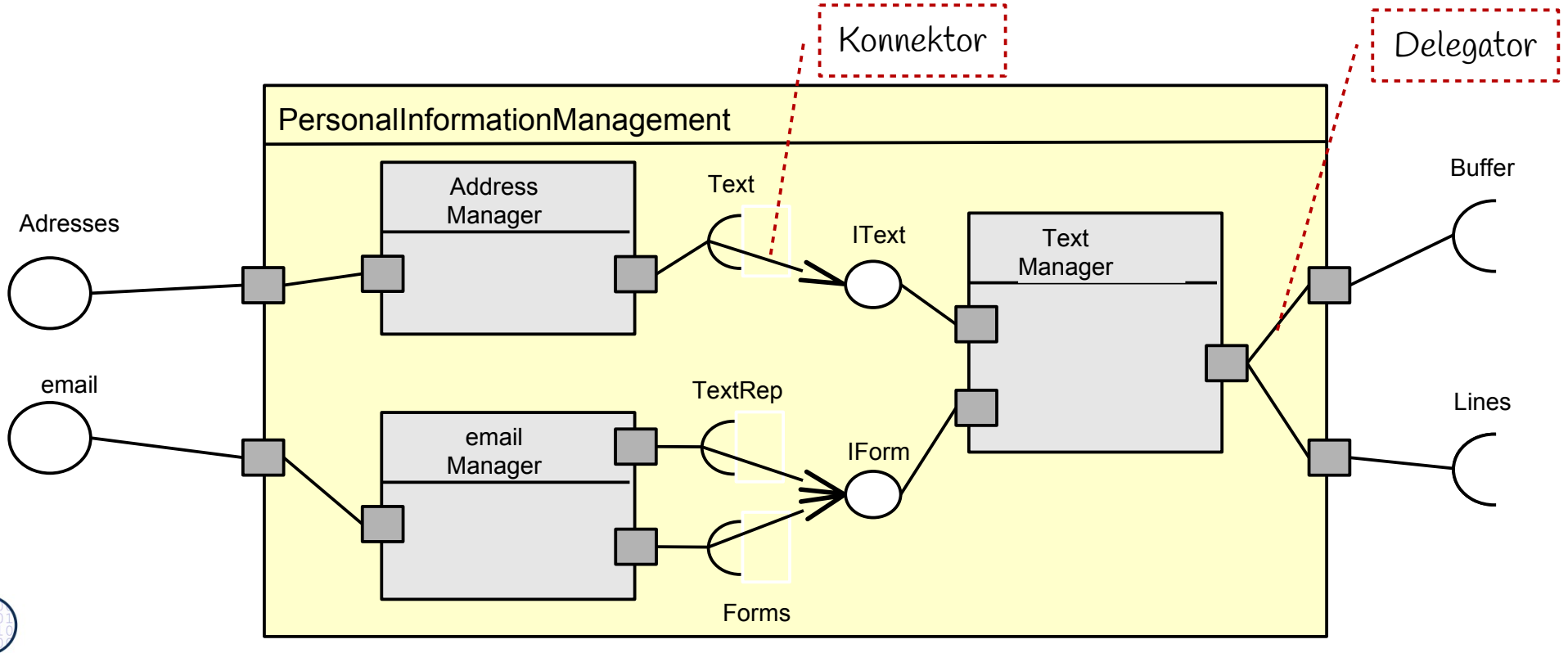
Entwurfsmuster “Component with Provided and Required Streams”

- Ein Komponentenobjekt hat
  - einen Strom-IN-Port, wenn dieser IN-Port einen Input-Stream (Iterator) enthält
  - einen Strom-OUT-Port, wenn dieser OUT-Port einen Output-Stream (Senke) enthält
- Diese Ports können an Kanäle (und Konnektoren) angebunden werden.



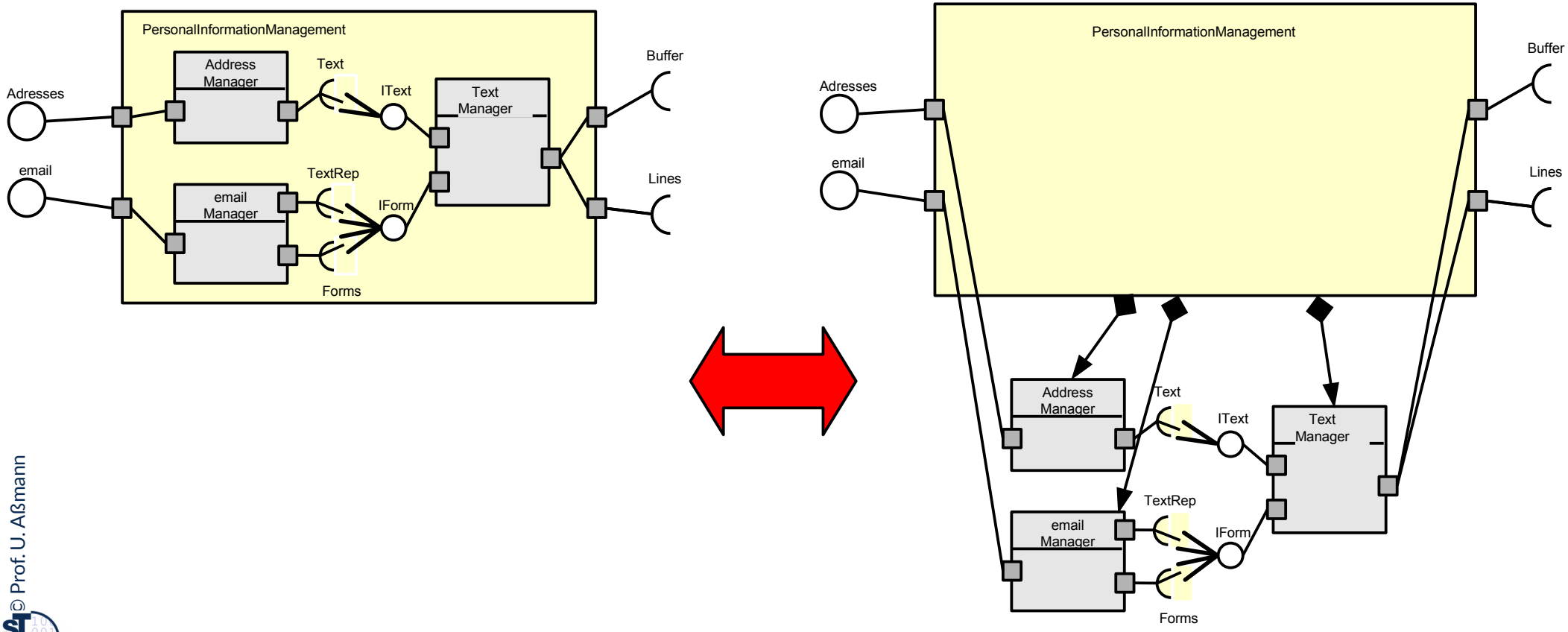
Schachtelung von Klassen-Komponenten

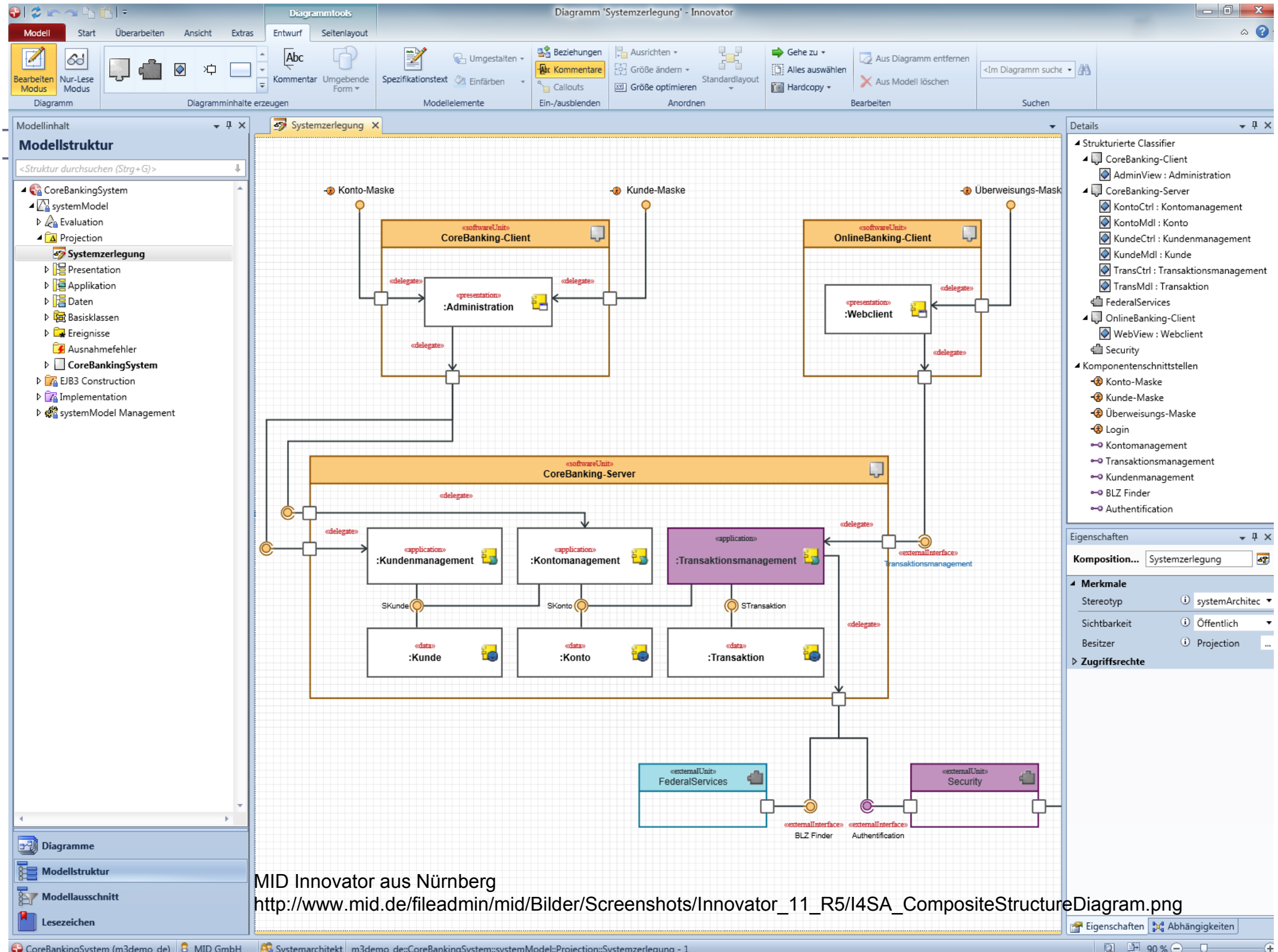
- Eine **Klassen-Komponente** ist eine Klasse mit angebotenen und benötigten Schnittstellen, mit Ports (Portklassen) und inneren Klassen
  - **Klassen**-Komponenten bilden hierarchische Klassen für hierarchische Objekte
  - Ports werden mit *Verbindern (Links, einfachen Konnektoren)* verbunden
  - *Delegator*: 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 Sichbarkeit begrenzen
  - Eine Komponente ist gleichzeitig ein *Paket* und eine *Facade* für alle Unterkomponenten
  - I.d.R. wird eine UML-Komponente mit einem Facade oder mehreren Bridge Pattern realisiert



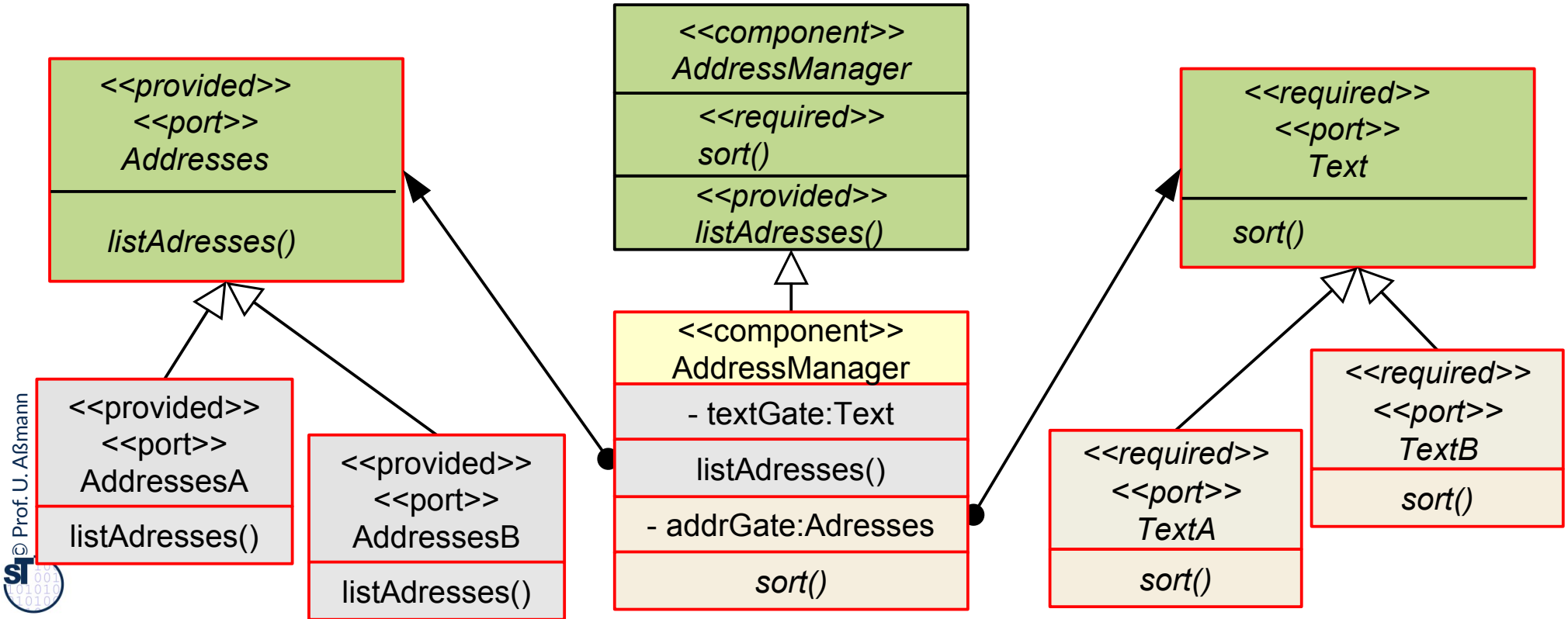


MID Innovator aus Nürnberg
http://www.mid.de/fileadmin/mid/Bilder/Screenshots/Innovator_11_R5/I4SA_CompositeStructureDiagram.png

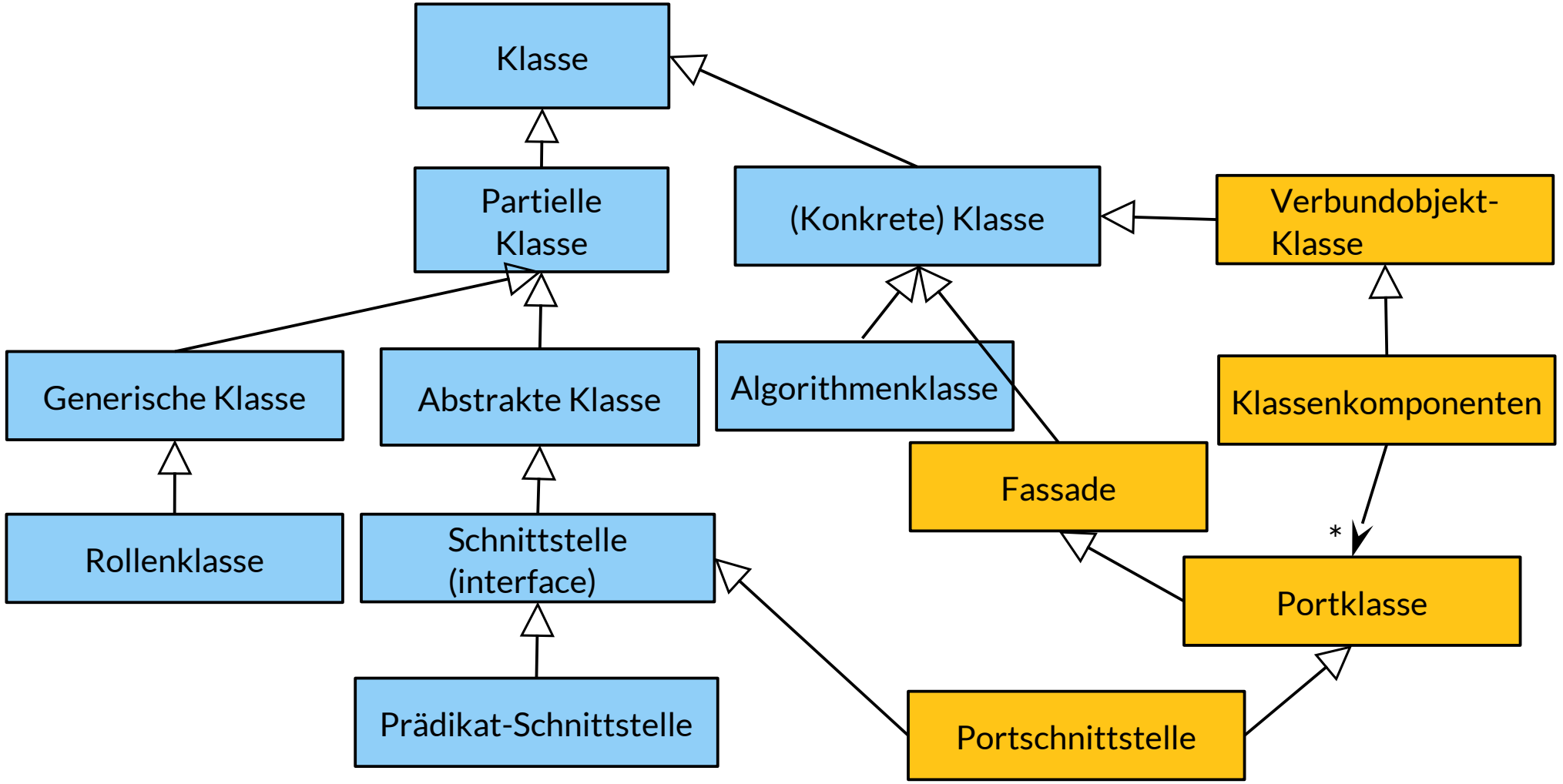
Port-Anreicherung

- Klassen können durch Anreicherung mit Port-Mixins zu Komponenten punktweise verfeinert werden
- Portklassen können variiert werden ***(Port-Polymorphismus)***

Port-Anreicherung ist eine wichtige Verfeinerungsoperation zur Erhöhung der Wiederverwendung von Klassen



Q2: Begriffshierarchie von Klassen (Erweiterung)



Warum ist das wichtig?

- Auf dem Weg vom Analysemodell zum Implementierungsmodell entstehen *komplexe Objekte (Verbundobjekte)*
- Verbundobjekte werden, durch mehrfache Anwendung des Entwurfsmusters Brücke, mit Mixins erweitert (punktweise Verfeinerung durch Mixin-Anreicherung, Objektverfettung)
- Komponenten sind Verbundobjekte mit Port-Mixins, die sehr gute Austauschbarkeit bieten
- Funktionale Variabilität:
  - Will man verschiedenen Kundengruppen verschiedene Funktionalitäten anbieten, verfeinert man spezifisch für die Kundengruppen
  - Und erhält mehrere *Produktvarianten*
  - Und insgesamt eine *Produktlinie*
- Die Modellierung von Geschäftsobjekten als komplexe Objekte ist eine Hauptaufgabe von Informationssystemen wie SAP
- Plattformportabilität:
  - Da Wiederverwendung ist das Hauptmittel der Softwarefirmen, um profitabel arbeiten zu können, erweitert man einen Entwurf zu *verschiedenen* Implementierungsmodellen, indem man verschiedene Brücken einzieht

Was haben wir gelernt?

- Es gibt komplexe Objekte, die so groß sind, dass sie aus Kern und Unterobjekten (Mixins, Satelliten) bestehen
  - Komplexe Objekte sind immer hierarchisch oder azyklisch und verwenden Endo-Assoziationen. Sie werden oft mit Multi-Bridge realisiert
  - Der Kern bildet meist eine Facade für die Unterobjekte
  - Unterobjekte sind typisch einer Kategorie zugeordnet (Rollen, Teile, Facetten, Phasen)
- Teile-Klassen sind unvollständige Klassen
- Rollenklassen sind dynamische, unvollständige Klassen
- Einfache Implementierung mit Bridge und Multi-Bridge
- Mixin-Anreicherung 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 Mixin-Anreicherung, um vom Problem des Kunden zum Analysemodell, dann über das Entwurfs- und Implementierungsmodell zum Programm zu kommen.

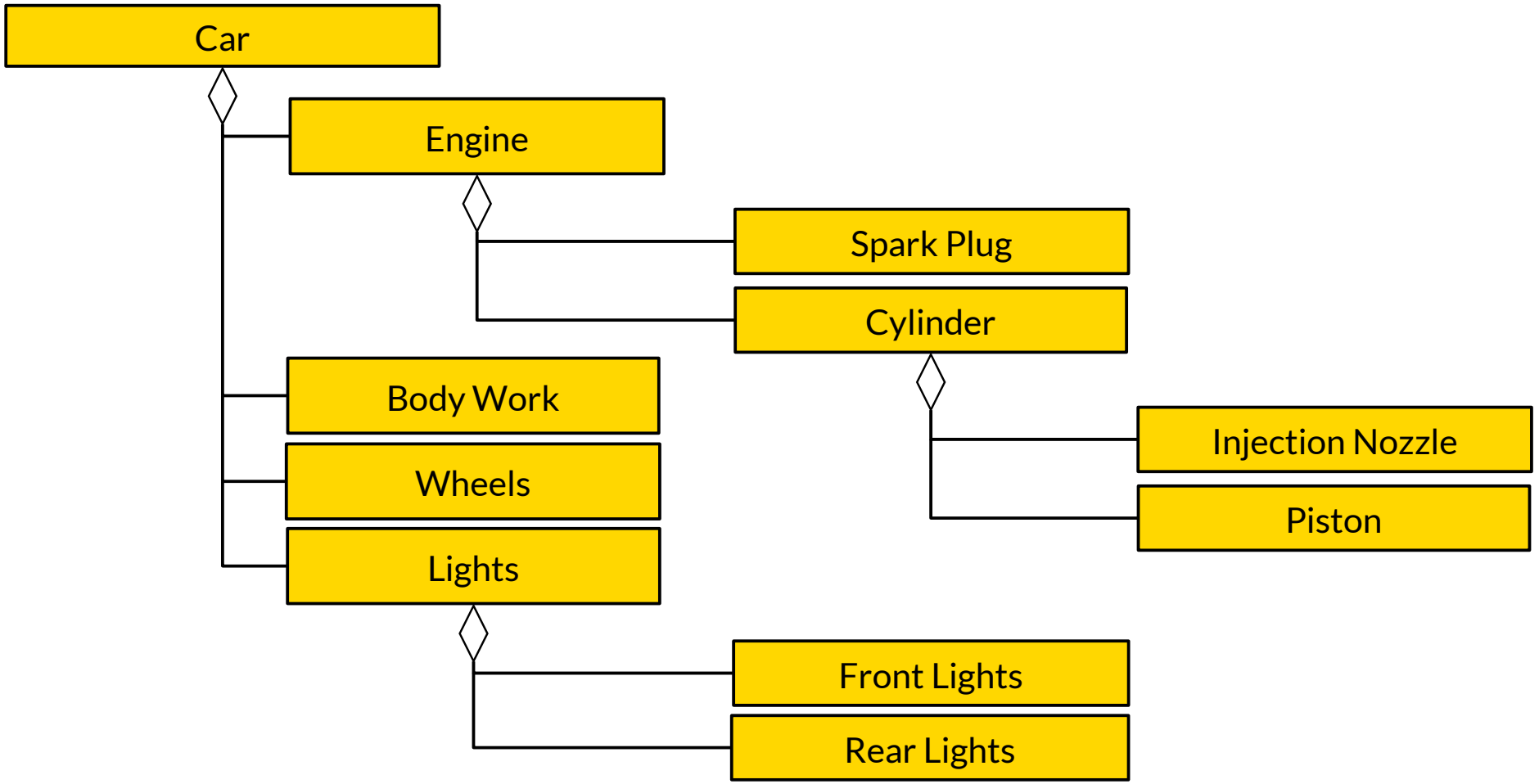
Anhang

- Erklären Sie den Unterschied zwischen einem Verbundobjekt, einem einfachen Objekt und einem Mixin.
- Worin besteht der Prozess der Mixin-Anreicherung? Warum ist Mixin-Anreicherung eine punktweise Verfeinerung?
- Wie implementiert man ein Verbundobjekt mit fester Anzahl von Unterobjekten?
- Was passiert zur Laufzeit in einer Multi-Bridge mit Rollen-Unterobjekten? Wie vermeidet man einen Absturz des Systems, wenn die Rolle nicht gespielt wird?
- Wenn Java Rollen und Teile durch Referenzen implementiert, gibt es dann auf Implementierungsebene überhaupt einen Unterschied? Warum macht es dennoch Sinn, in der Analyse Rollen und Teile zu unterscheiden?
- Warum sollte man Rollen von Objekten konzeptuell unterscheiden?
- Warum erhöht eine Komponentenklasse die Wiederverwendung?
- Erklären Sie den Prozess der Port-Anreicherung. Warum ist er eine Spezialisierung der Mixin-Anreicherung?

Ende des obligatorischen Materials

Darstellung komplexer Objekte mit privaten Teilen

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



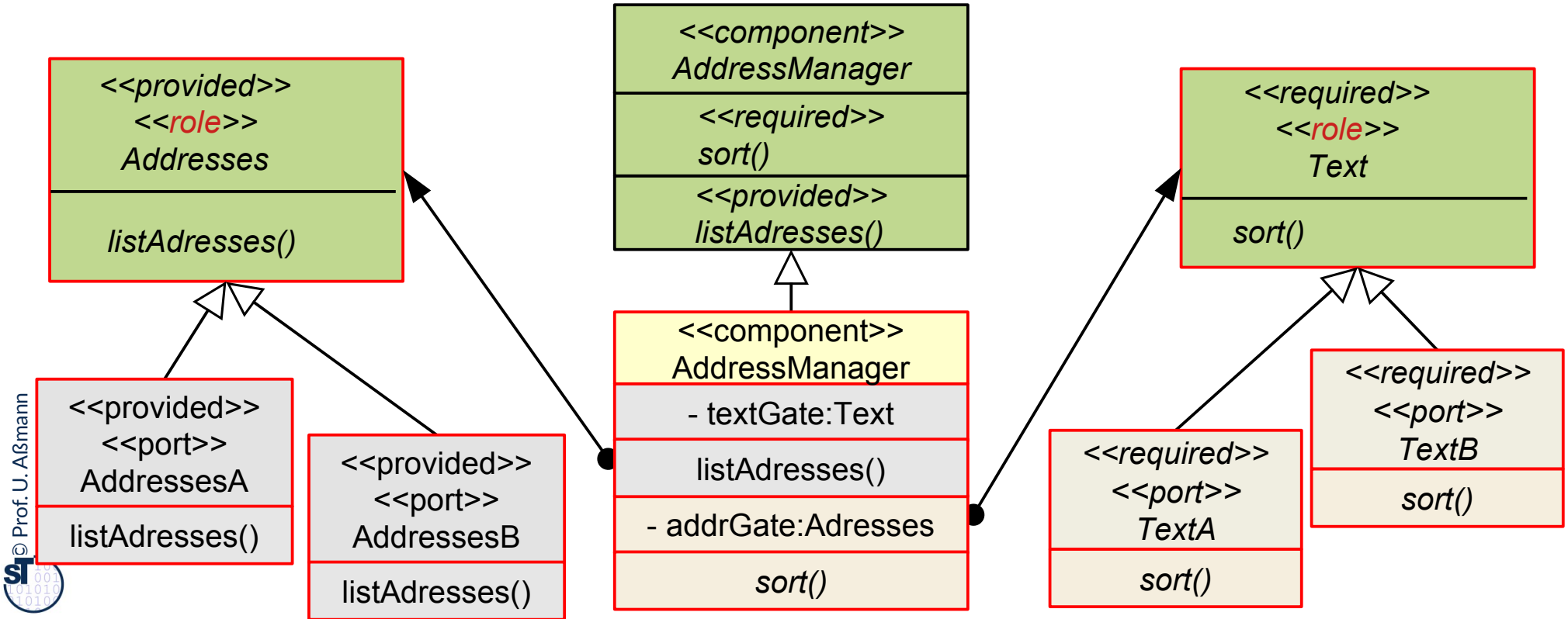
32.A. Experimentell

(optional, nicht klausurrelevant)

Komponenten werden zu Kollaborationen

- Komponente sind Kollaborationen, wenn ihre Ports Rollen sind, die die Identität mit dem superimponierten Objekt teilen.
- Die Frage im Interaktionsdiagramm sind:
  - Wie eingebettet sind die Objekte? (Rollen, Unterobjekte, separate Objekte)
  - Wie sehen die Aufrufschemata auf provided (Lieferant) und required (Zulieferer) Seite aus.
- Eine ***Zuliefererrolle (supplier)*** ist eine benötigte Rolle. Ein ***Angebot (offer)*** ist eine angebotene Rolle.

Komponenten sind Kollaborationen, wenn sie Rollenobjekte anbieten



Arten von Szenen

- Eine ***Server-Kollaboration*** ist eine Kollaboration, die nach außen einen oder mehrere Angebote liefert.
- Eine ***synchrone Kollaboration*** lässt sich nach außen wie eine synchrone Nachricht kapseln.
- Eine ***verzögert synchrone Kollaboration*** lässt sich nach außen wie eine verzögert synchrone Nachricht kapseln.
- Eine ***exogene Kollaboration*** ist eine Kollaboration, die alle benötigten Rollen startet, bzw. Ihnen eine Initialbotschaft sendet.
- Eine ***erweiterbarer Port*** ist eine Lieferenten oder Zulieferer-Schnittstelle, die erweiterbar ist.

Basic Composition Operations

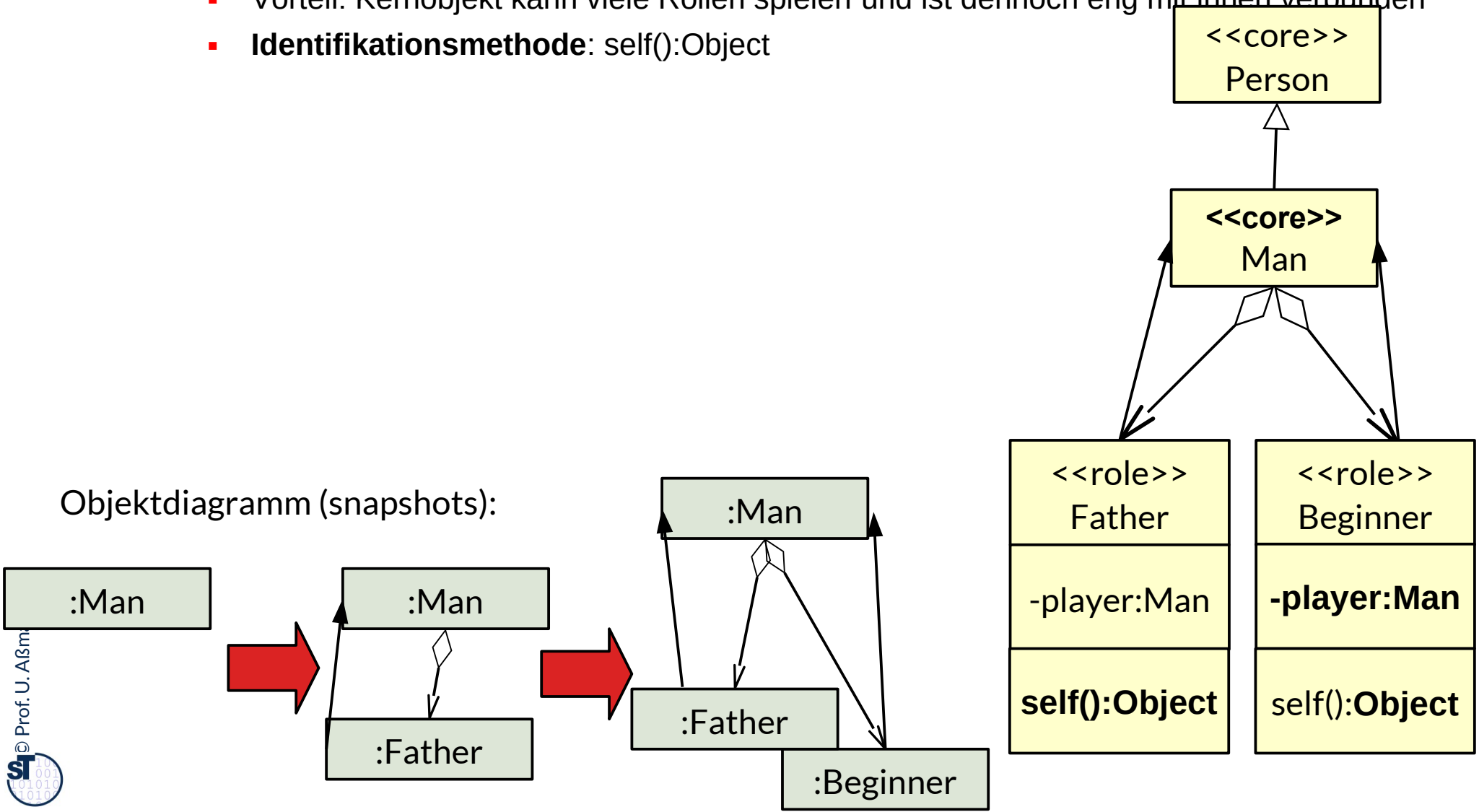
- Injection of a
  - Initial message (Initialbotschaft)
  - Terminal message
  - Call
  - Channel receive
  - Channel send
  - Timeline snippet (a role), together with its initial message and its terminal message
- Unification of a timeline snippet
-

32.A.1 Weitere Realisierungen von Rollen-Unterobjekten

(optional, nicht klausurrelevant)

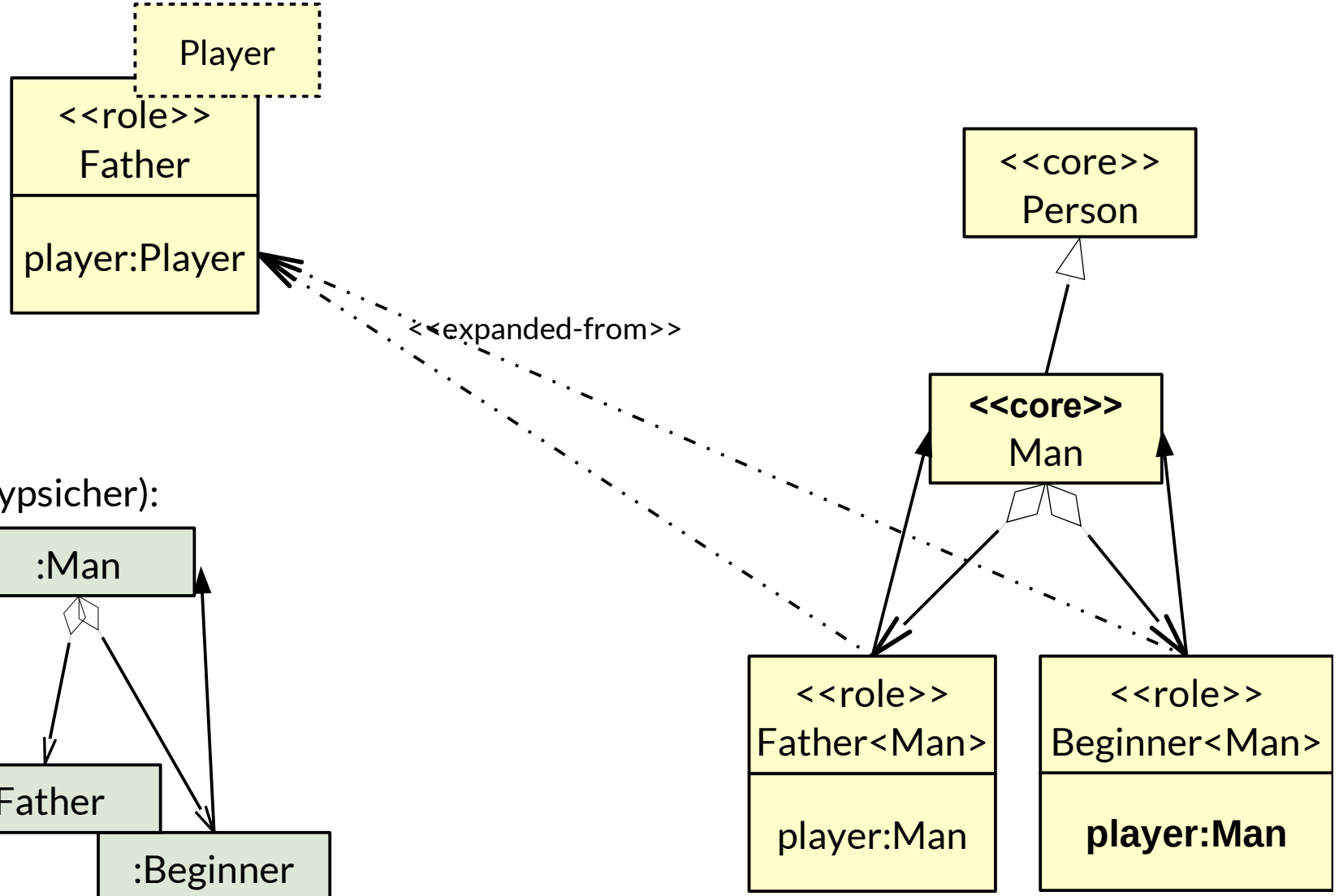
Implementierungsmuster: “Rollen mit Aggregation und Kern-Link”

- Rollen sollten zusätzlich durch einen Rückwärts-Link `player` mit dem Kernobjekt verbunden sein, da sie semantisch zu ihm gehören
  - Vorteil: Kernobjekt kann viele Rollen spielen und ist dennoch eng mit ihnen verbunden
  - **Identifikationsmethode**: self():Object

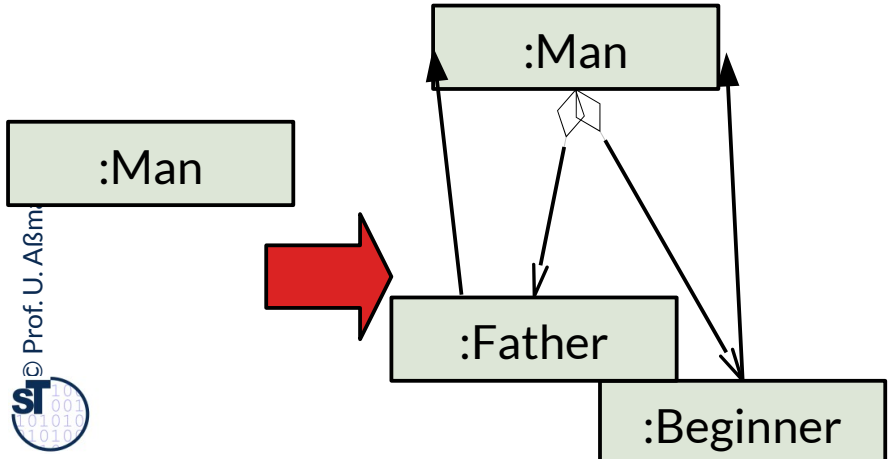


Implementierungsmuster: “Rollen mit generischem Player-Link”

- Rollen können als generische Klassen gesehen werden, die als generischen Parameter den Player erwarten
  - Dann ist das Netz zwischen Kernobjekt und Rollen typsicher



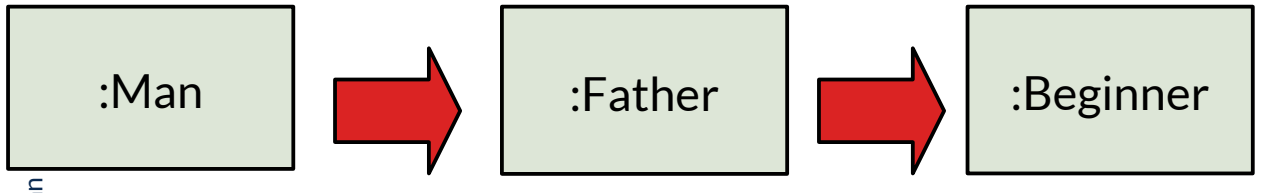
Objektdiagramm (typsicher):

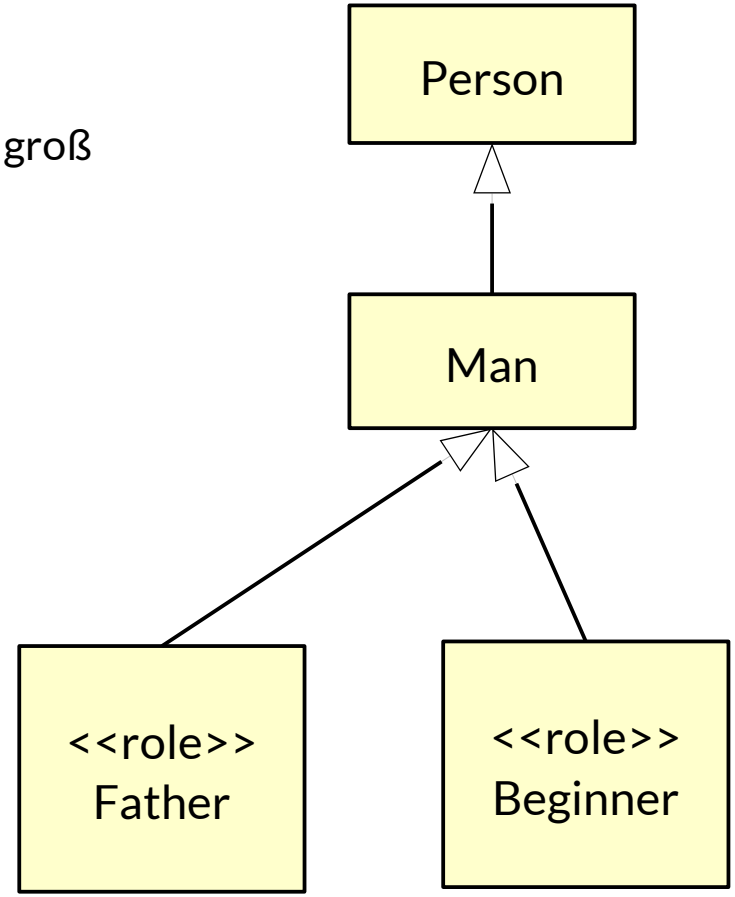


Oft genutztes, aber ungeschicktes Implementierungs-muster: “Rollen als Unterklassen”

- Man kann mit Unterklassen die Rollen einer Oberklasse realisieren (“Implementierungsmuster”)
- Man implementiert den Wechsel einer Rolle durch den Wechsel der entsprechenden Unterklasse durch
  - Alloziere und fülle neues Objekt aus den Werten des alten Objektes heraus
  - Setze Variable um auf neues Objekt
  - (Dealloziere altes Objekt)
- Problem: bei vielen Kontexten werden die Objekte viel zu groß

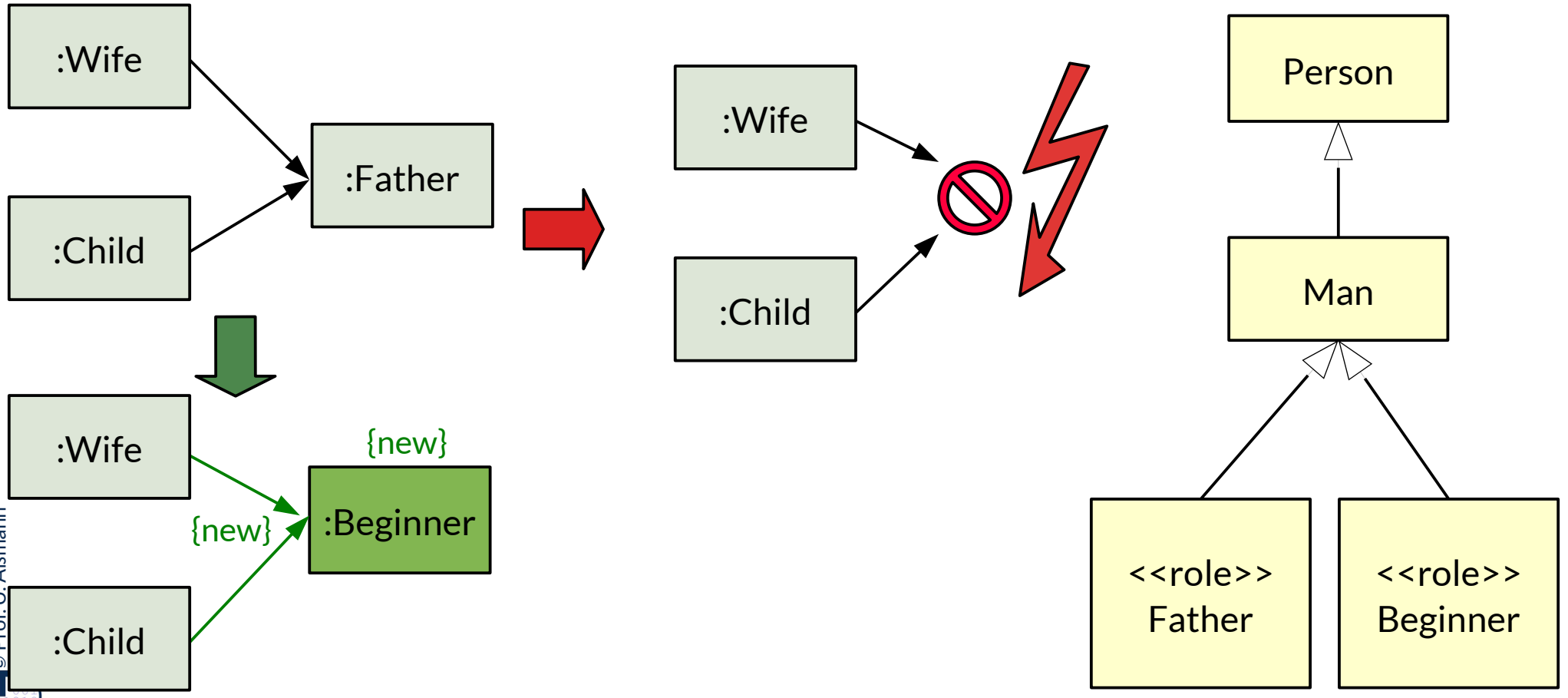
Objektdiagramm (snapshots):





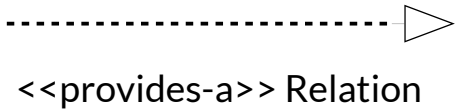
“Nirvana” im Implementierungsmuster: “Rollen als Unterklassen”

- Problem: Referenzen auf Objekte müssen bei Rollenwechsel umgesetzt werden und hinterlassen “Nirvana-Kanten” (dangling edges)
- Einer der häufigsten Programmierfehler überhaupt

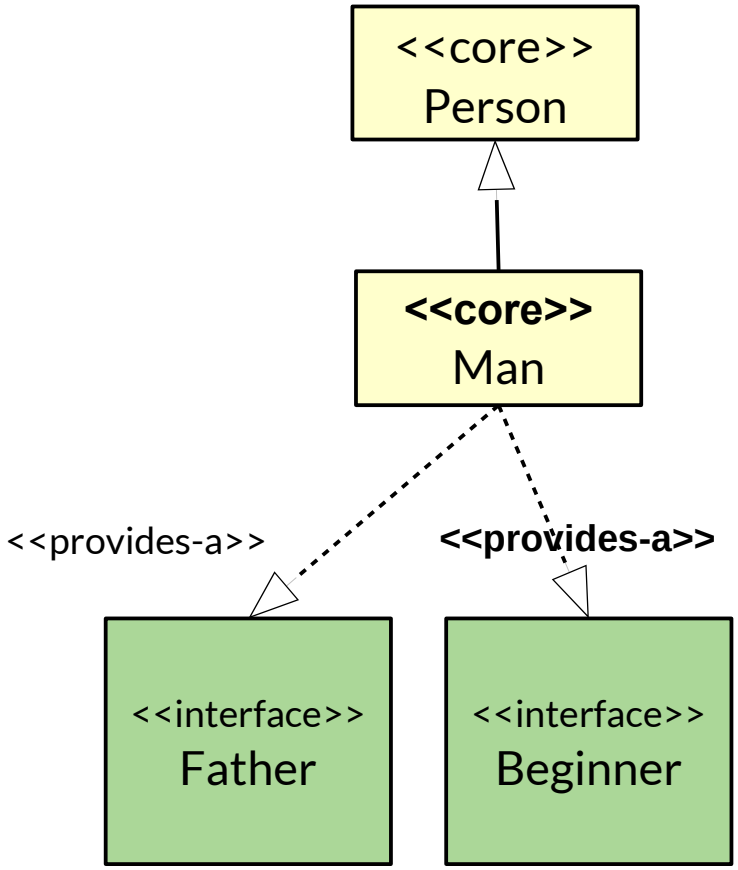


Implementierungsmuster: “Rollen mit Mehrfachvererbung”

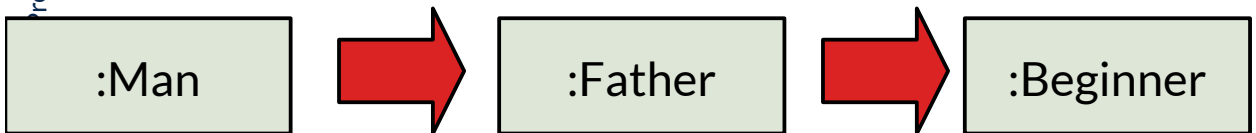
- Rollen können auch durch *Mehrfachvererbung* realisiert werden
  - .. ähnlich zur Realisierung mit Einfachvererbung
- Vorteil: Kernobjekt erbt Rollenverhalten
- Nachteil: nur in Scala und C++ möglich



<<provides-a>> Relation



Objektdiagramm (snapshots):



32.A.2 Facettenklassifikation und Facetten-Unterobjekte

(optional, nicht klausurrelevant)

Facettenklassifikation

- 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

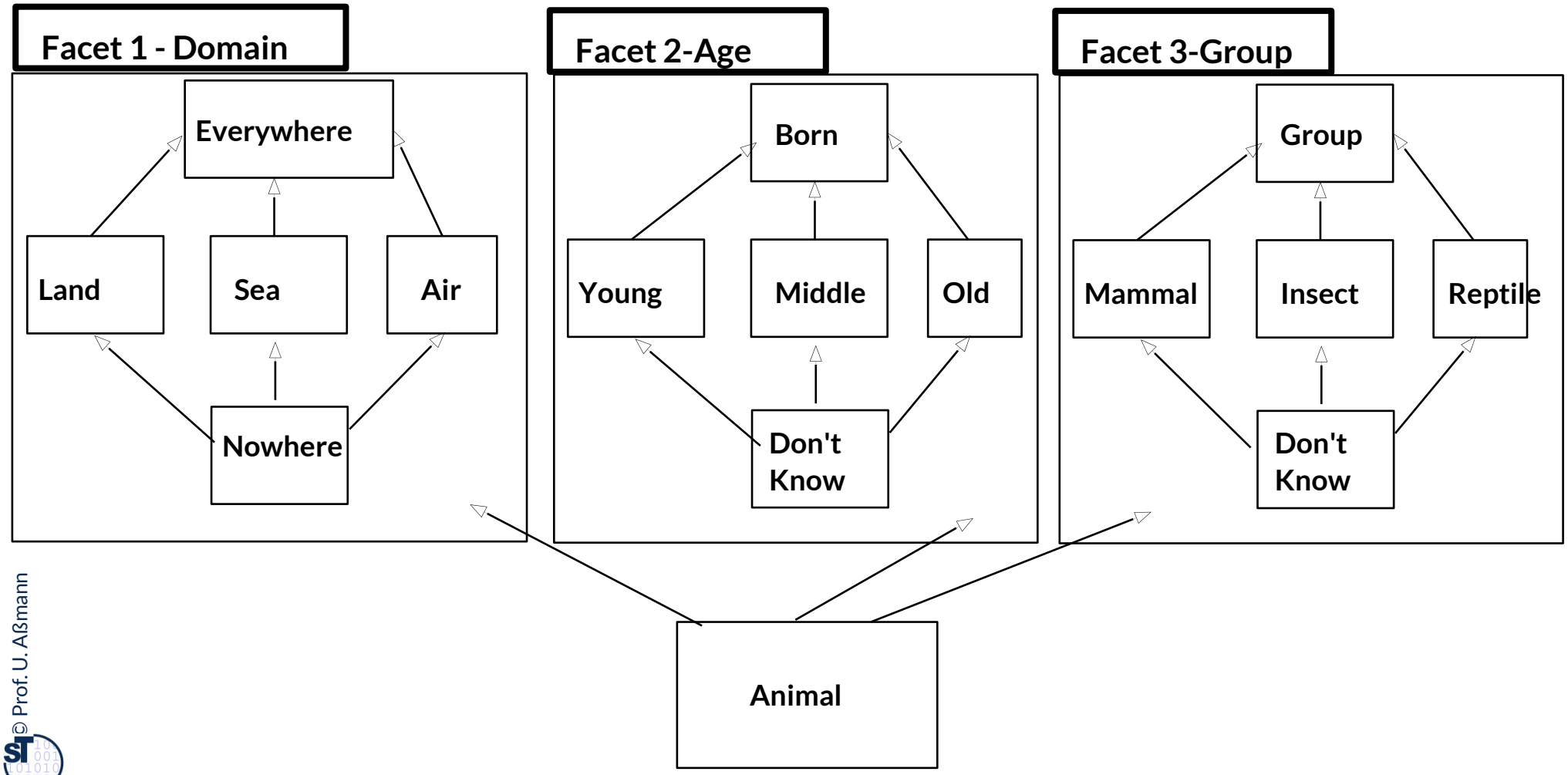
http://www.webdesignpractices.com/navigation/facets.html

Facetten von Lebewesen

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

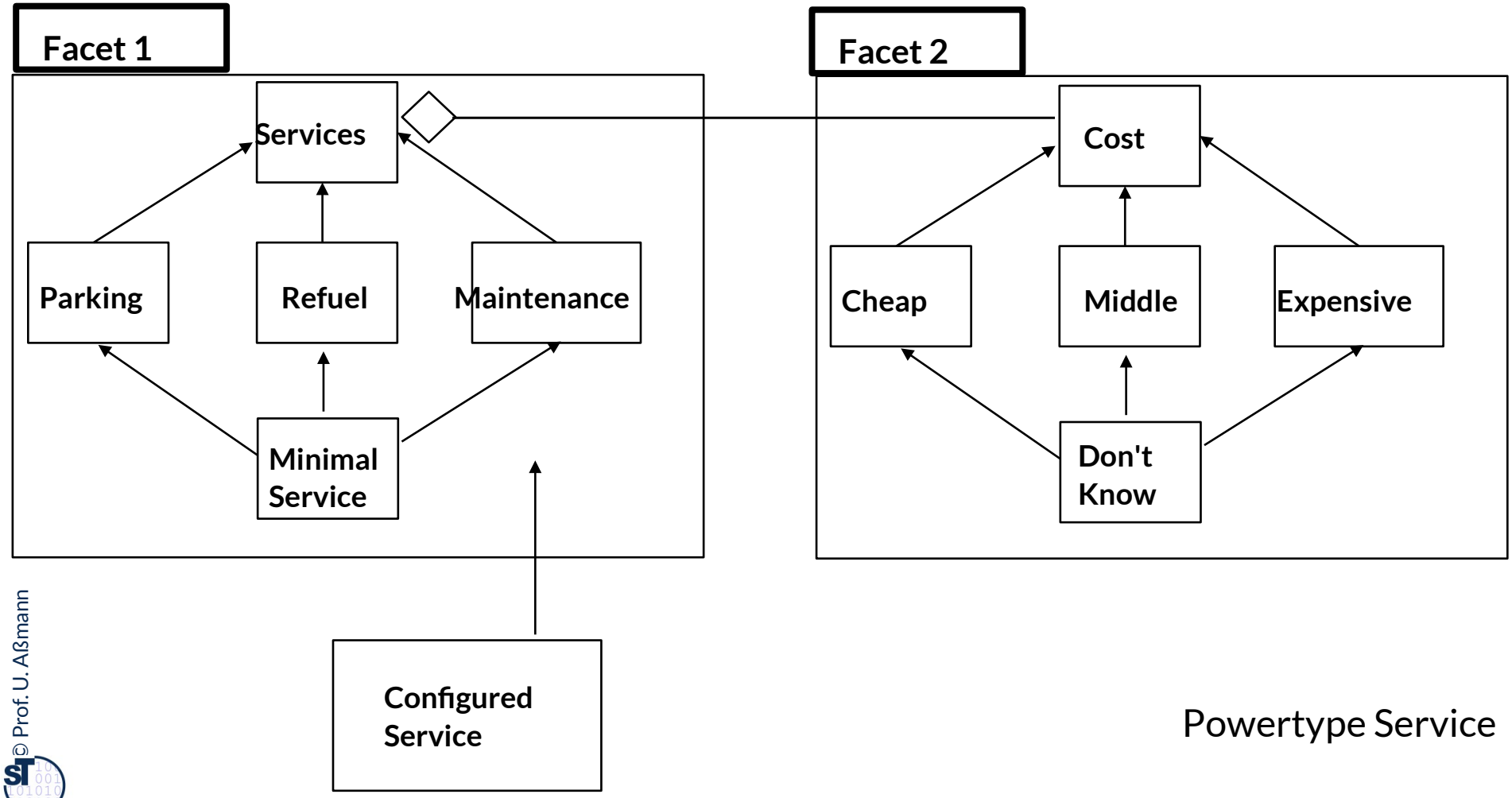
Facetten faktorisieren aus

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



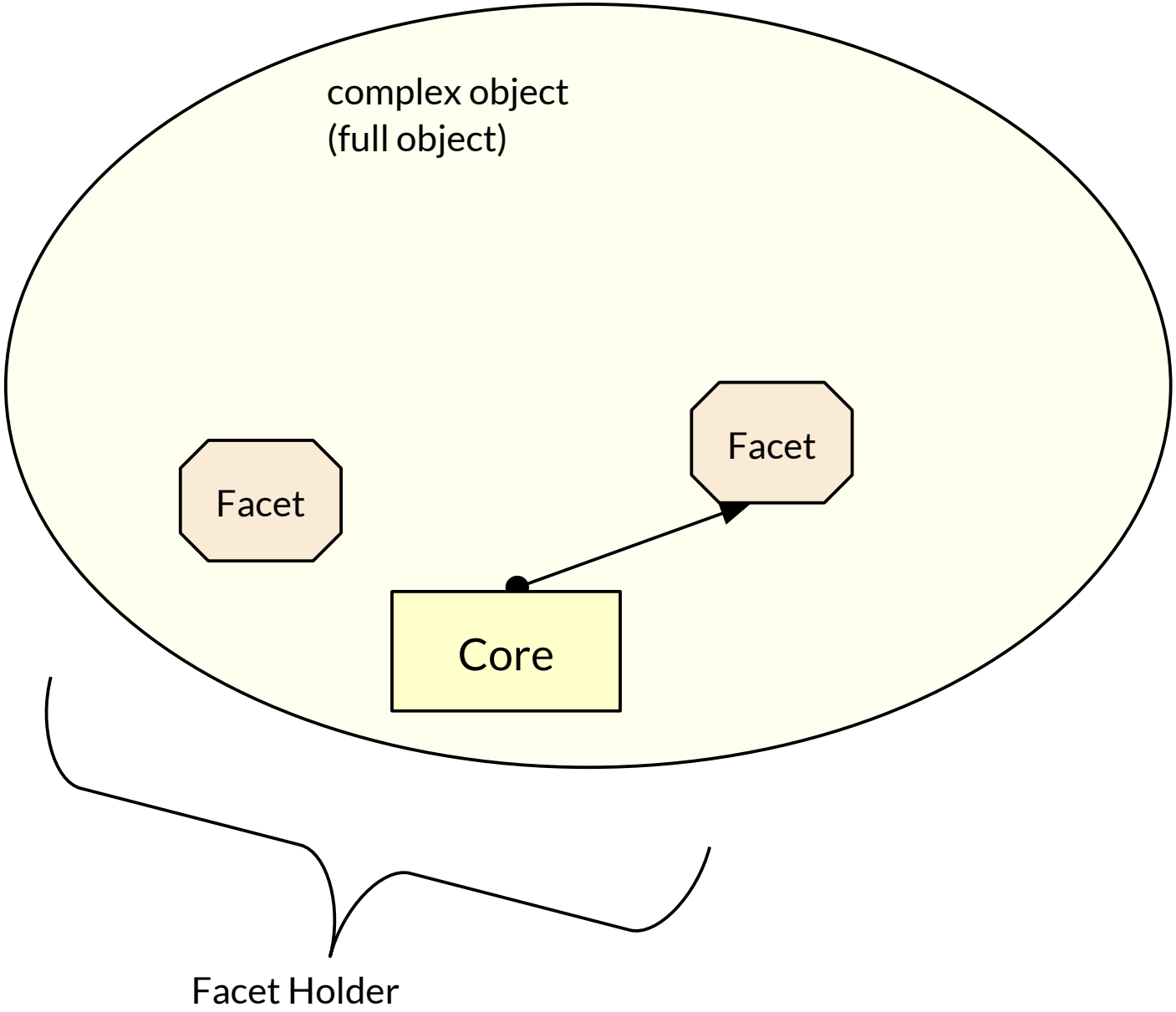
Einfache Realisierung durch Delegation

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



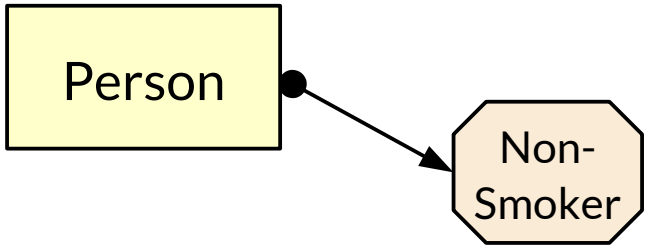
Powertype Service

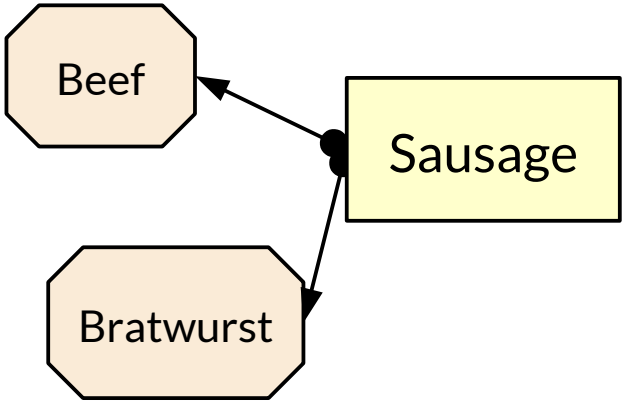
Facetten, repräsentiert als Unterobjekte

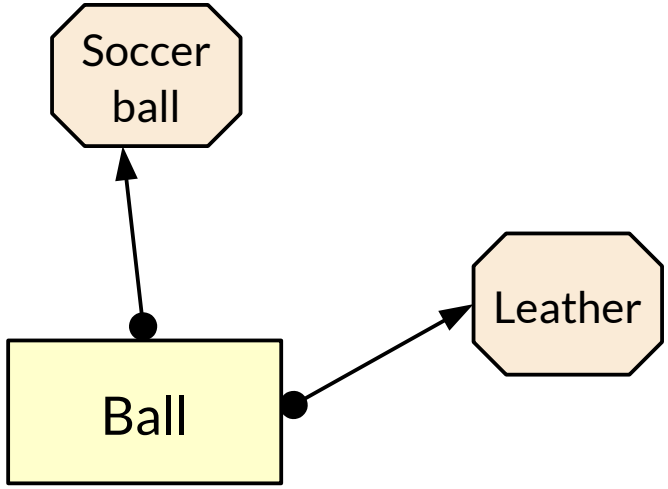


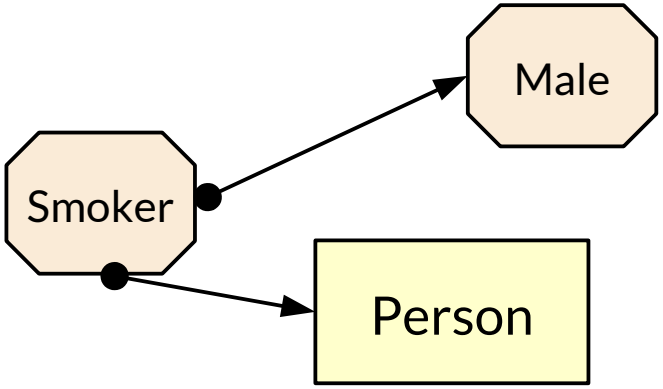
Facetten, repräsentiert als Unterobjekte

- non-founded; rigid









32.A.2 Phasen als Typen

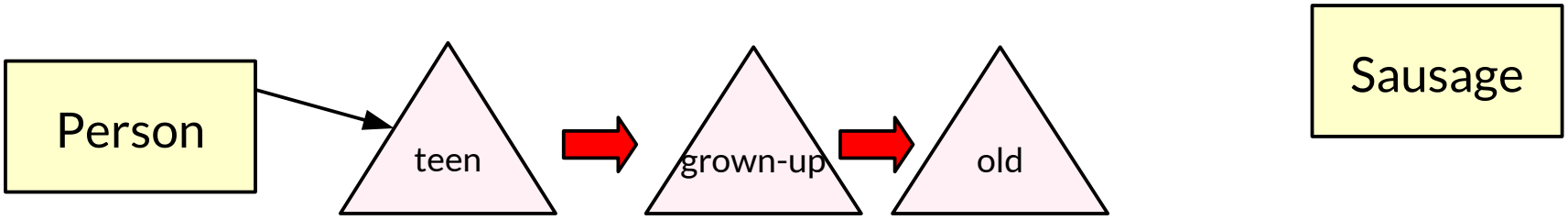
(optional)

Phasentypen

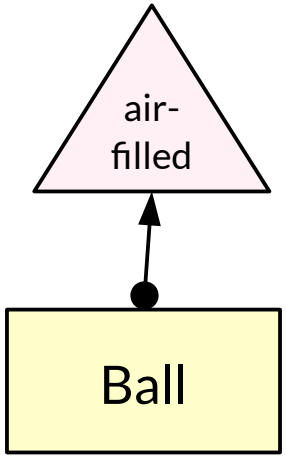
- Ein **Phasenobjekt** ist ein Unterobjekt, das eine Lebensphase (Zustand) eines komplexen Objektes beschreibt
- Ein **Phasentyp** charakterisiert die Lebensphase eines Objektes in seinem Lebenszyklus
  - Ein Phasentyp ist nicht-rigide, da er sich ändert im Laufe des Lebens

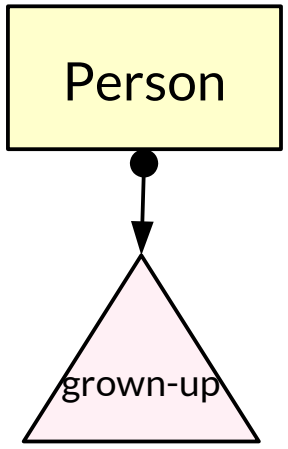
Was sind Phasen?

- Phasen sind nicht-fundiert, nicht-rigide Typen

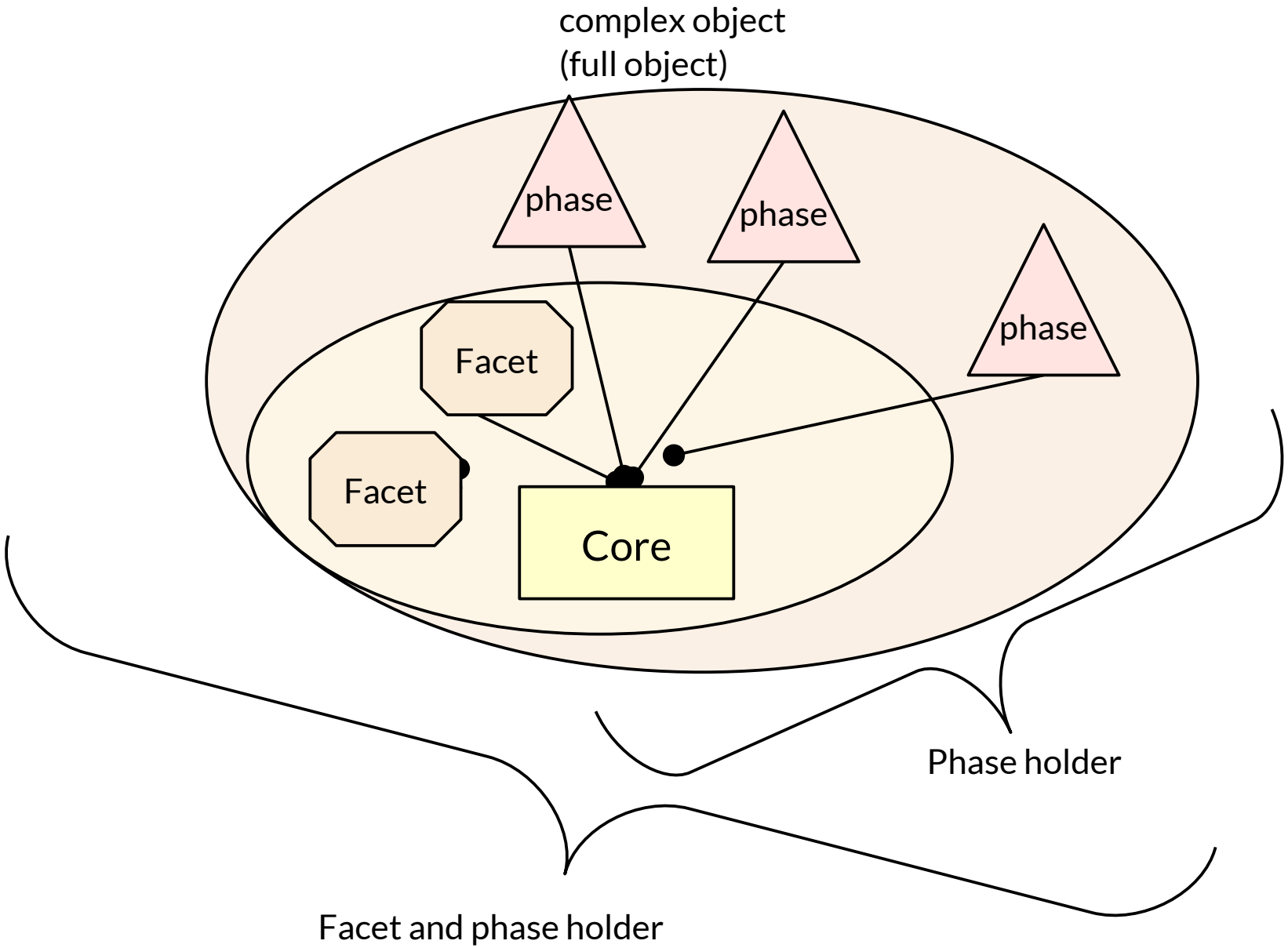


States in a lifecycle





Komplexes Objekt mit Facetten und Phasen



The End