43. Querschneidende Verfeinerung mit Plattformkonnektoren

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 16-0.1, 09.07.16

1) Einteilung in TAM
2) Verfeinerung mit Kollaborationen
3) Plattformverfeinerung mit Plattform-Konnektoren
4) Abbildung der Integrates-Relation
5) Gesamtbild der Verfeinerung

Objektorientierter Entwurf (Object-Oriented Design, OOD)

1) Einführung in die objektorientierte Softwarearchitektur
    1) Modularität und Geheimnisprinzip
    2) Entwurfsmuster für Modularität
    3) BCED-Architekturstil (3-tier architectures)
2) Verfeinerung des Entwurfsmodells zum Implementierungsmodell
    1) Anreicherung von Klassendiagrammen
    2) Verfeinerung von Lebenszyklen
    3) Querschneidende Verfeinerung mit Object Fattening
3) Objektorientierte Rahmenwerke (frameworks)
4) Softwarearchitektur mit dem Quasar-Architekturstil

Literatur

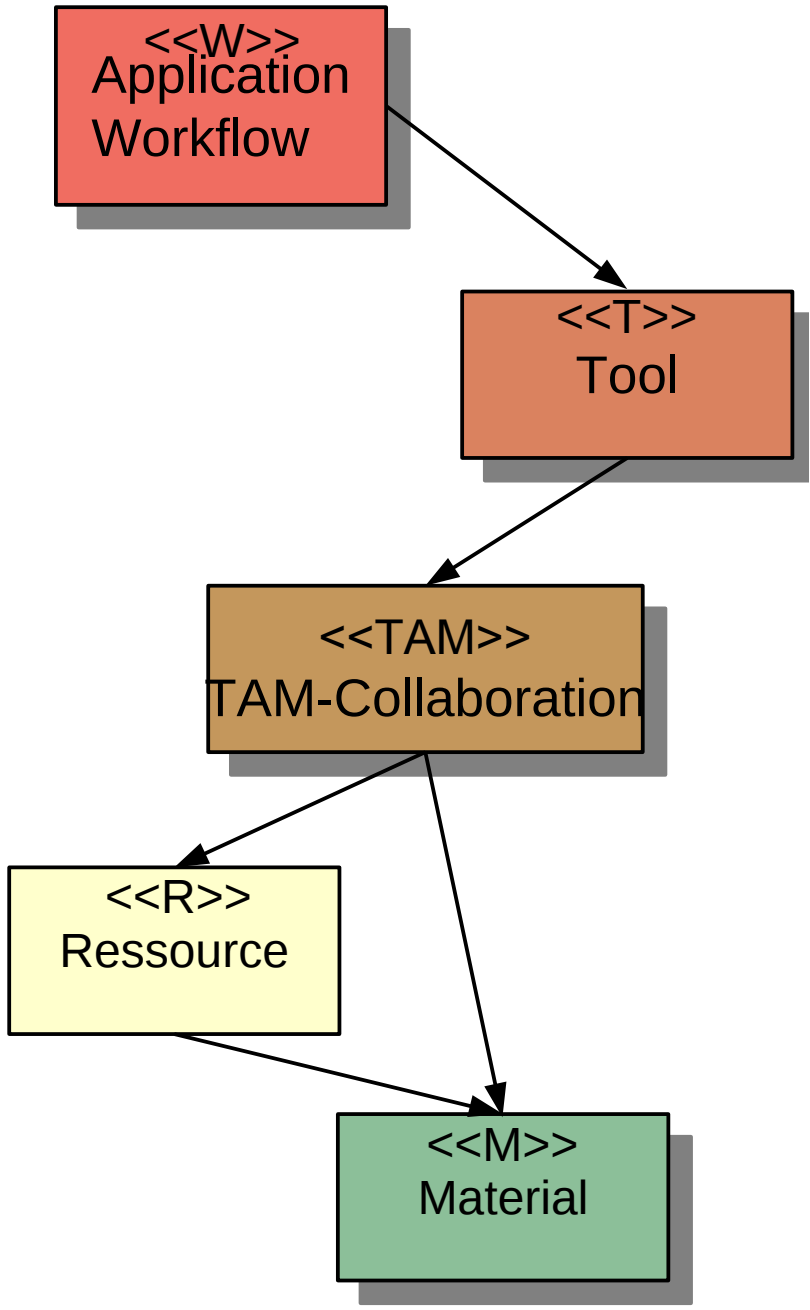
- Obligatorisch:
  - D. Riehle, H. Züllighoven. A Pattern Language for Tool Construction and Integration Based on the Tools&Materials Metaphor. PLOP I, 1995, Addison-Wesley.
  - OSGI Technical White Paper. www.osgi.org
- Fakultativ:
  - Heinz Züllighoven. Object-oriented construction handbook - developing application-oriented software with the tools and materials approach. dpunkt.verlag, 2005, ISBN 978-3-89864-254-5.

Perspektivenmodell TAM: Trennung von aktiven und passiven Komponenten

Tools-and-Materials [Züllighoven] ist ein Perspektivenmodell, das folgende Aspekte in einem Profil definiert:

1) Tools (aktive Prozesse)
2) Ressources (belegbar)
3) Materials (passive Daten, Schicht E)
4) TAM-Collaboration
5) Workflows koordinieren Tools

- Klassen, Module, Komponenten, Pakete sollten mit diesen Aspekten qualifiziert werden



43.1 Identifikation von Tools, Materials, zur Einordnung von Klassen in die Schichten

Was wird interaktiv (asynchron) aufgerufen?

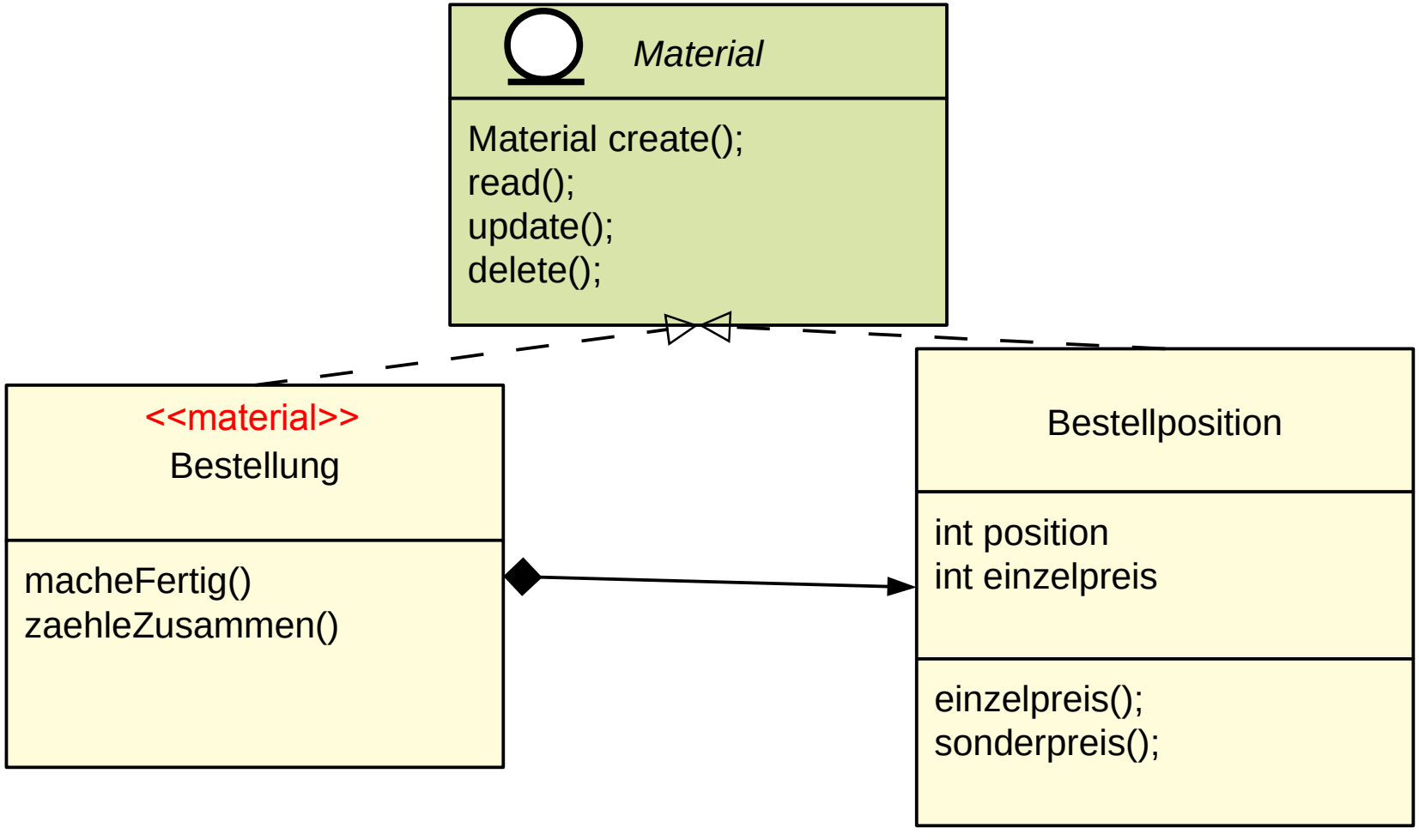
Was ist passiv?

Was muss belegt werden?

Welche Klasse wird in welche Schicht eingeordnet?

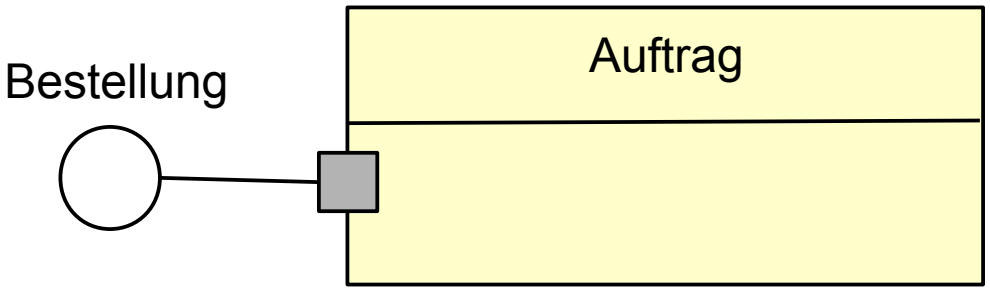
Material-Klassen und -Schnittstellen

- Materialobjekte sind passiv, d.h. werden von außen aufgerufen und geben den Steuerfluss nach außen hin zurück
- Materialobjekte können komposit sein (Muster Composite)
- Materialien folgen der CRUD-Schnittstelle



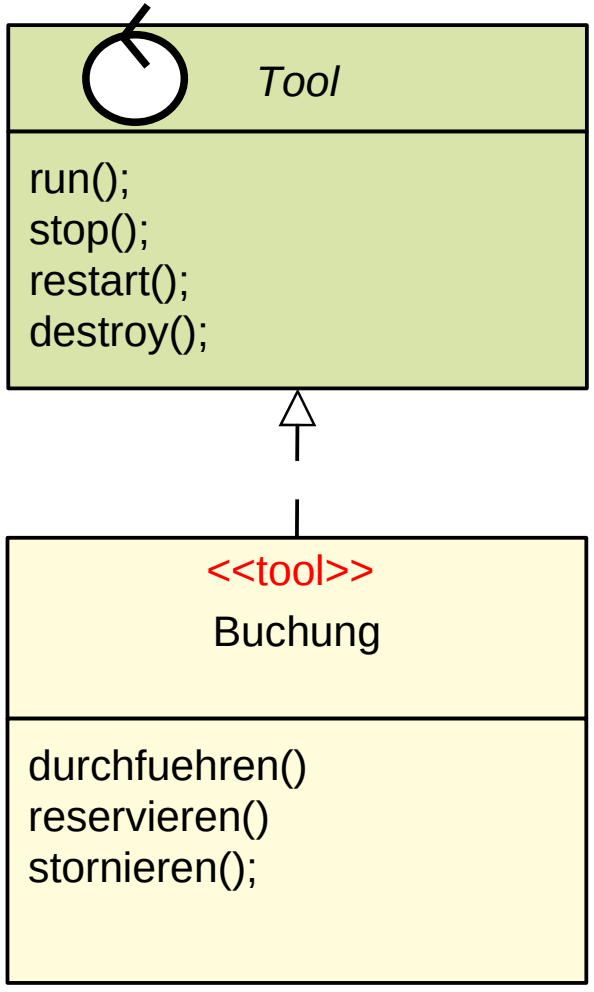
Material-Klassen und -Schnittstellen

- Materialien können in Ports auftauchen



Tool-Klassen und -Schnittstellen

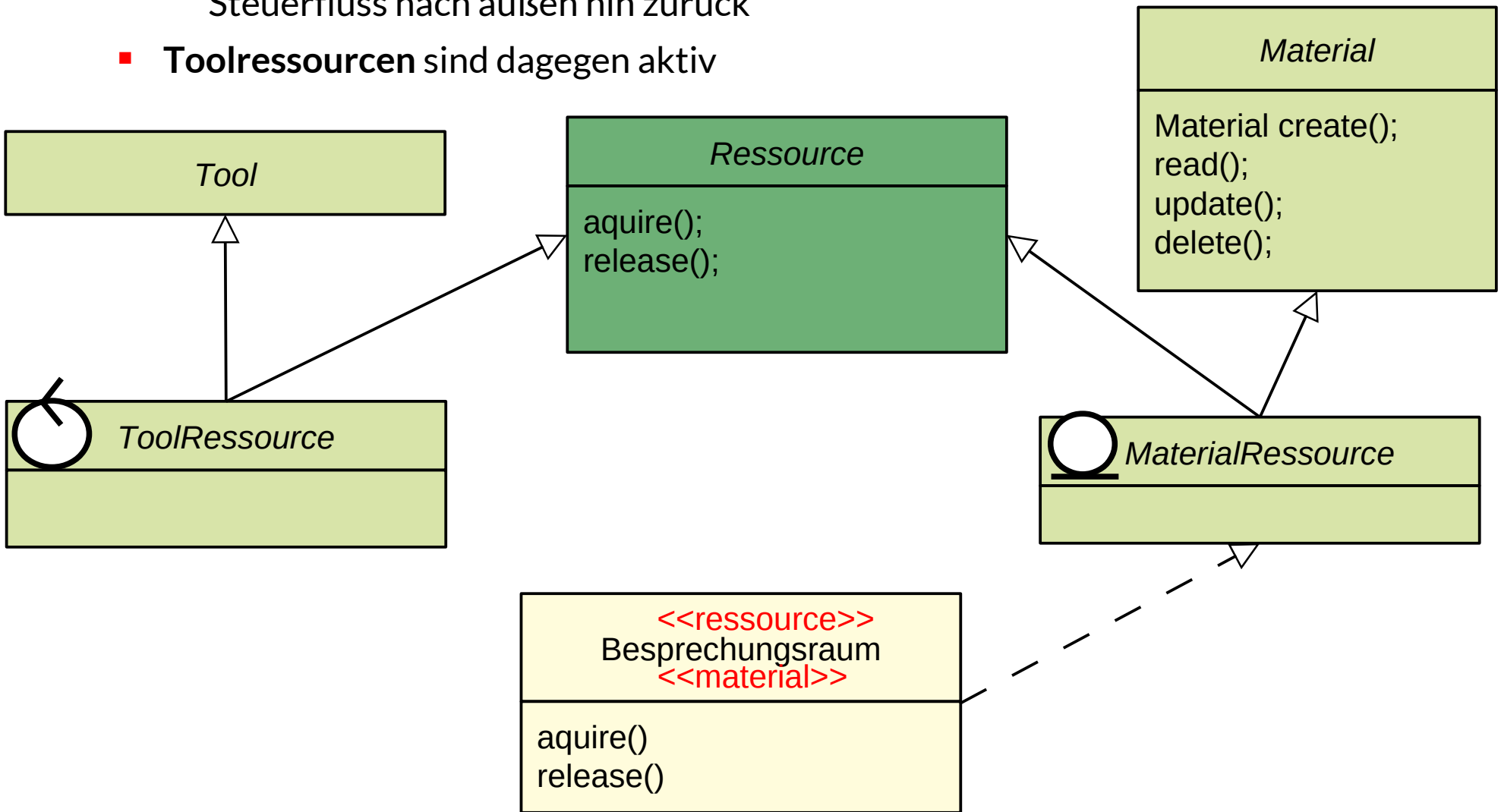
- Toolobjekte sind I.d.R. aktiv, besitzen eigenen Steuerfluss (thread, process)



[Züllighoven]

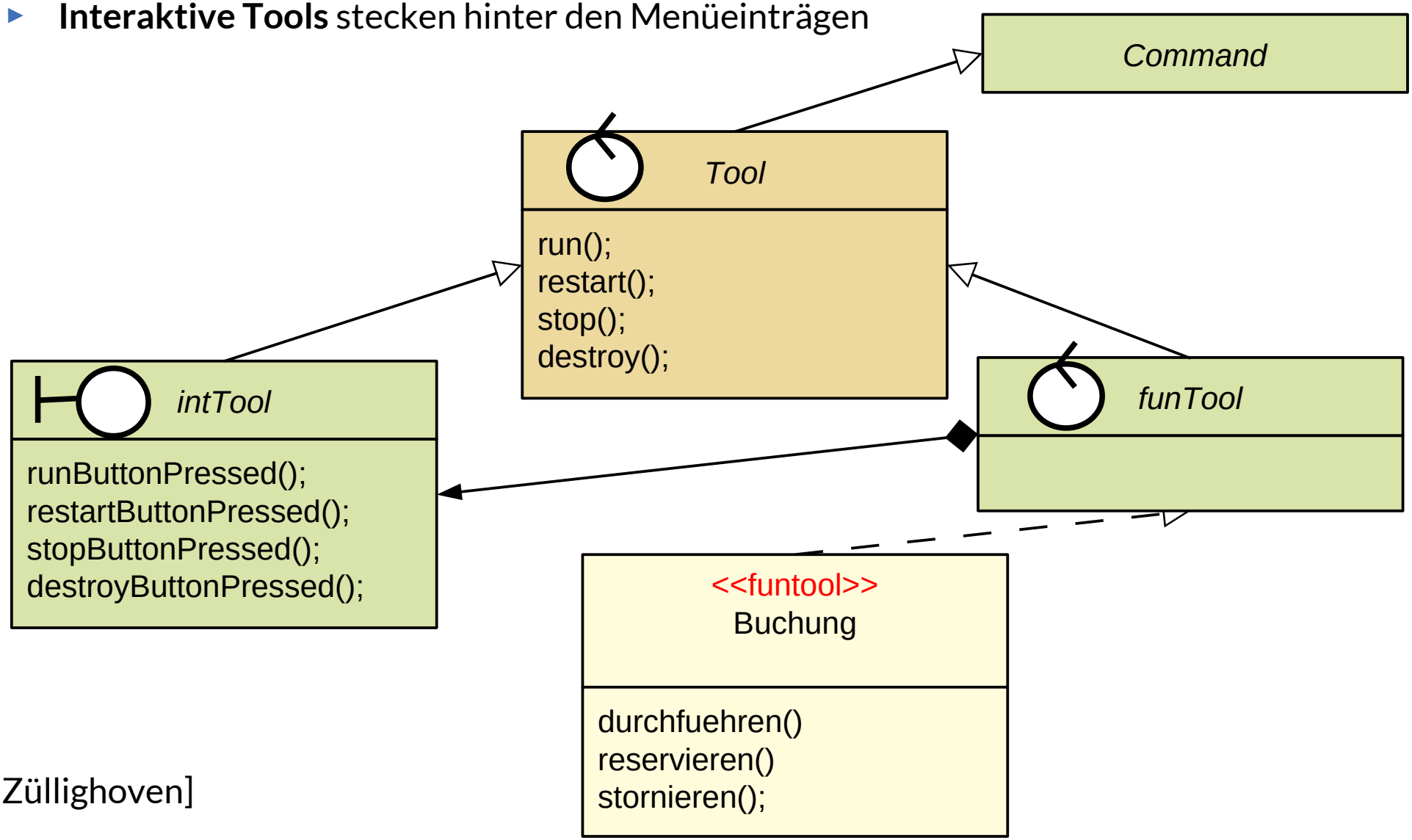
Ressourcen-Klassen und -Schnittstellen

- **Resourcenobjekte** sind Tools oder Materialien, die vor Nutzung zu *belegen* sind, d.h. sie müssen vor Nutzung alloziert und nach Nutzung freigegeben werden
  - **Materialressourcen** sind passiv, werden von außen aufgerufen und geben den Steuerfluss nach außen hin zurück
  - **Toolressourcen** sind dagegen aktiv



Tool-Klassen und -Schnittstellen

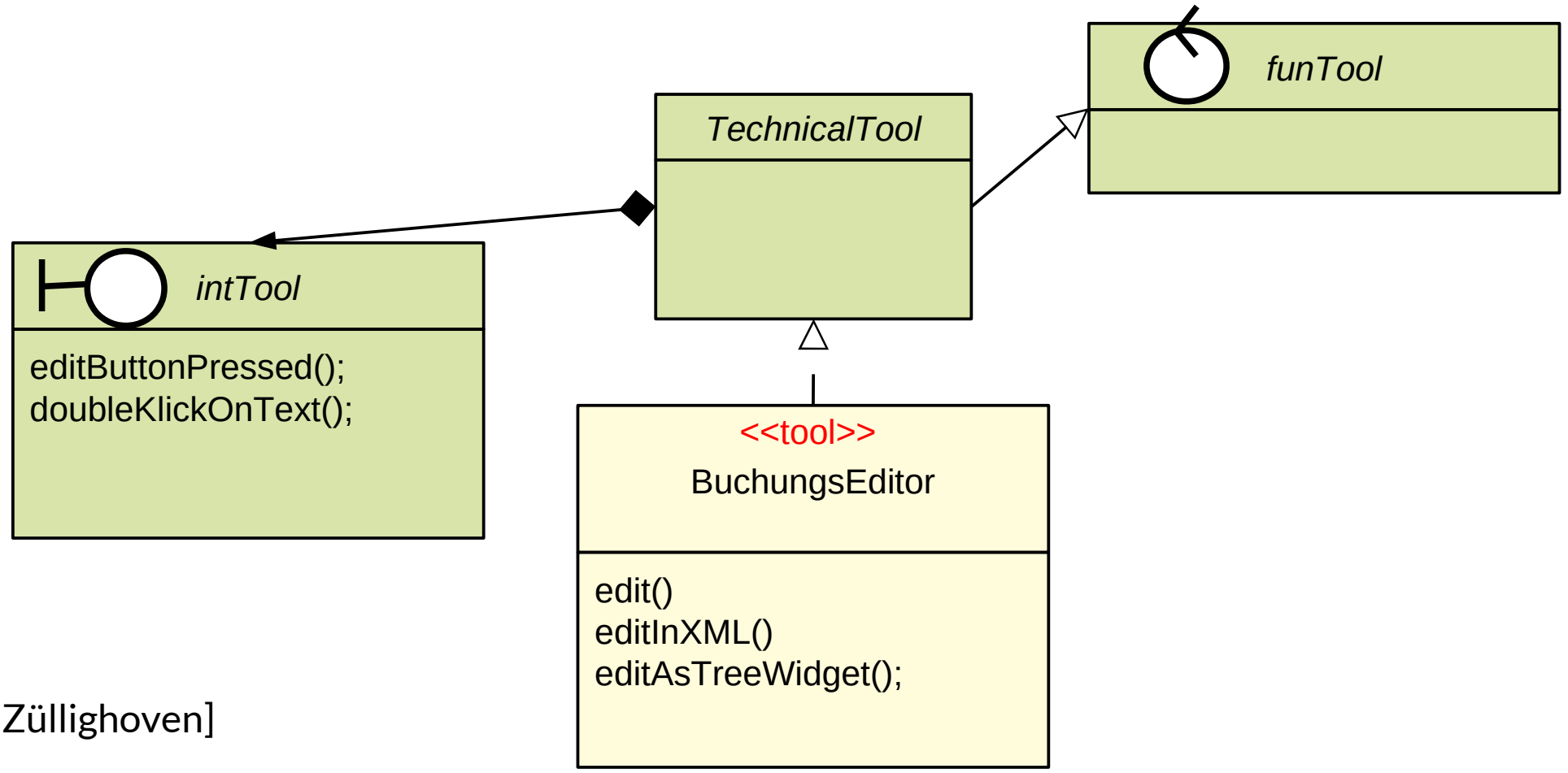
- Toolobjekte haben einen interaktiven Teil (intTool, boundary) und einen ausführenden, funktionalen Teil (funTool, control), der aus dem Command-Pattern abgeleitet ist
- **Interaktive Tools** stecken hinter den Menüeinträgen



[Züllighoven]

Technische Tool-Klassen und -Schnittstellen

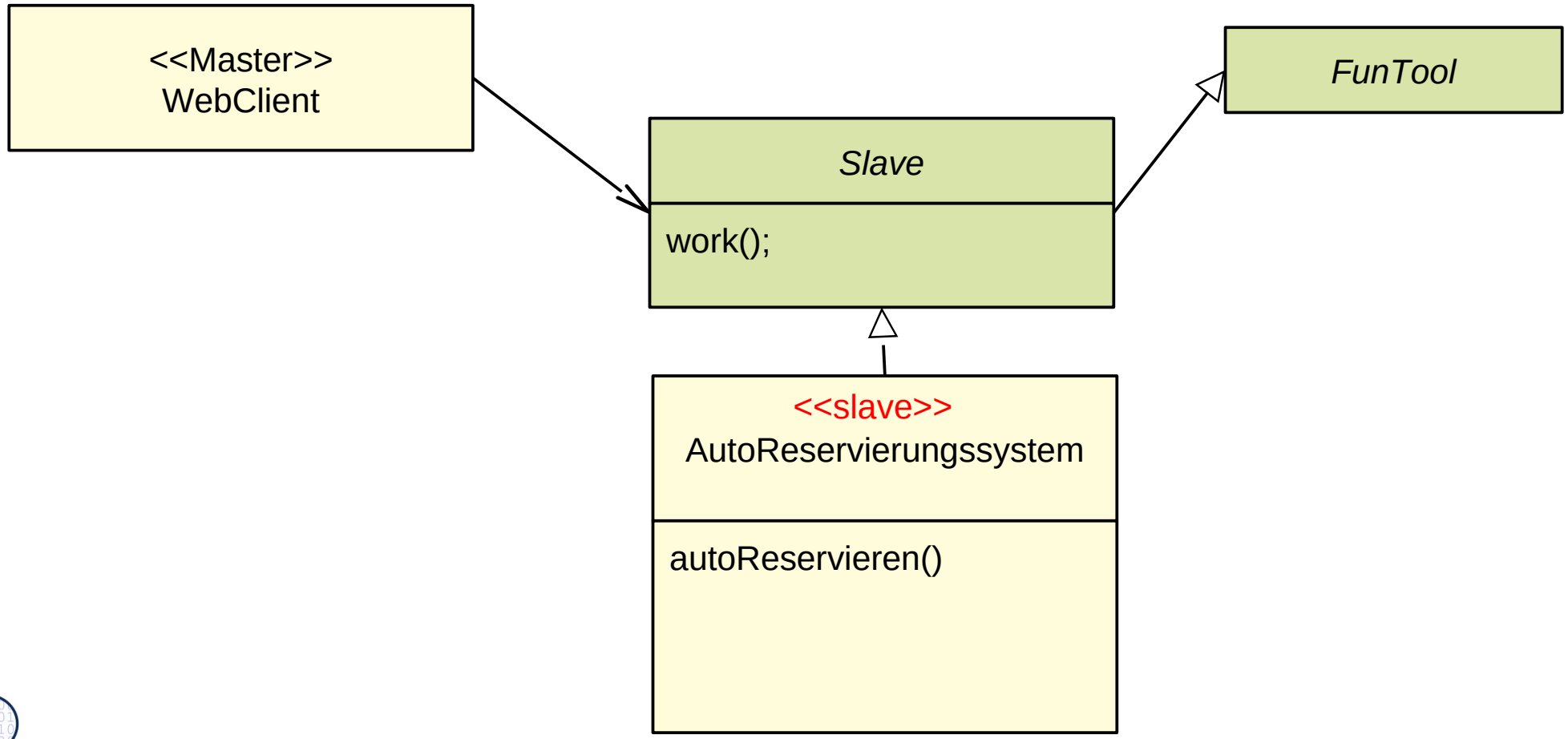
- Technische Tool sind funktionale Tools, die eine technische Funktionalität tragen, die nicht anwendungsunspezifisch ist
  - Bsp.: Editor, Lister, Inspektor, Browser, Verschlüsseler, Komprimierer, Optimierer
- Technische Tools verwalten das Material und bilden eine Schicht direkt über der Materialschicht



[Züllighoven]

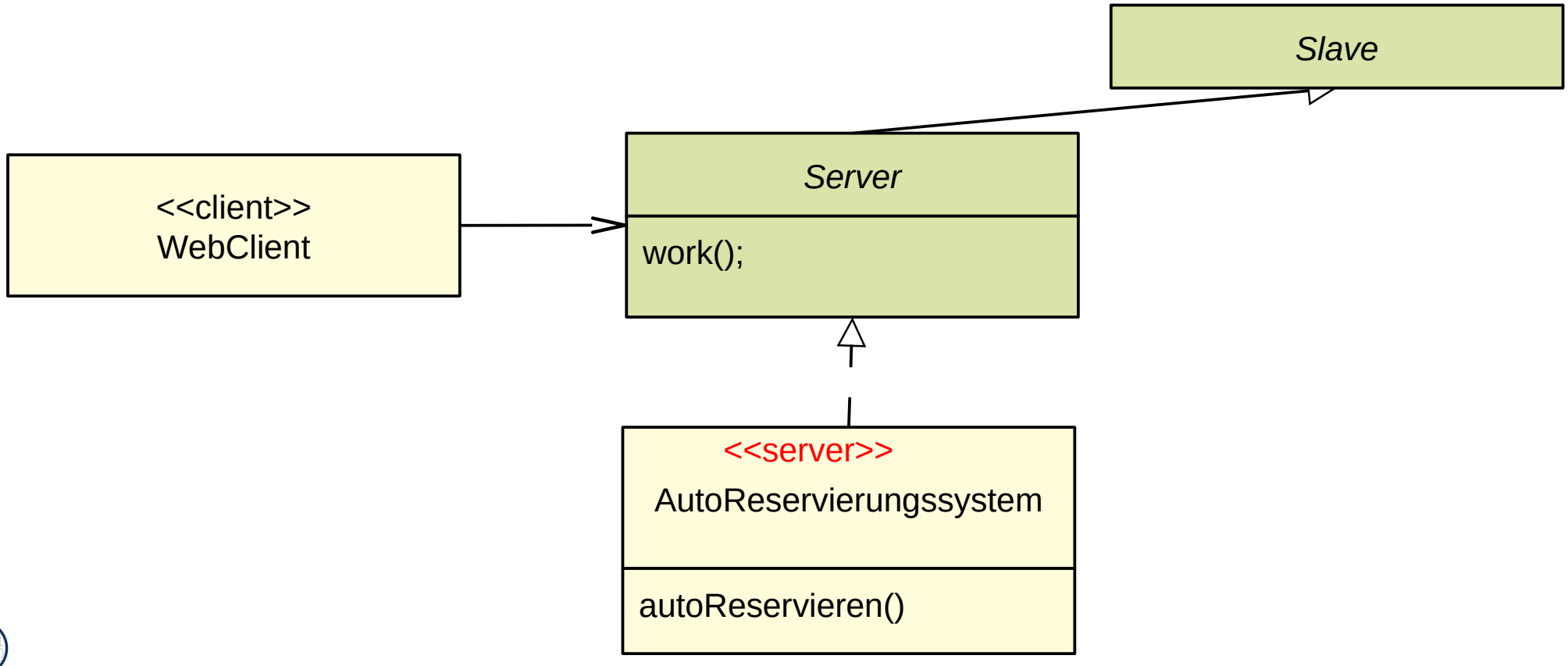
Slave-Klassen und -Schnittstellen

- Slave-Objekte sind passiv funktionale Tools. Sie werden beauftragt, laufen im batch ab (Design pattern “Master-Slave”)
- Slave-Objekte bilden also spezielle passive Tools (Kommandoobjekte)



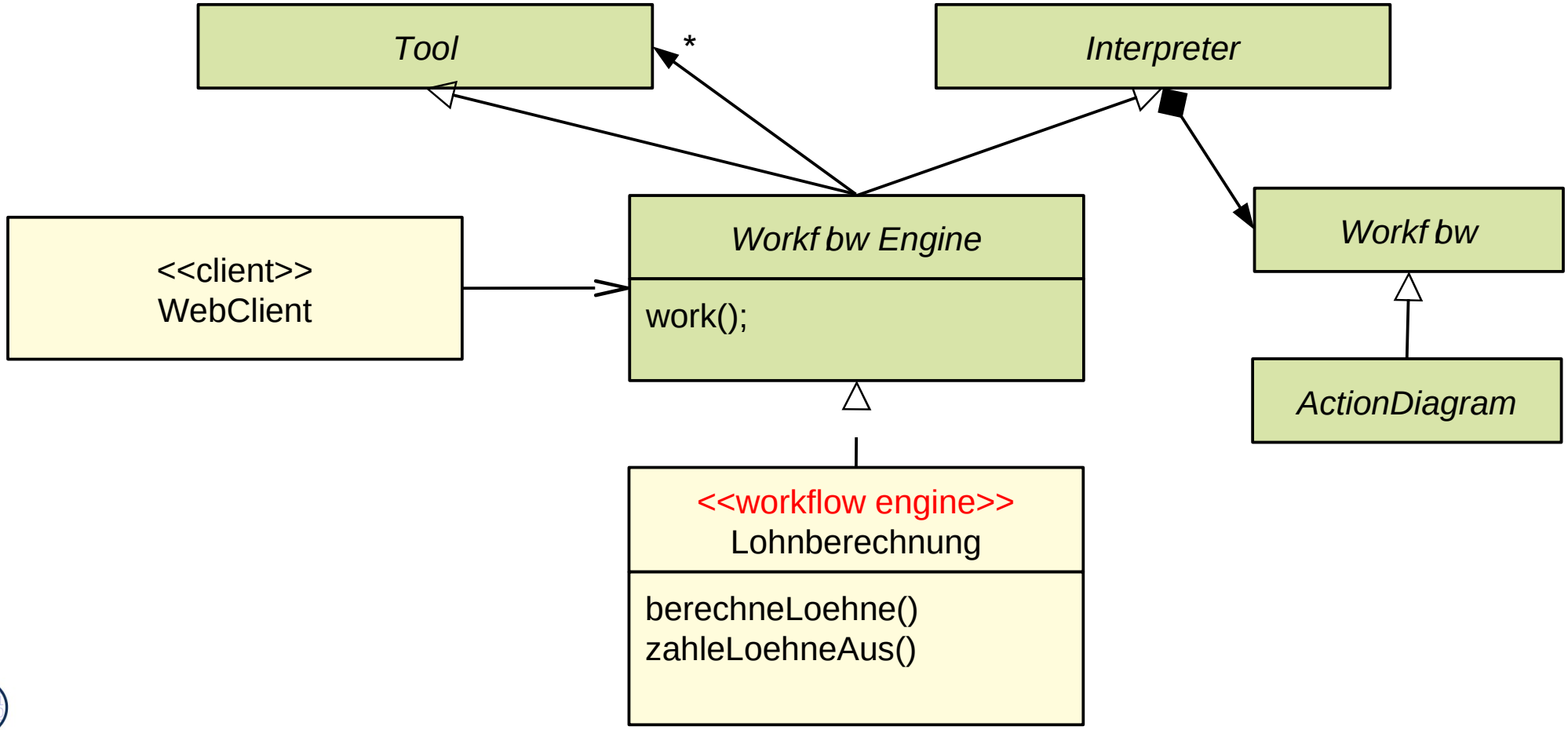
Server-Klassen und -Schnittstellen

- Serverobjekte sind spezielle Slaves, die von einem “Client” beauftragt werden (Design pattern “Client-Server”)
  - Sie können aber durchaus verborgenen eigenen Steuerfluss besitzen (thread, process) und damit mehrere Anfragen gleichzeitig bearbeiten
- Serverobjekte bilden also spezielle passive Slaves (Kommandoobjekte)



Workflow-Engine-Klassen und -Schnittstellen

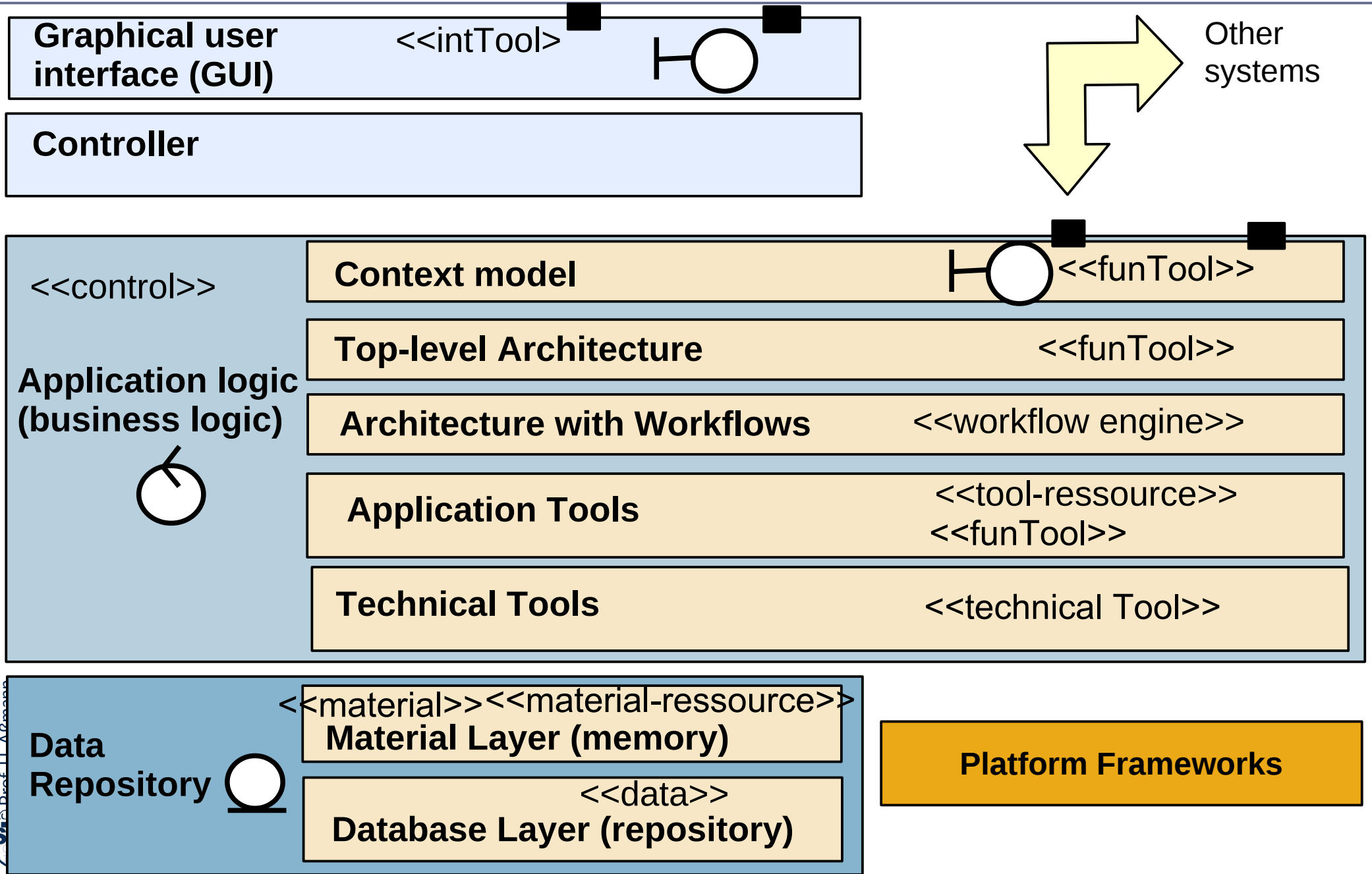
- Eine Workflow-Engine ist ein funktionales Tool, das einen komplexen Arbeitsablauf (Workflow) abarbeitet (interakiv oder batch)
- Workflow-Engines rufen andere Tools auf, setzen also auf ihnen auf
- Workflows werden durch Aktionsdiagramme (Aktivitätendiagramm Statechart, BPMN) beschrieben



Schichten und TAM-Klassifikation

- Die TAM-Klassifikation erlaubt uns, Klassen Schichten der Anwendung zuzuordnen.

Q7': Verfeinerte BCE-Schichtung eines Systems



43.2 Verfeinerungsbeispiel für Objektanreicherung in der Analyse

.. Verfeinerung durch Integration von Unterobjekten..

Teile und Rollen

Objektanreicherung in der Analyse

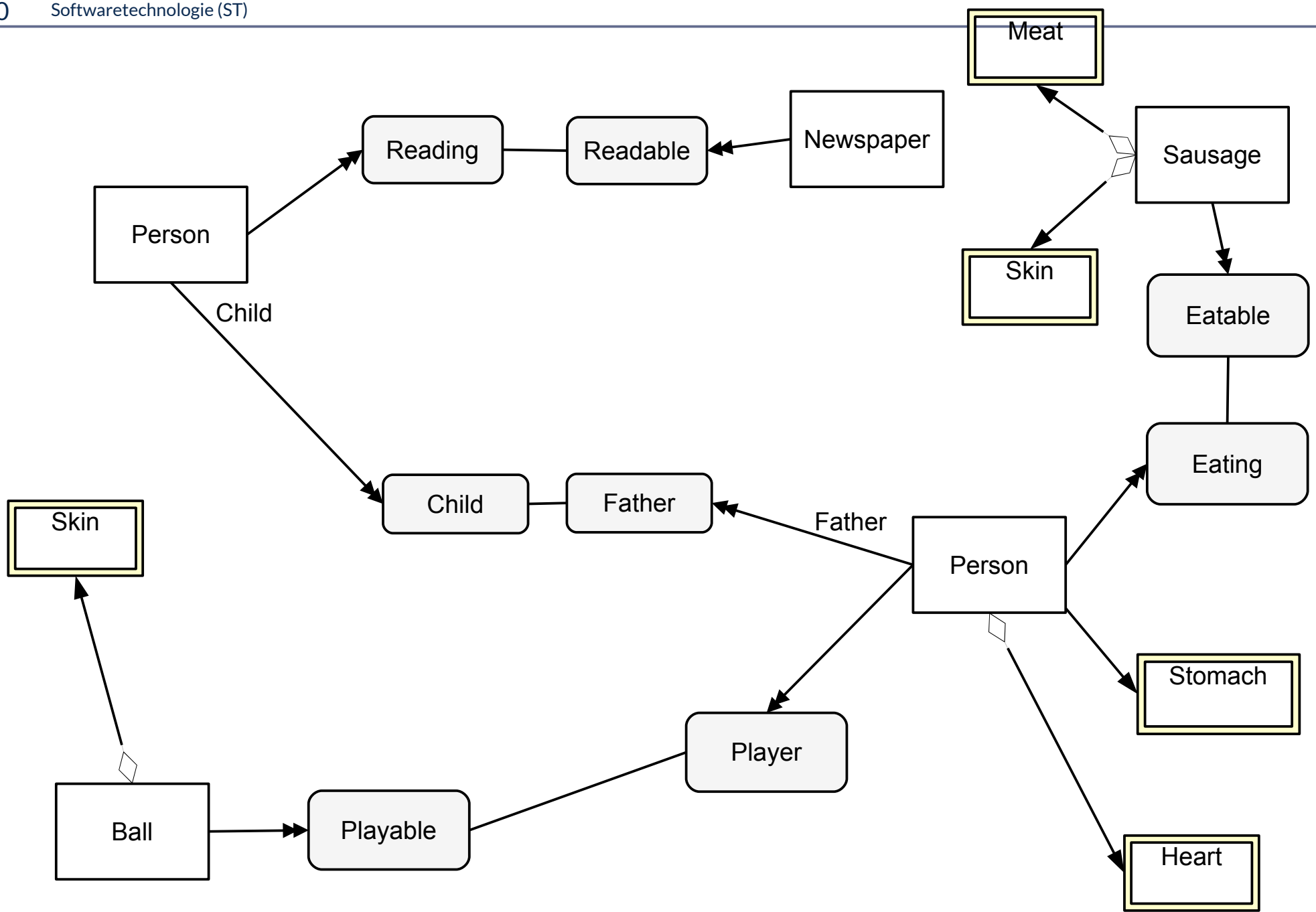
- **Objekt-Anreicherung (Object fattening) durch Unterobjekte** ist ein Verfeinerungsprozess, der an ein Kernobjekt aus dem Domänenmodell Unterobjekte anlagert, die
  - Teile ergänzen (Teile-Verfeinerung)
  - Rollen ergänzen (Konnektor-Verfeinerung), die Beziehungen klären zu
    - Plattformen (middleware, Sprachen, Komponenten-services)
    - Komponentenmodellen (durch Adaptergenerierung)
- Ziel: Entwurfsobjekte, Implementierungsobjekte

- Achtung. Wir nähern uns, nach vielen einzelnen Schritten, dem Höhepunkt der Vorlesung:

> Querschneidende **Objektanreicherung** ist der entscheidende Schritt bei der Verfeinerung von den Analyse- und Entwurfsmodellen zum Implementierungsmodell und zur Implementierung.

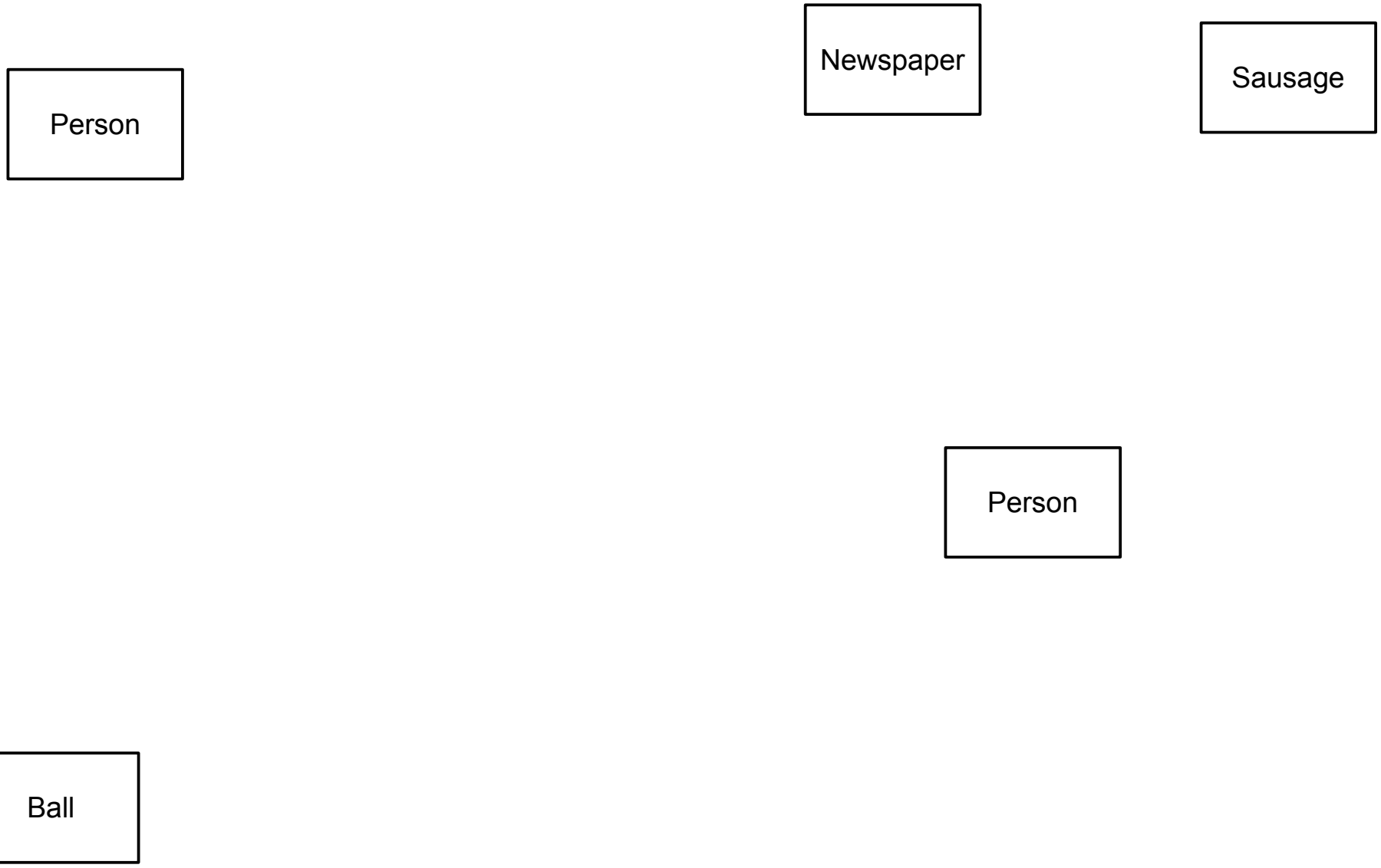
- Gründe:
  - Der objekt-orientierte Software-Entwicklungsprozess startet mit einer Simulation der realen Welt durch Objekte, die zu Systemobjekten erweitert werden und dabei durch technische Informationen angereichert werden müssen

Personen-Analysemodell mit Rollenobjekten und Teilen – Wie komme ich bloß dahin?



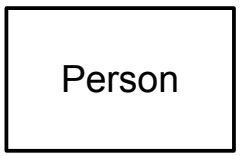
Mit Verfeinerung durch Integration von Unterobjekten (Objektanreicherung, Object Fattening)

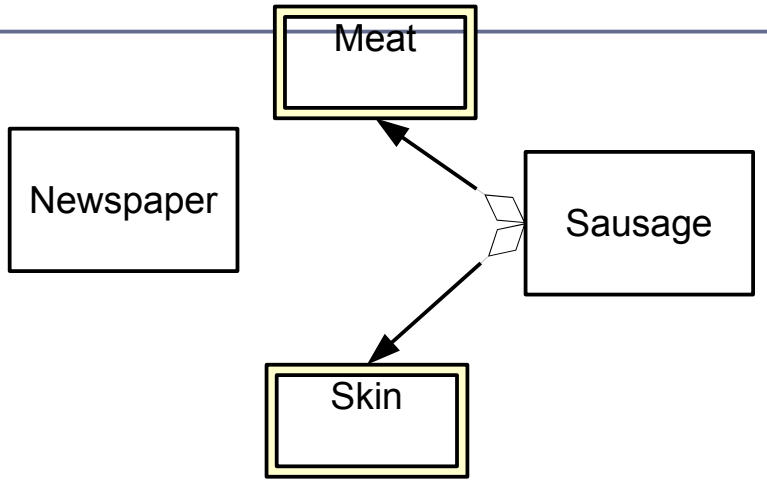
- Rohzustand: Identifikation der natürlichen Typen (in dem Domänenmodell)

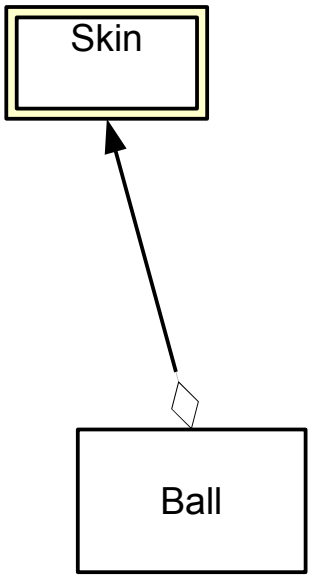


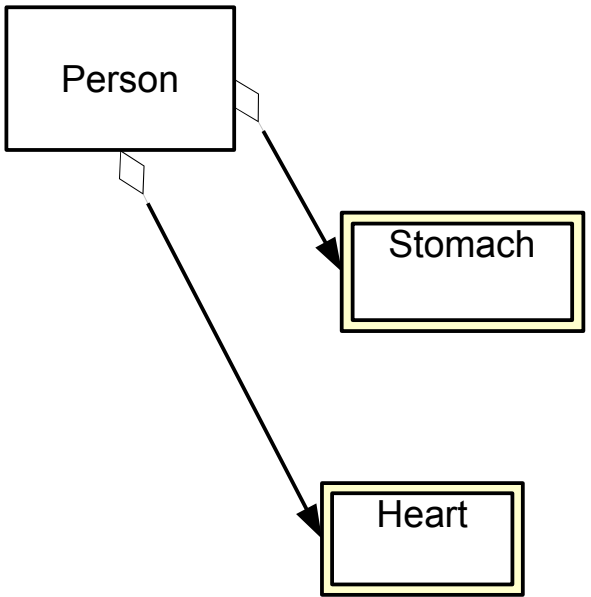
Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 1: Teile-Verfeinerung



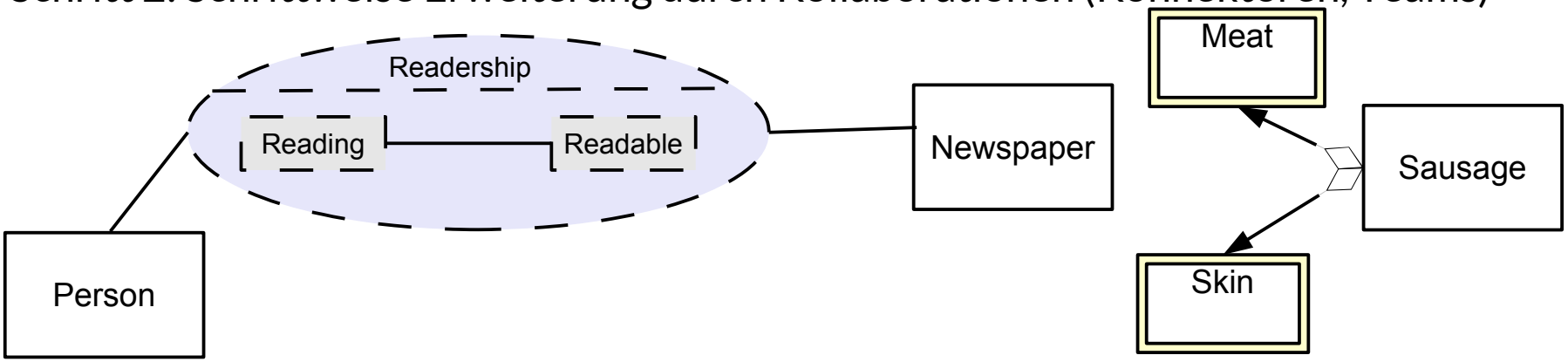


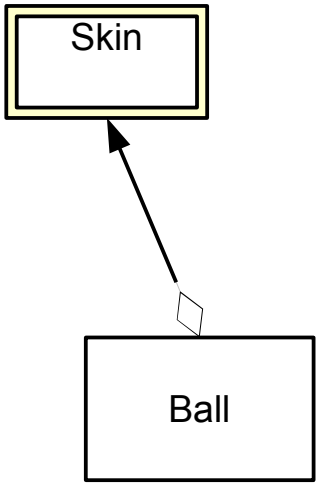


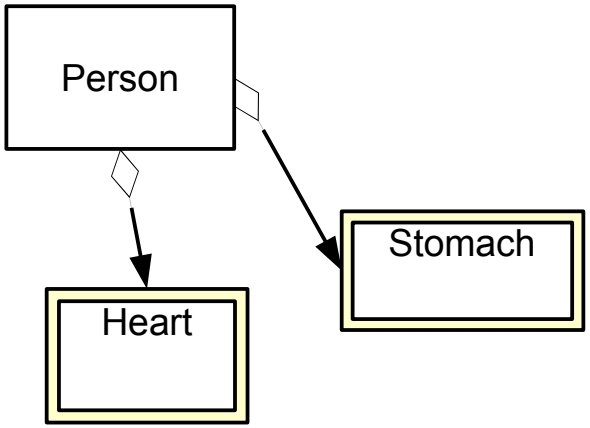


Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 2: Schrittweise Erweiterung durch Kollaborationen (Konnektoren, Teams)

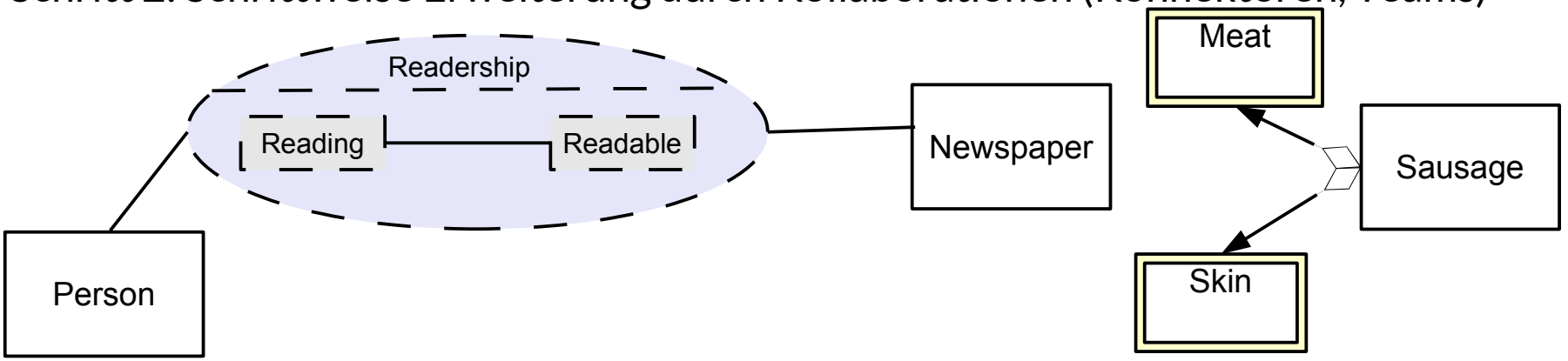


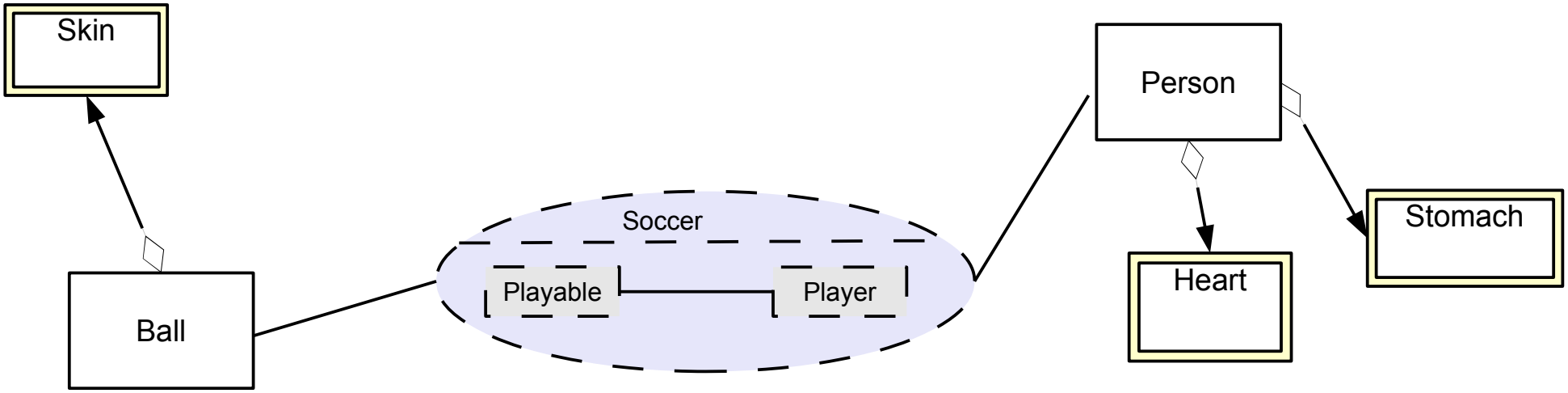




Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

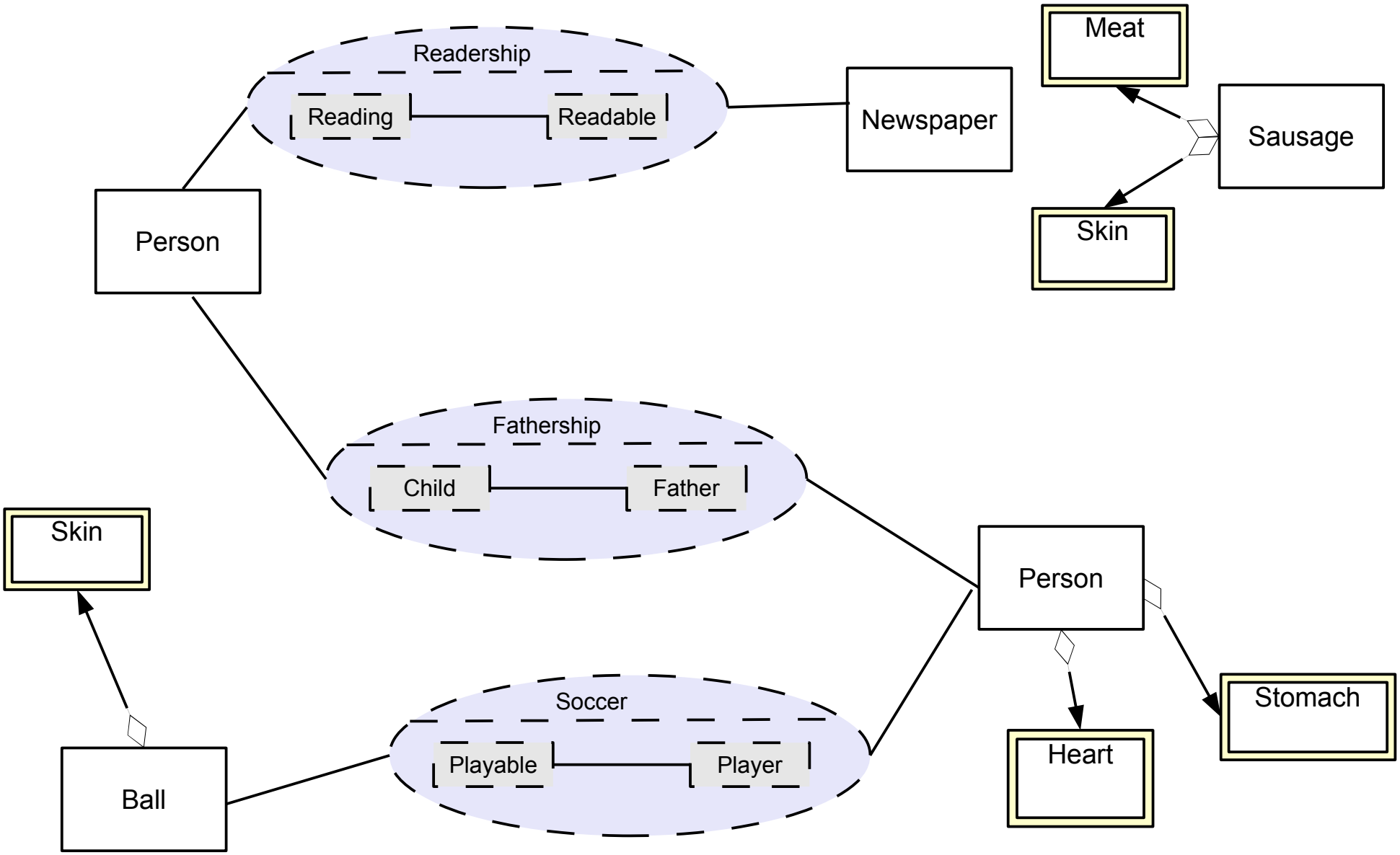
- Schritt 2: Schrittweise Erweiterung durch Kollaborationen (Konnektoren, Teams)



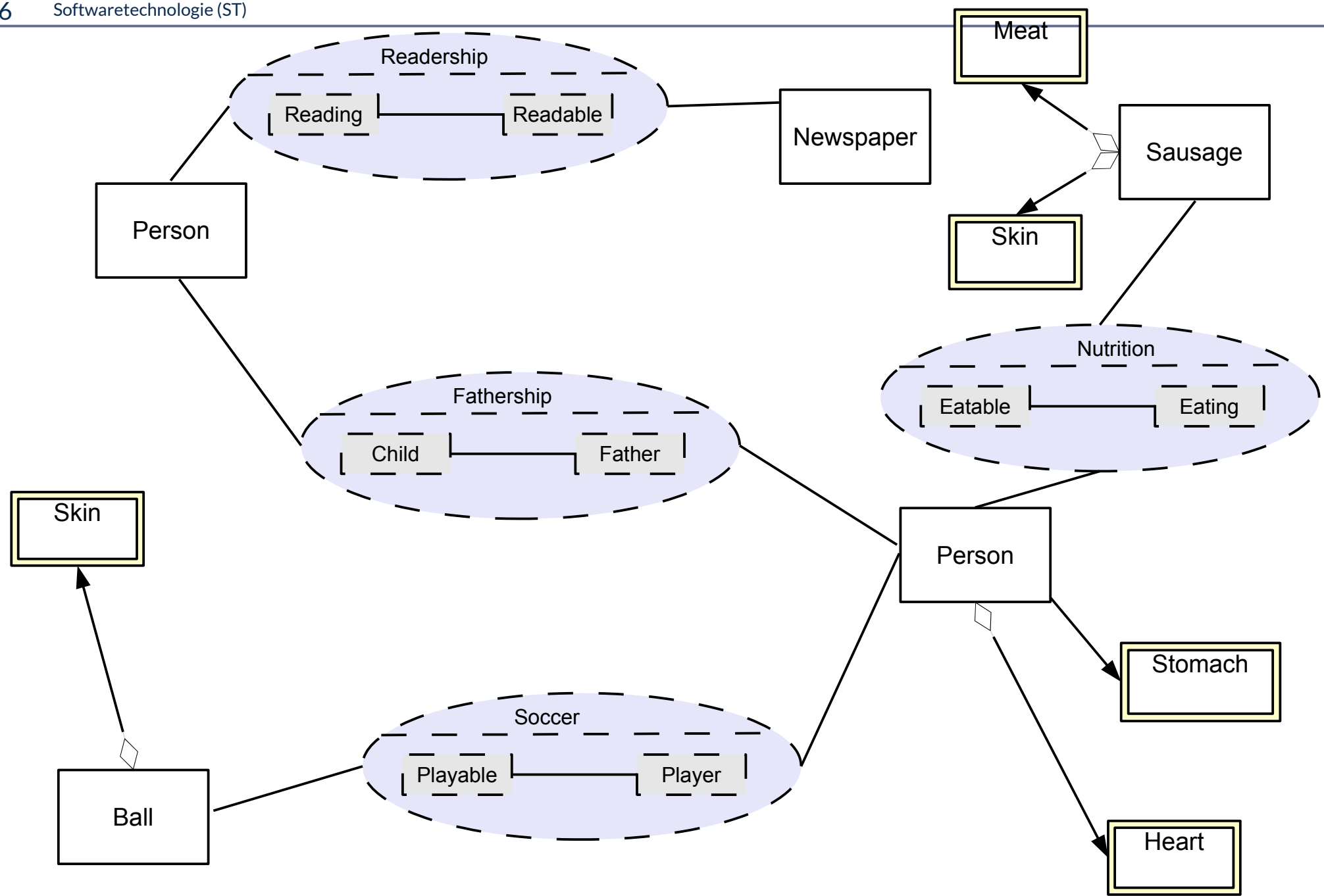


Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 2: final: alle Kollaborationen



Personen-Analysemodell – Angereichert durch Einziehen von querschneidenden Kollaborationen

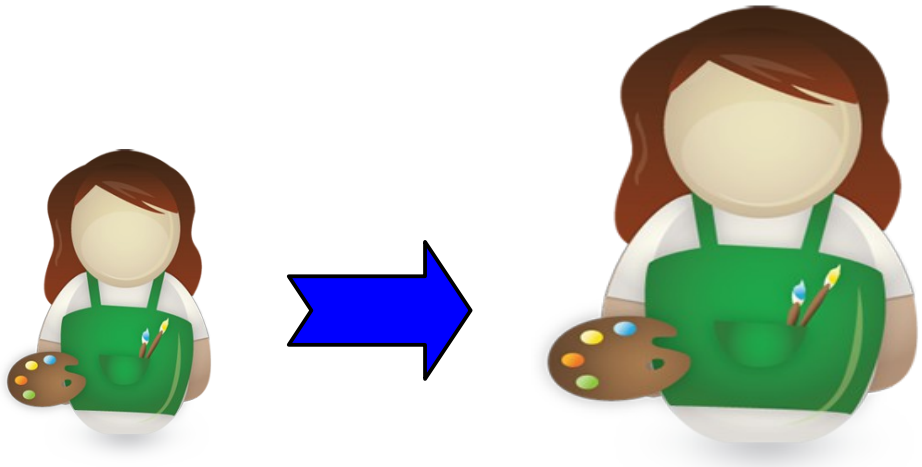


43.3 Objektanreicherung mit Plattforminformation (Querschneidende Verfeinerung für Plattformen)

.. Verfeinerung durch Integration von Unterobjekten..

Plattform-Objektanreicherung

- Plattform-Objektanreicherung ist ein Objektanreicherungs-Prozess zur Entwurfszeit, der Konnektoren mit plattform-spezisches Verhalten ergänzt (Plattform-Verfeinerung)
- Die hinzugefügten Konnektoren mit ihren Rollen und Kollaborationen klären Beziehungen zu
  - Plattformen (Betriebssystem, Middleware, Sprachen, Komponenten-services)
  - Komponentenmodellen (durch Adaptergenerierung)
- Ziel: Entwurfsobjekte, Implementierungsobjekte



Objektanreicherung – Weitere Schritte im Entwurf

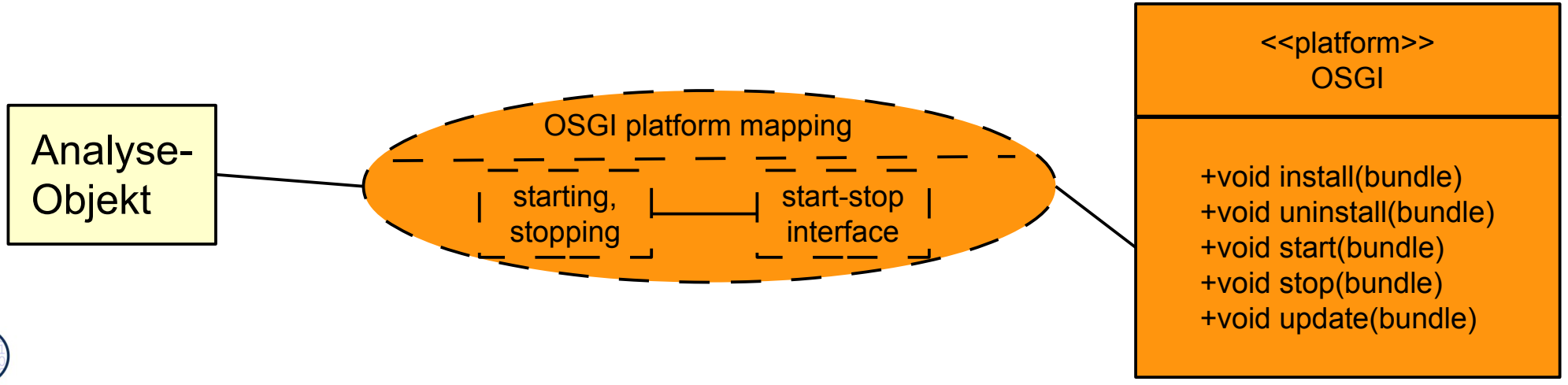
- Bei Entwurfsobjekten kommt hinzu:
  - Finden von **Plattform-Kollaborationen,** fundierte Unterobjekte, die das spezifische Verhalten bezüglich eines Plattformobjektes kapseln
- Beim Implementierungsmodell kommt hinzu:
  - Realisierung der Kollaborationen und der Integrationsrelation

Plattformanreicherung – Weitere Schritte im Entwurf

- Teile- und Rollenverfeinerung startet schon in der Analyse
- Bei Entwurfsobjekten kommt **Plattformanreicherung** hinzu:
  - Finden von **Plattform-Konnektoren,** die plattform-fundierte Unterobjekte anlagern, die das spezifische Verhalten bezüglich eines Plattformobjektes kapseln
    - **Plattformfähigkeiten (platform abilities, platform-founded types)** bilden fundierte Typen, die die Beziehungen zu Plattformen klären
    - **Komponentenadapter (component-model-founded adapters)** klären die Beziehung zu Komponentenmodellen
- Ziel im Entwurf: Implementierungsobjekte ableiten
  - Rollen ergänzen, die Beziehungen klären zu
    - Plattformobjekten (middleware, Sprachen, Komponenten-services)
    - Komponentenmodellen (durch Adaptergenerierung)
  - Realisierung der Integrationsrelation
- Einfache Implementierung durch Konnektoren oder Entwurfsmuster

Plattformobjekte und -konnektoren

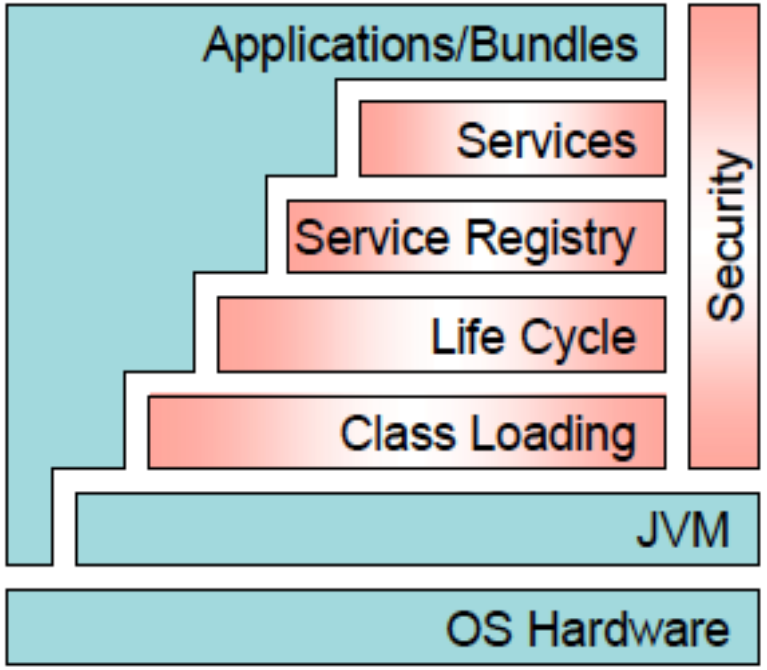
- Ein **Plattformobjekt** ist ein Objekt eines Plattform-Frameworks, das wesentliche Laufzeitfunktionalität bietet und auf die eine Software angepasst werden muss
  - Bietet Schnittstelle an bzgl. bestimmter Funktionalität, z.B. abstrakte Maschine (Interpretierer)
  - Variabel: je nach Maschine, Middleware, Betriebssystem, Datenbank, Programmiersprache unterschiedlich ausgeprägt
- Die Kollaboration mit der Plattform wird durch einen Konnektor zum Plattformobjekt, dem **Plattformkonnektor**, ausgedrückt
- OSGI: Komponentenplattform www.osgi.org
  - im Handy, 5er BMW, in Eclipse 3.0, Shell home automation HomeGenie
  - Ein *bundle (Komponente)* paketiert verschiedene Klassen



Plattformobjekt OSGI

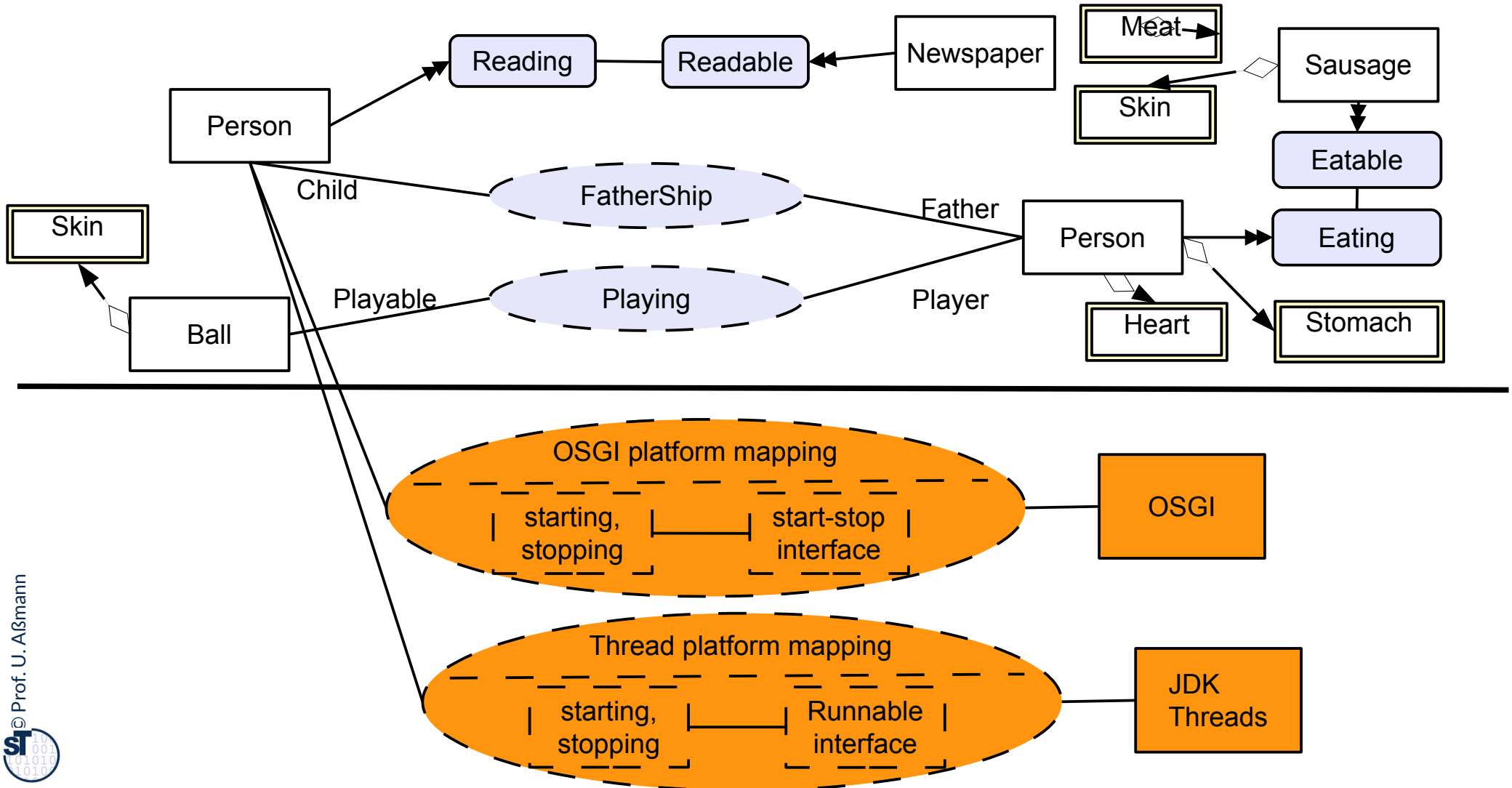
- OSGI bietet 5 Schnittstellen (rot)
  - Klassenlader (für Ersetzung von bundles)
  - Lebenszyklus (life cycle) von *bundles* (Paketen von Klassen, mit zip gepackt und verschickt)
  - Register (service registry): dient zum Registrieren von Bundles und ihren Zuständen
  - Dienste (services) verschiedener Art
  - Sicherheitsfunktionalität

- [OSGI Technical White Paper]



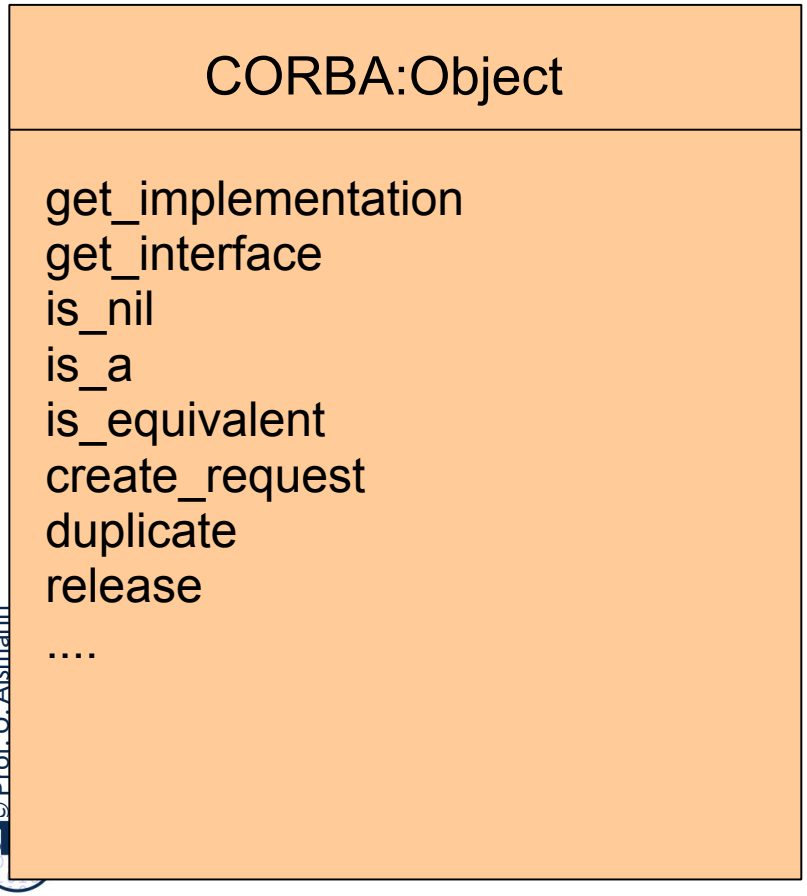
Mit Verfeinerung durch Plattform-Konnektoren (platform fattening)

- Plattform-Konnektoren beschreiben die Beziehungen zu Plattformobjekten sowie die Interaktion der Anwendungsobjekte mit ihnen (orange; Analyse-Konnektoren: lila)
- Plattformobjekte können als Alternativen existieren (hier OSGI, JDK threads) für die Plattform "Lebenszyklus"



Plattform CORBA: CORBA:Object

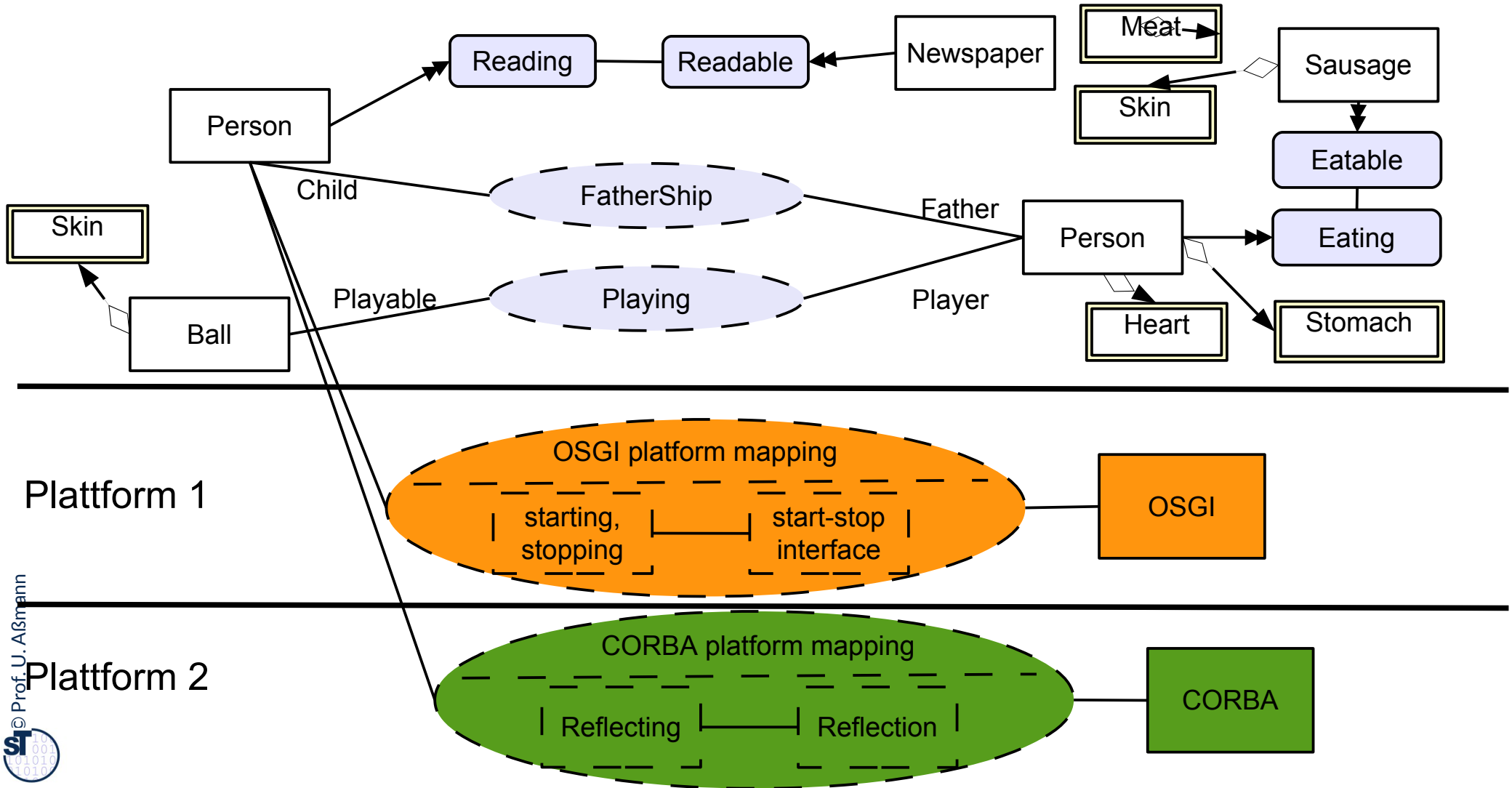
- CORBA bildet eine Komponentenplattform für heterogen programmierte Systeme



- In der Klasse CORBA:Object wird elementare Funktionalität einer CORBA Komponente definiert
  - heterogen benutzbar über viele Sprachen hinweg
- CORBA untertützt Reflektion:
  - get_interface liefert eine Referenz auf ein "Schnittstellenobjekt"
  - get_implementation eine Referenz auf eine "Implementierung" (Klassenprototyp)

Mit Verfeinerung durch mehrere Plattform-Konnektoren verschiedener Plattformen

- Plattform-Verfeinerung kann auf verschiedenen Stufen ablaufen, und somit verschiedene Plattformen behandelt werden
- Plattformkonnektoren werden stufenspezifisch eingesetzt und können gegen Varianten ausgetauscht werden



Das Portabilitätsgesetz

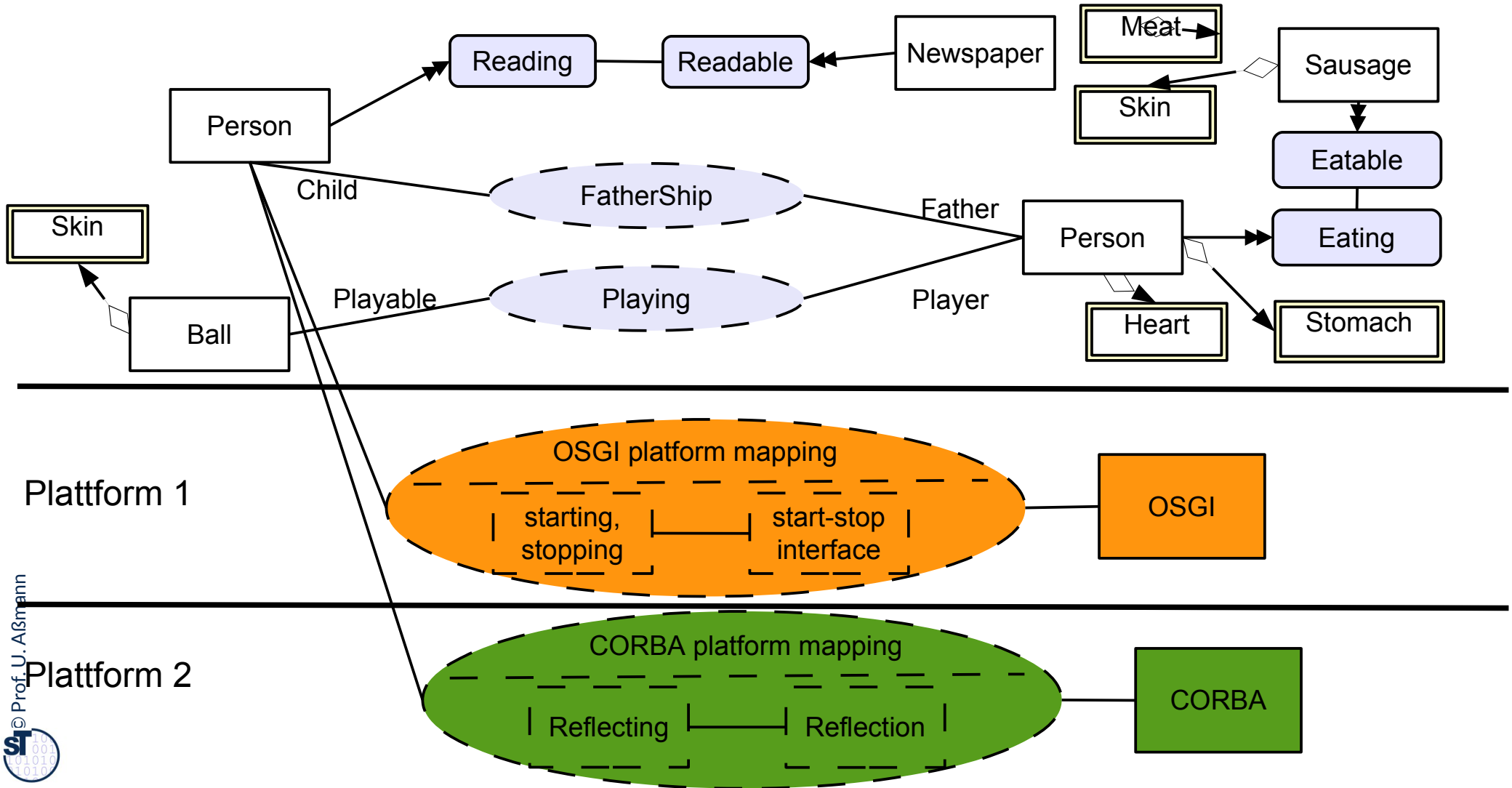
Kapselt man Plattformabhängigkeiten in einen Plattformkonnektor, können sie leicht ausgetauscht werden und die Software wird portabel.

43.4 Abbildung der Integrationsrelation auf klassische Programmiersprachen

.. in der Implementierung ..

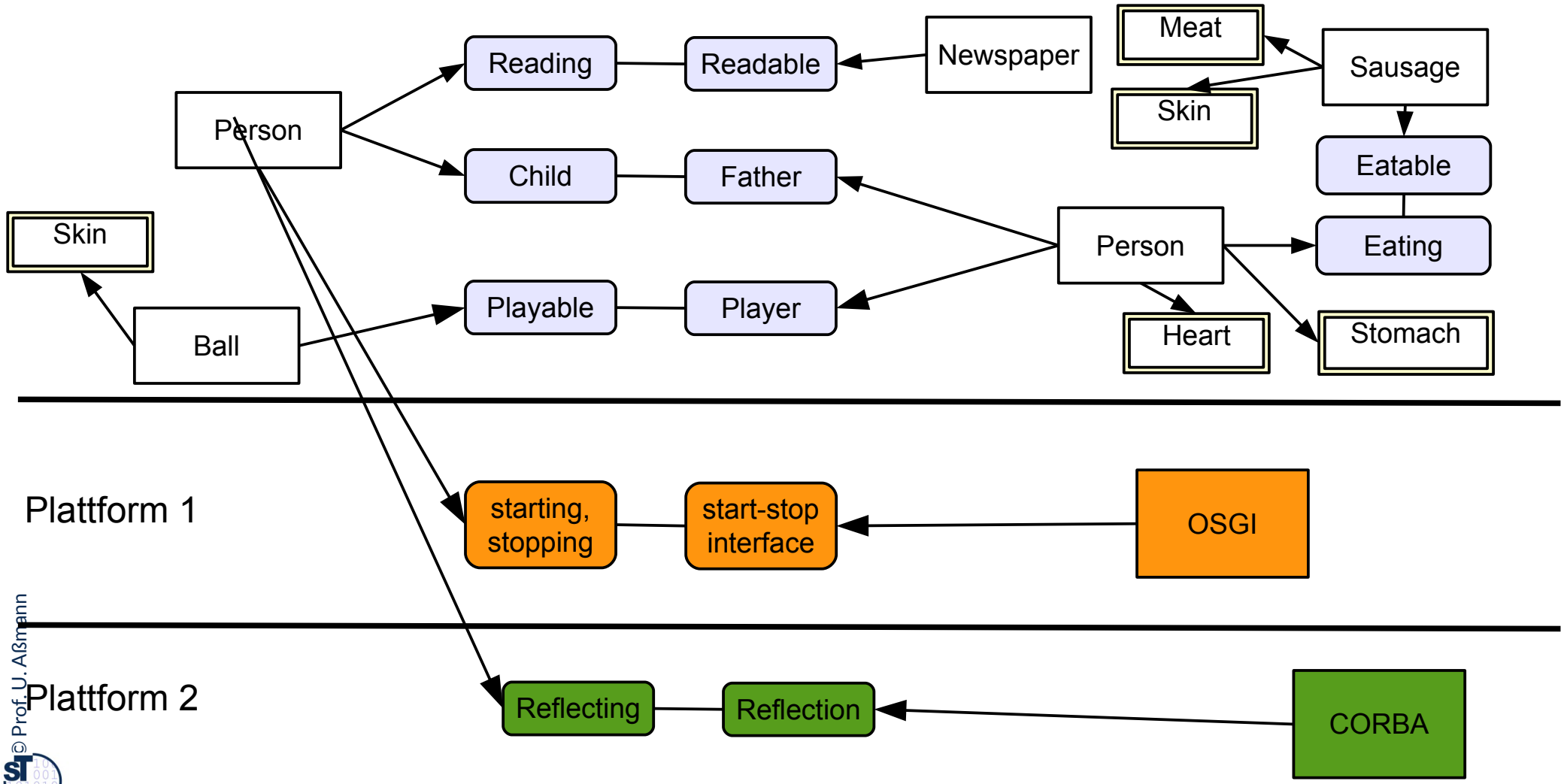
Wie bilde ich “integrates-a” ab?
a) mit einer Rollen-Programmiersprache

- Kollaborationen/Konnektoren und die “integrates”-Relation können verschieden auf eine Programmiersprache abgebildet werden
  1) Durch Rollensprachen wie ObjectTeams; dann liegt die Abbildung im Übersetzer



b) Wie bilde ich “integrates” durch Delegation ab?

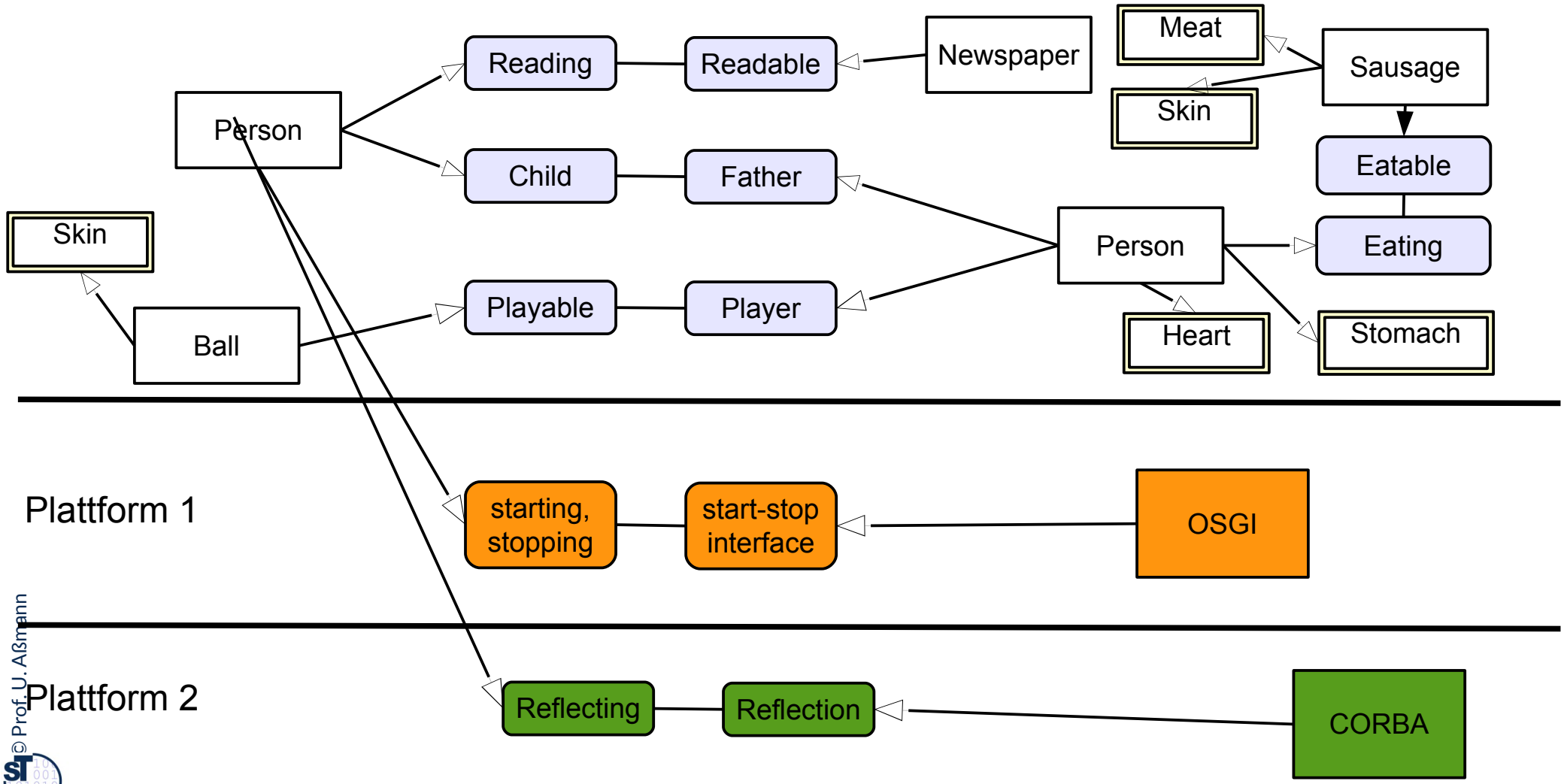
- Ersetze alle “integrates”, “plays”, “mandatory-part”, etc. durch Delegationen
- Einfach, allerdings splittert man alle logischen komplexen Objekte in unzählige Implementierungsobjekte auf (siehe Vorlesung “Design Patterns and Frameworks”)
- Statische Komposition der Initial- und Terminal-Botschaften nötig



Statische Komposition der Initial- und Terminal-Botschaften

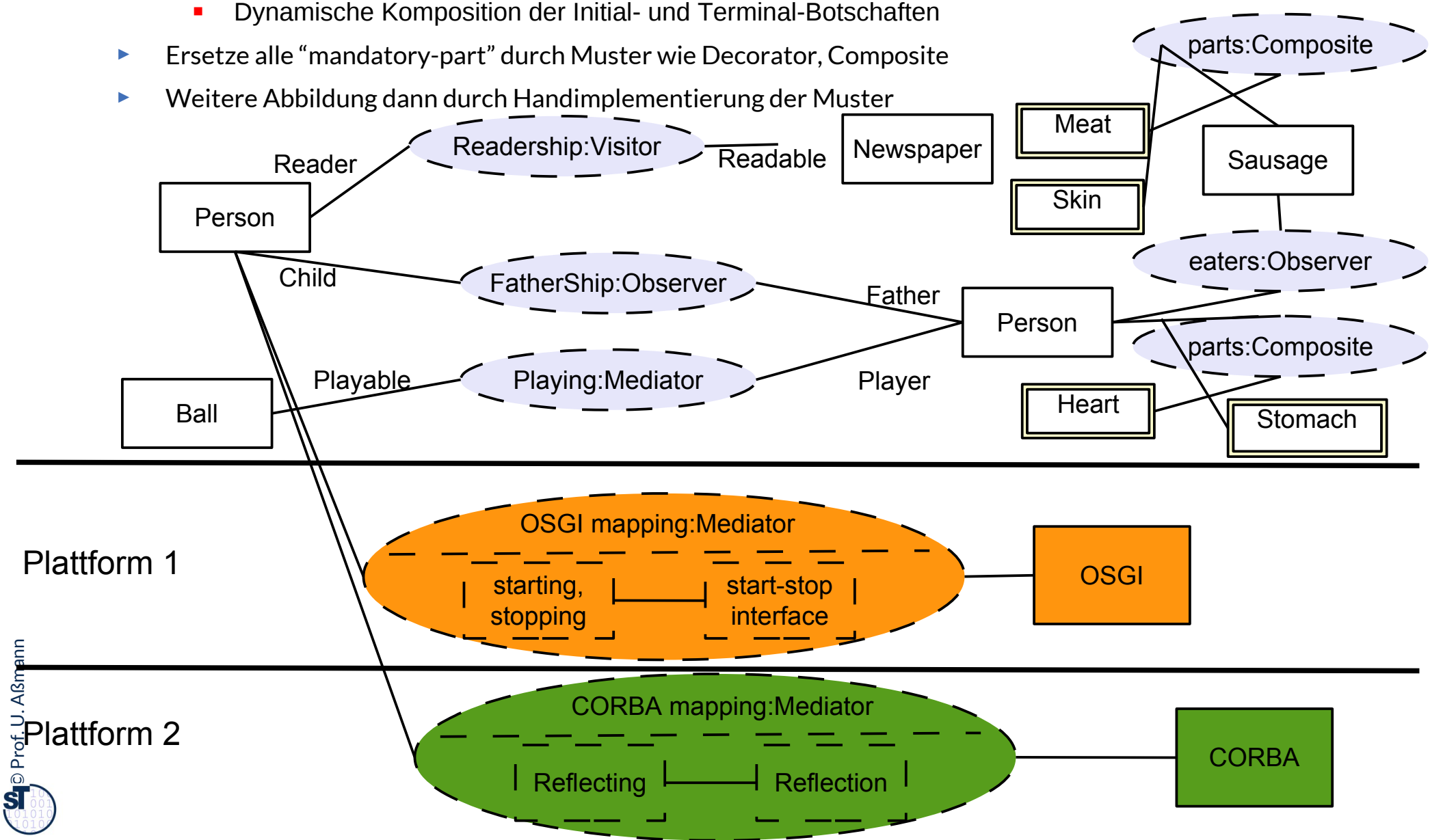
c) Wie bilde ich “integrates” durch Vererbung ab?

- Ersetze alle “integrates”, “plays”, “mandatory-part”, etc. durch Vererbung
- Einfach, allerdings braucht man Mehrfachvererbung oder “mixin inheritance”
- Statische Komposition der Initial- und Terminal-Botschaften nötig



d) Wie bilde ich “integrates” durch Implementierungsmuster ab?

- Ersetze alle “integrates”, “plays”, etc. durch Muster wie Observer, Visitor
  - Dynamische Komposition der Initial- und Terminal-Botschaften
- Ersetze alle “mandatory-part” durch Muster wie Decorator, Composite
- Weitere Abbildung dann durch Handimplementierung der Muster

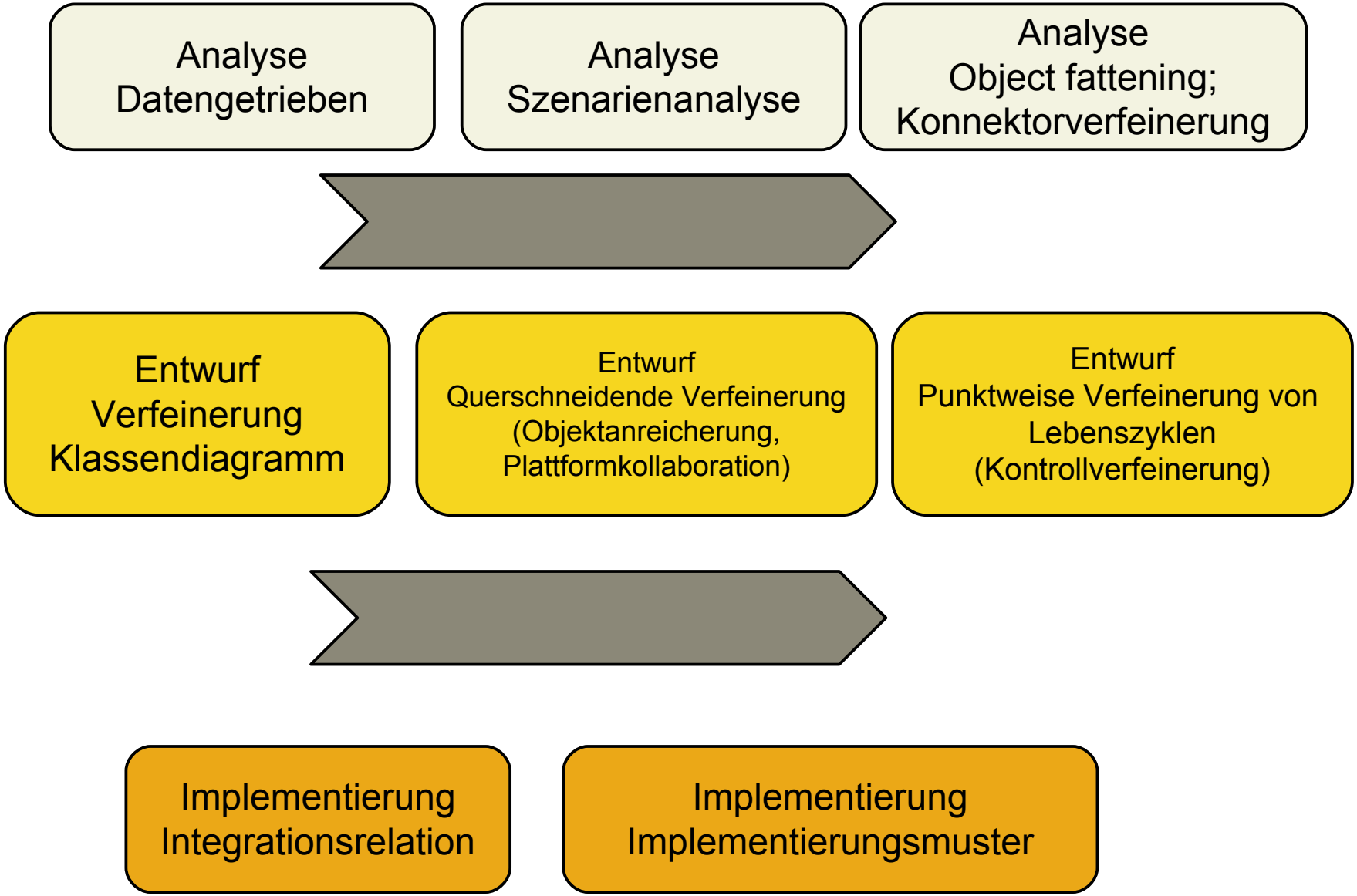


Dynamisch Komposition der Initial- und Terminal-Botschaften

d) Wie bilde ich “integrates” durch Transformation ab?

- Ersetze alle “integrates”, “plays”, etc. durch *Transformationsregeln*
- Führt auf *Modellgetriebene Architektur (model-driven architecture, MDA)*
- Weiter in der Softwaretechnologie-II

43.5 Gesamtbild der Verfeinerung



The End

Anhang A: Nebenbemerkung

- Integration von Unterobjekten in Kernobjekte kann *zu verschiedenen Zeiten* erfolgen
  - Zur Entwurfszeit
  - Zur Bindezeit
  - Zur Allokationszeit eines Objekts
  - Zur Laufzeit
  - Zur Zeit der Software-Pflege und -Migration

Anhang: Rollen in der Literatur

- Rollenorientiertes Datenmodell (Bachmann 77)
- Entity-Relationship-Modell (ER model, Chen 76): Hier bilden die Enden einer Assoziation eine Rolle. Vorbild für UML-Klassendiagramme
  - Kurs “Softwarewerkzeuge (SEW)” im WS
- Entwurfsmuster (Riehle 98)
  - Kurs “Design patterns and frameworks (DPF)” im WS
- Produktlinien-Engineering (Smaragdakis,Batory 02)
- Kollaborationen in Architektursprachen (Garlan, Shaw 95)
  - Kurs “Component-based Software Engineering (CBSE)” im SS
- Objektorientierte Modelierung mit der OORAM Methode (Reenskaug 95)

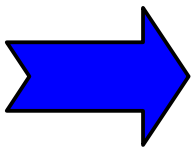
Anhang B Volles Verfeinerungsbeispiel für Objektanreicherung mit allen Arten von Unterobjekten

.. Verfeinerung durch Integration von Unterobjekten..

(optional)

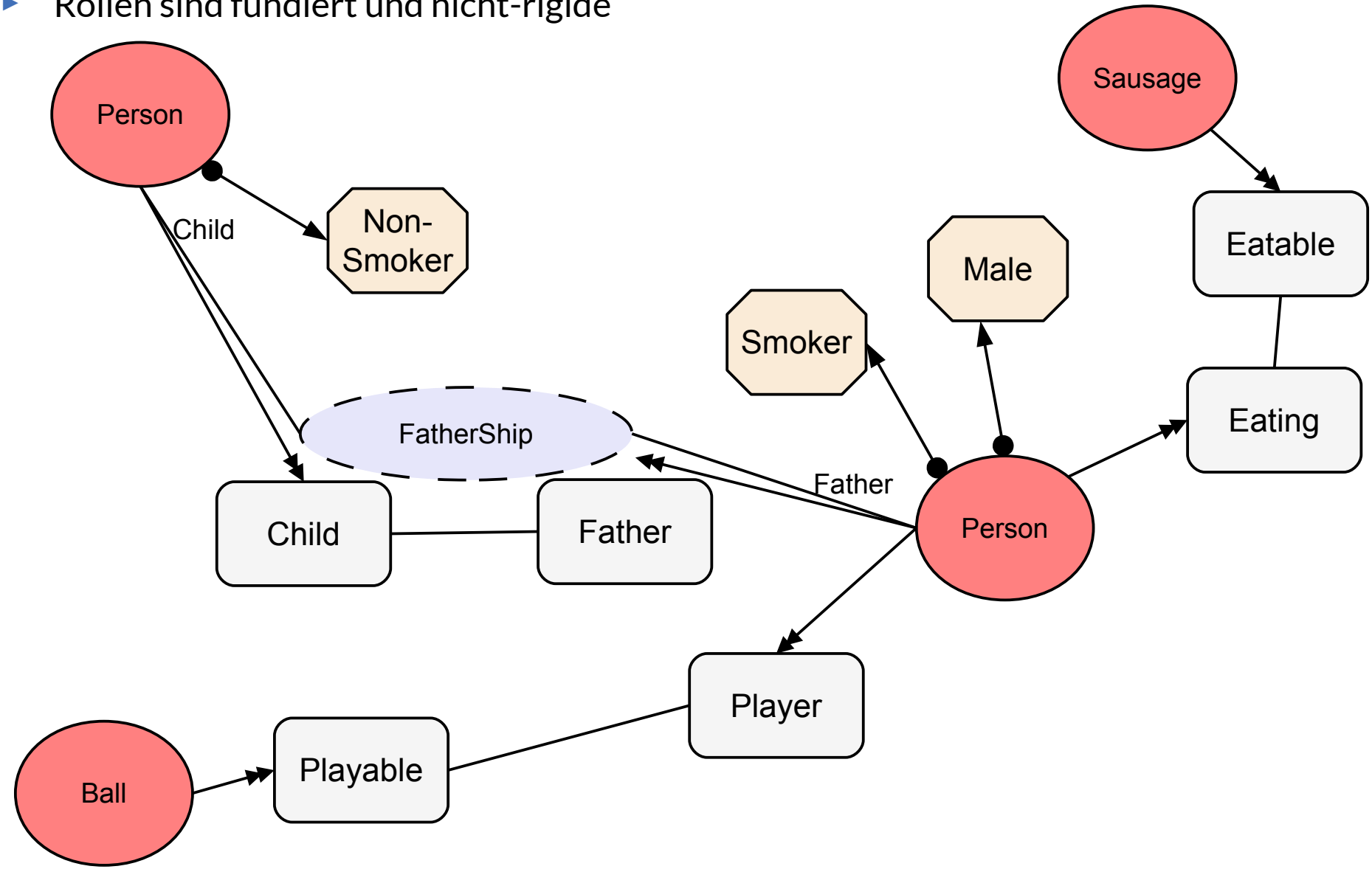
Erweitertes Konzept der Objektanreicherung

- Weitere Unterobjekte können integriert werden
  - *Phasen ergänzen (Phasen-Verfeinerung)*
  - *Facetten ergänzen (Facetten-Verfeinerung)*
  - Teile ergänzen (Teile-Verfeinerung)
  - Rollen ergänzen (Kollaboration-Verfeinerung),
  - Rollen ergänzen (Kollaboration-Verfeinerung), die Beziehungen klären zu
    - Plattformen (middleware, Sprachen, Komponenten-services)
    - Komponentenmodellen (durch Adaptergenerierung)
- Ziel: Entwurfsobjekte, Implementierungsobjekte

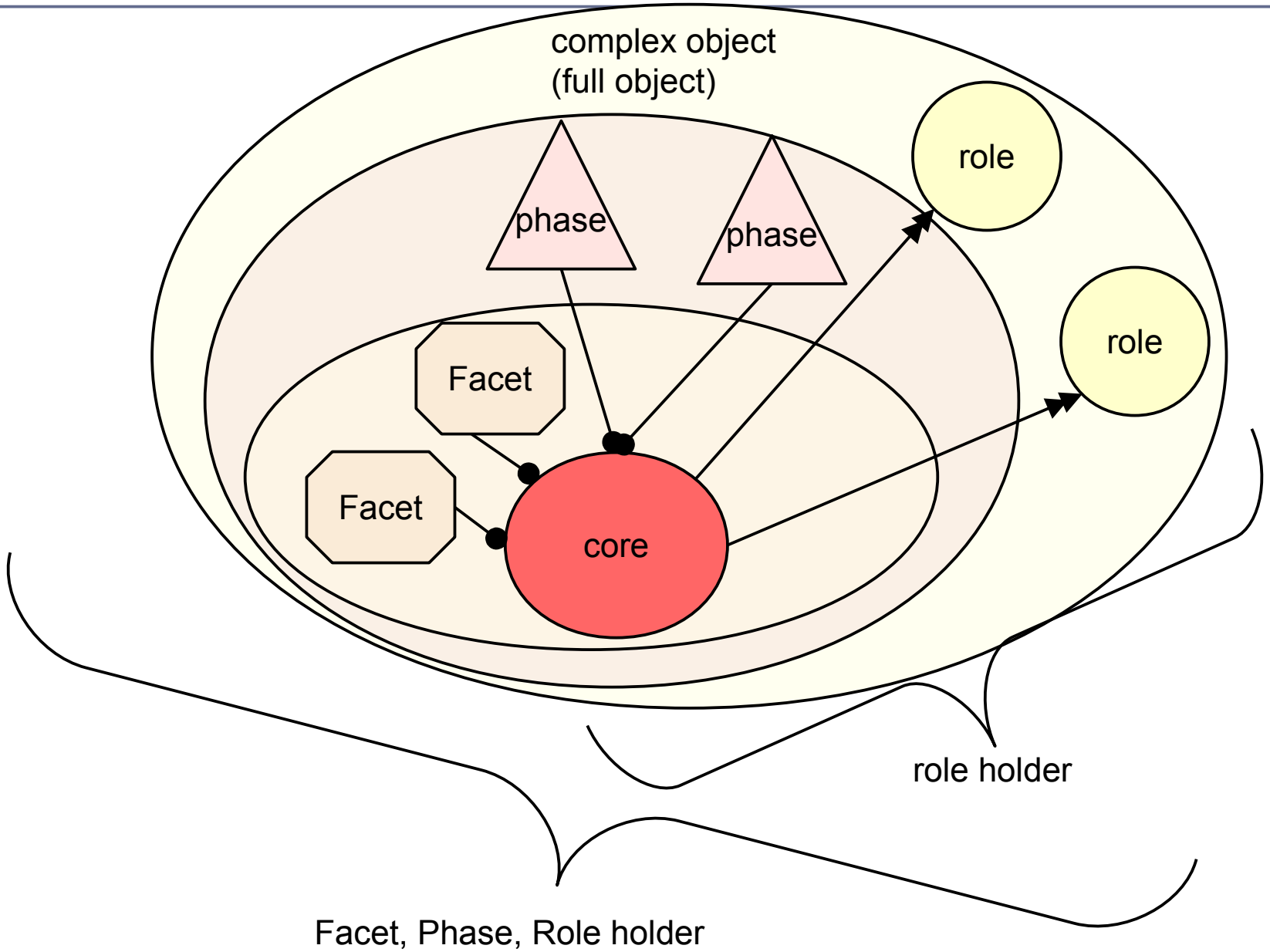


Facetten im Vergleich zu Rollen (Wdh.)

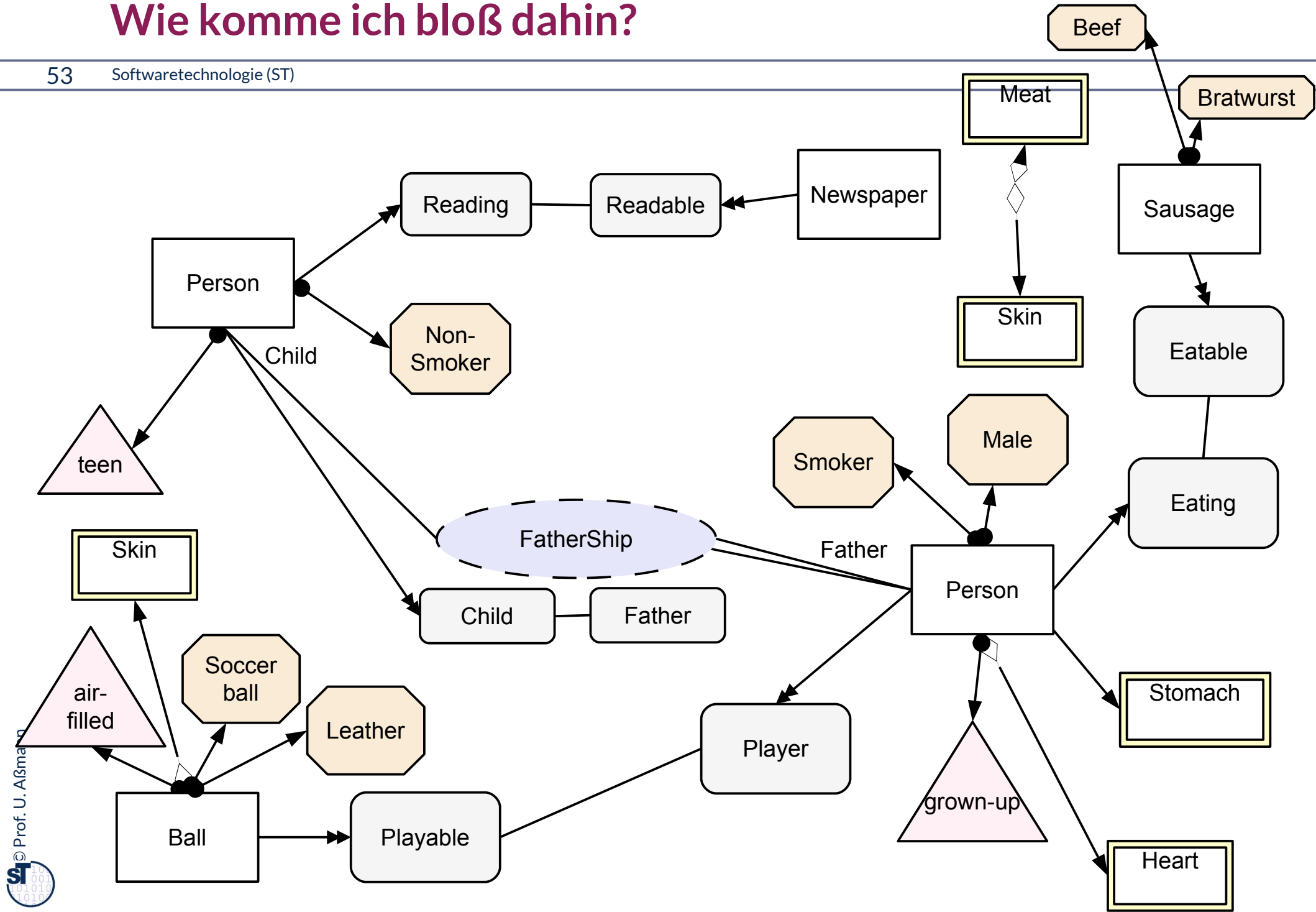
- Facetten sind nicht-fundiert und rigide (natürlich)
- Rollen sind fundiert und nicht-rigide



Komplexe Objekte

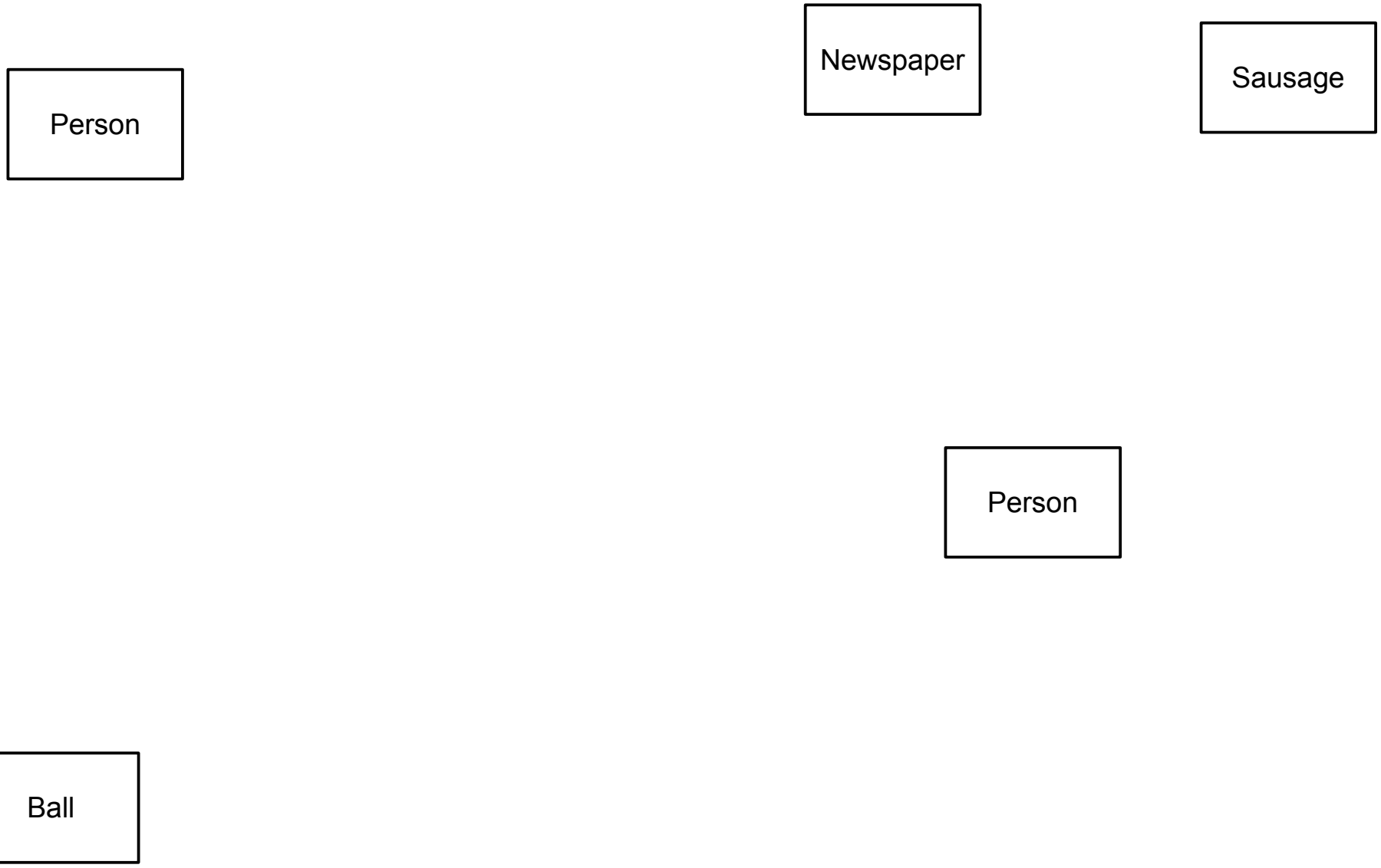


Personen-Analysemodell – Wie komme ich bloß dahin?



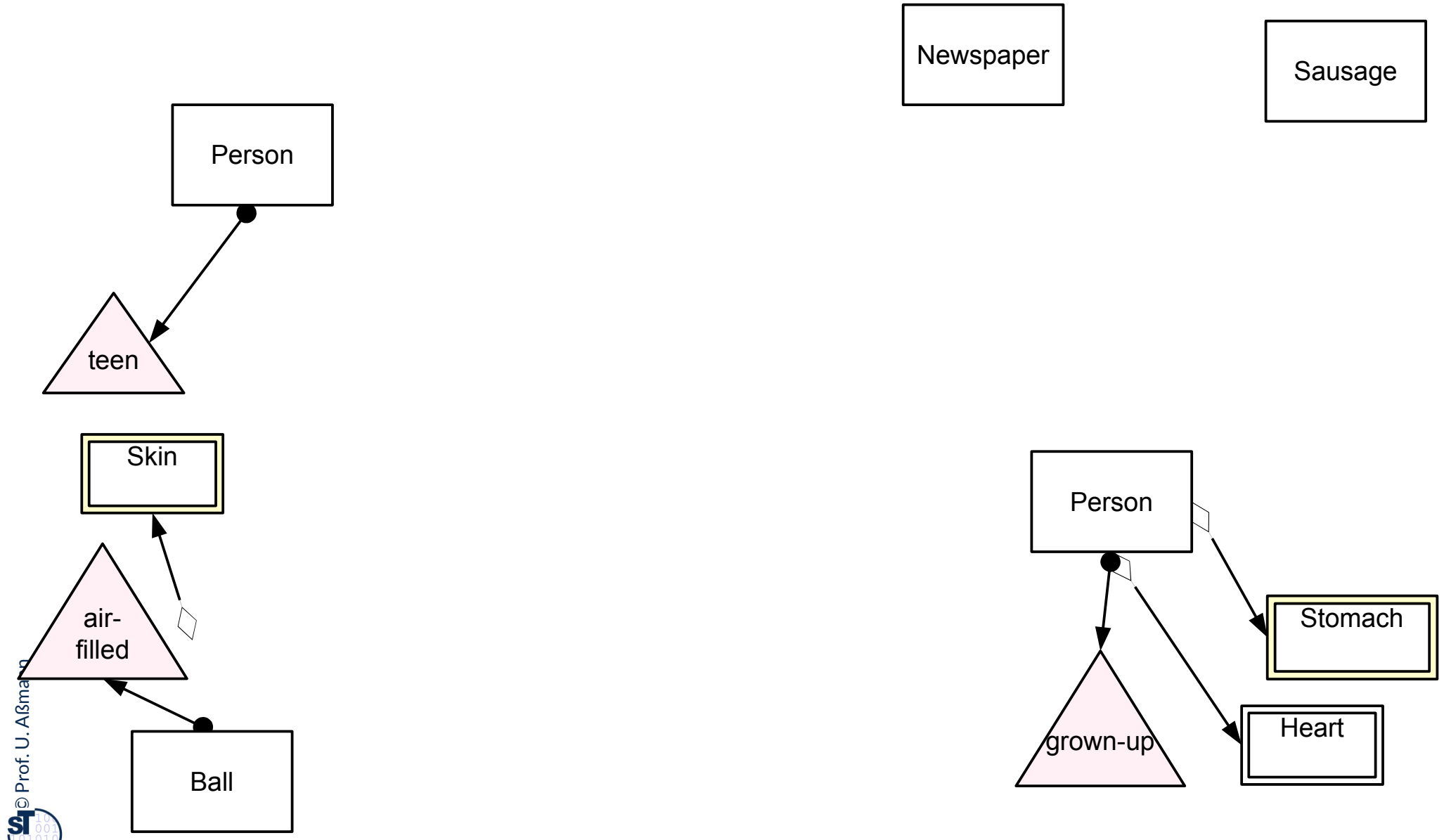
Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Rohzustand: Identifikation der natürlichen Typen



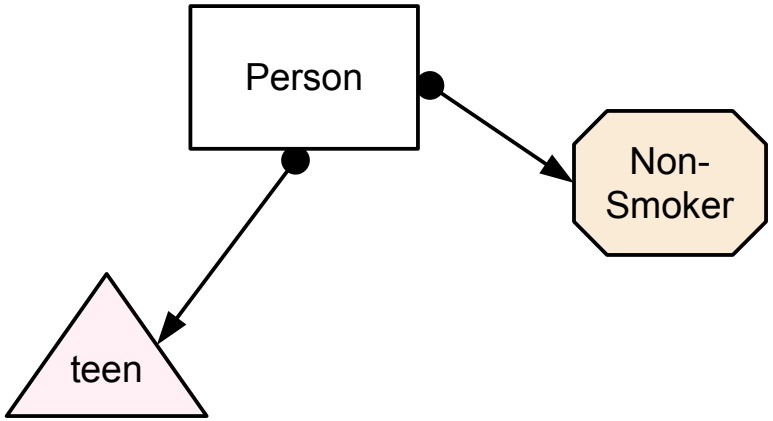
Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

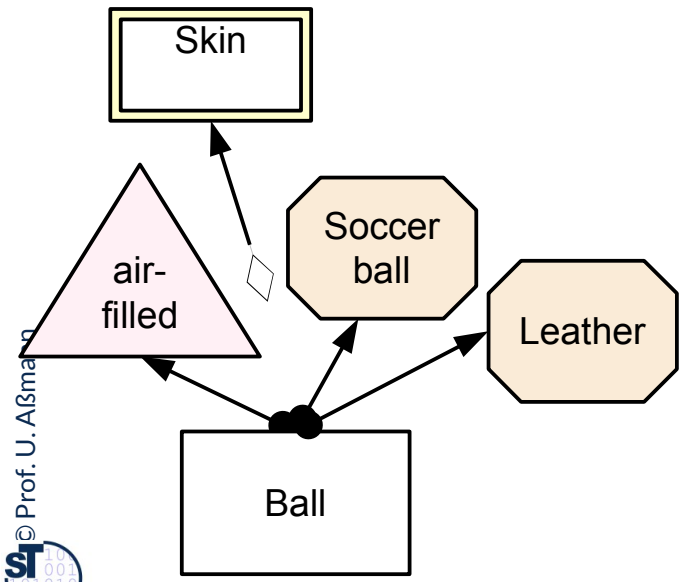
- Schritt 1: Teile-Verfeinerung, Phasen-Verfeinerung

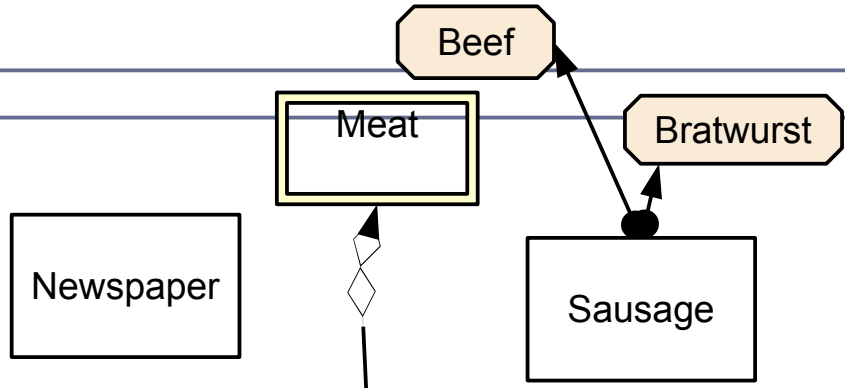


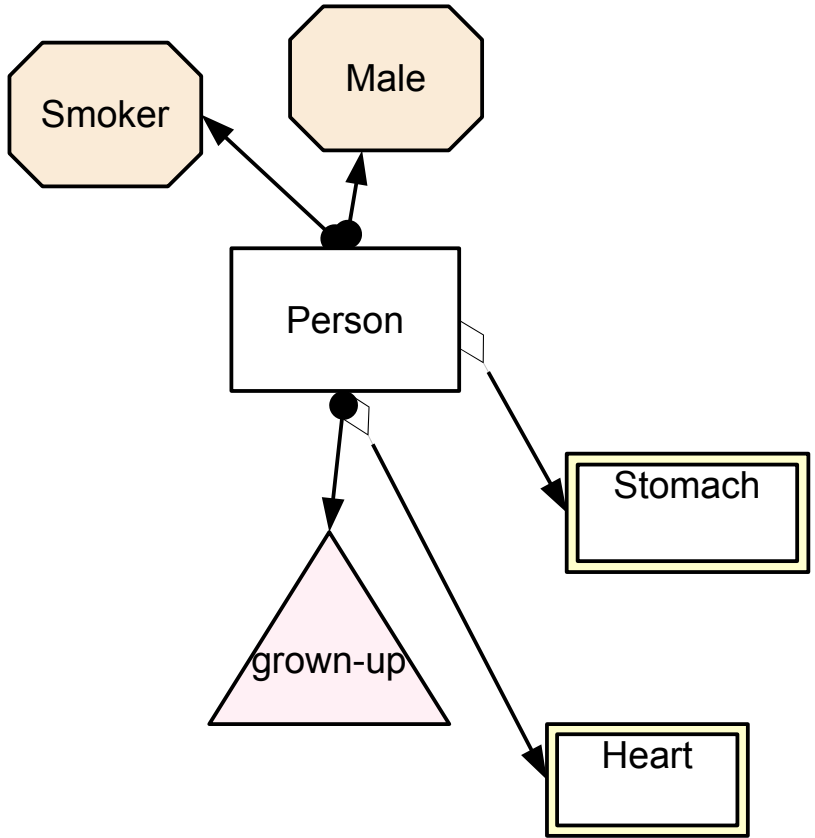
Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 2: Facetten-Verfeinerung



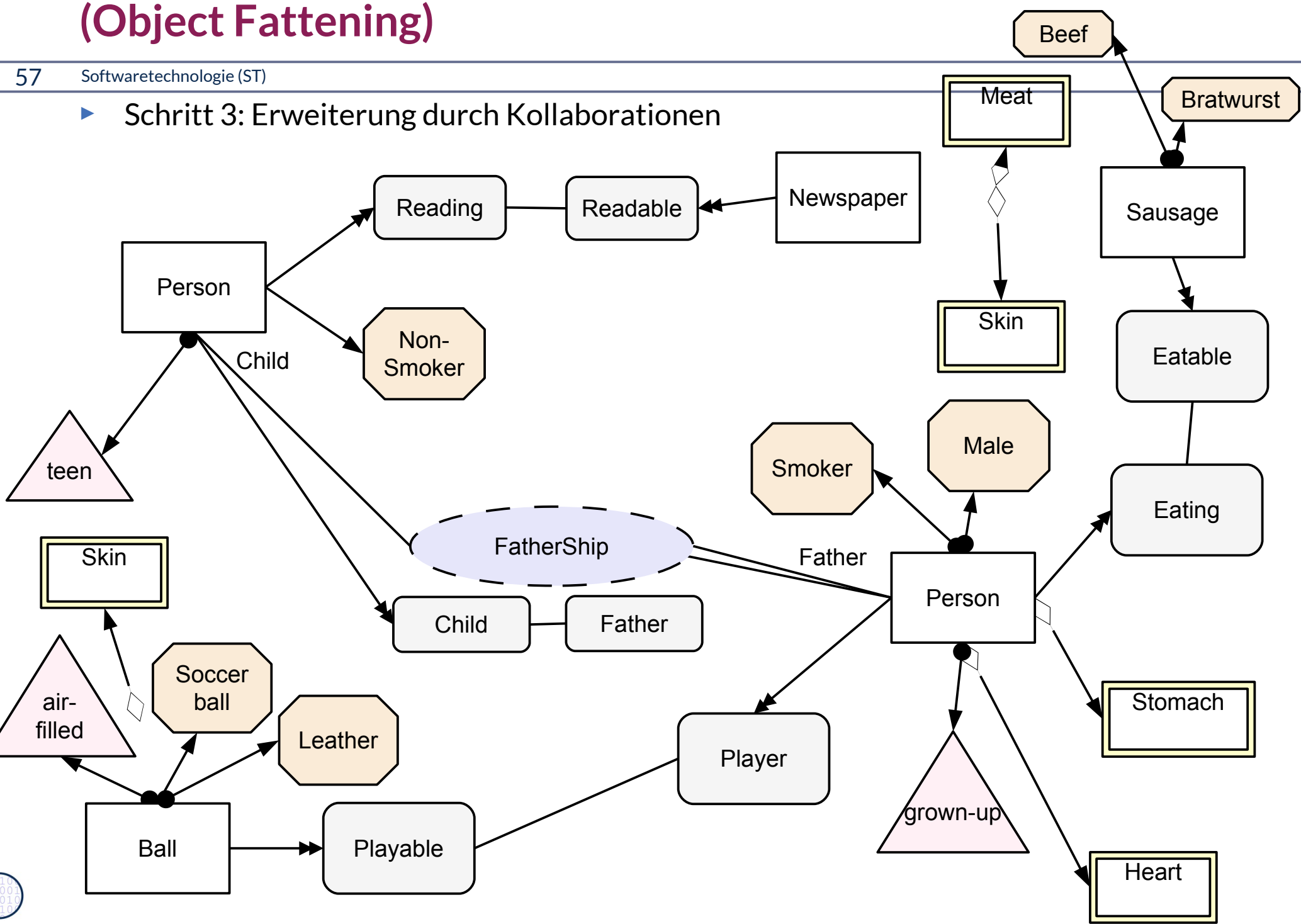






Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 3: Erweiterung durch Kollaborationen



Komposition von Kollaborationen im Entwurf

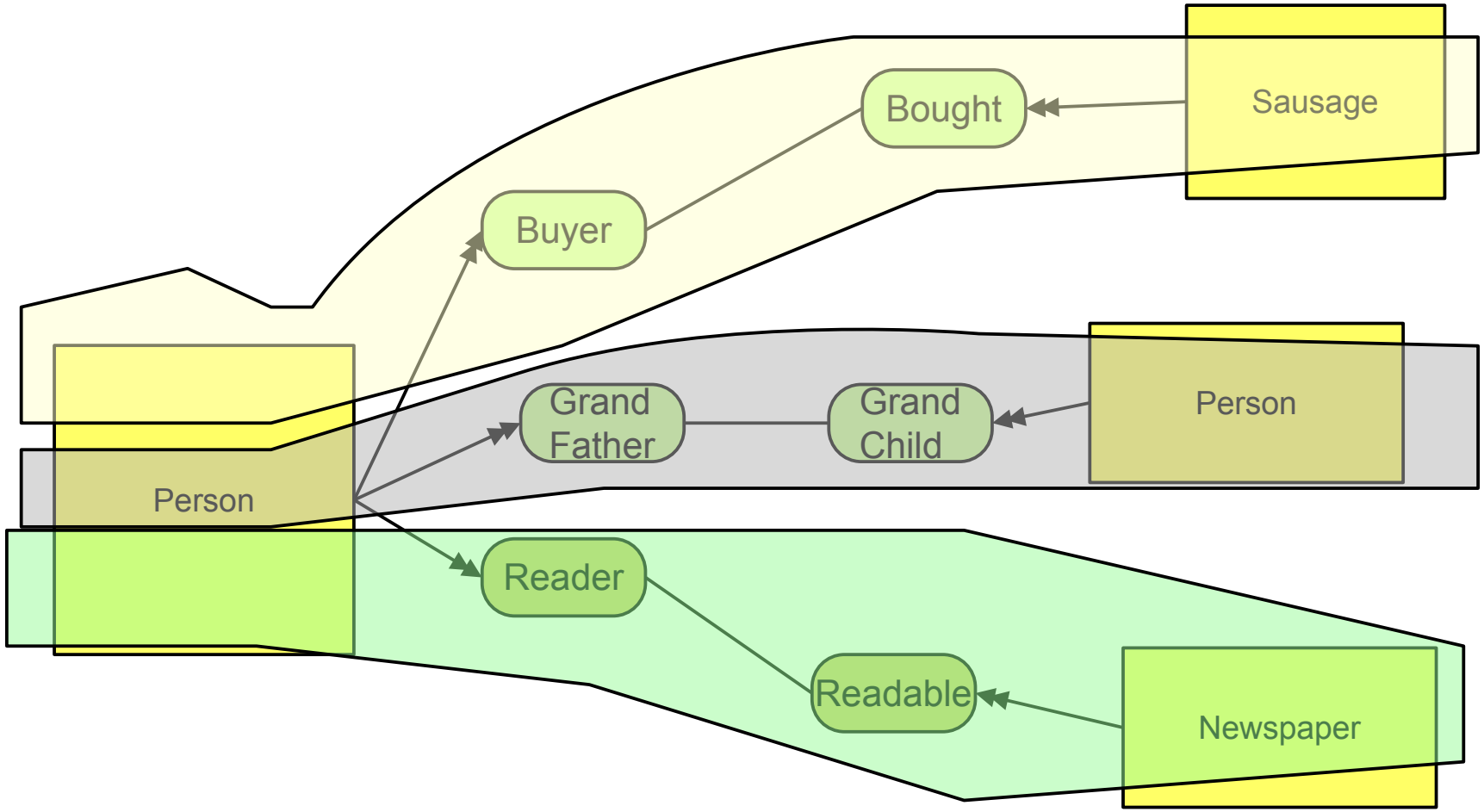
Szenarienanalyse wird nicht nur für Kontextmodell und TLA eingesetzt, sondern auch im Entwurf

Querschneidende Verfeinerung besteht aus zwei Schritten:

- Szenarienanalyse zur Erstellung von Kollaborationen
- Superimposition der Kollaborationen auf das bisherige Entwurfsmodell

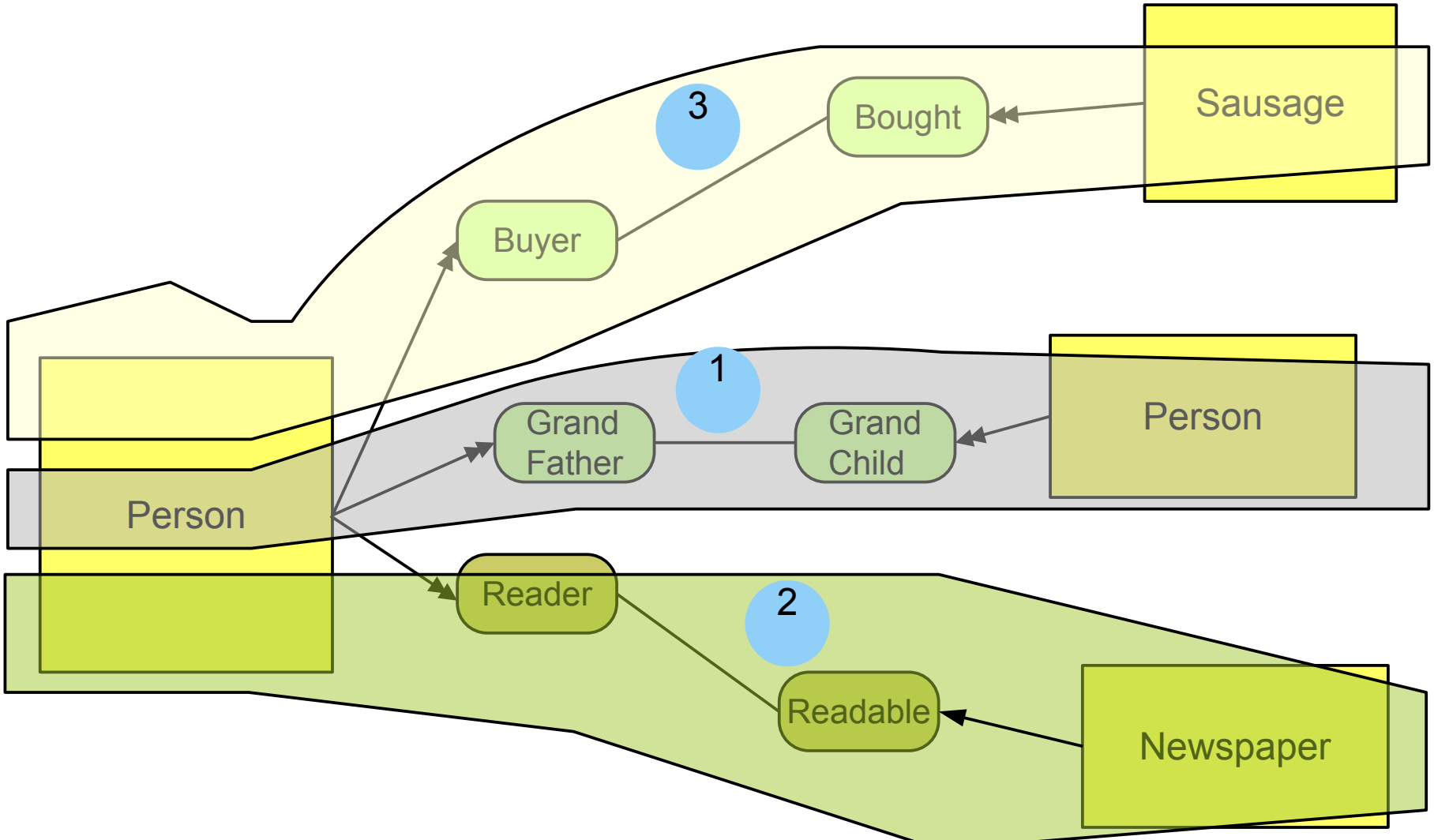
Kollaborationen als Schnitte durch die Anwendung

- Kollaborationen bilden *Schnitte (slices)* durch die Anwendung
- Mehrere Kollaborationen können auf die Anwendung superimponiert werden



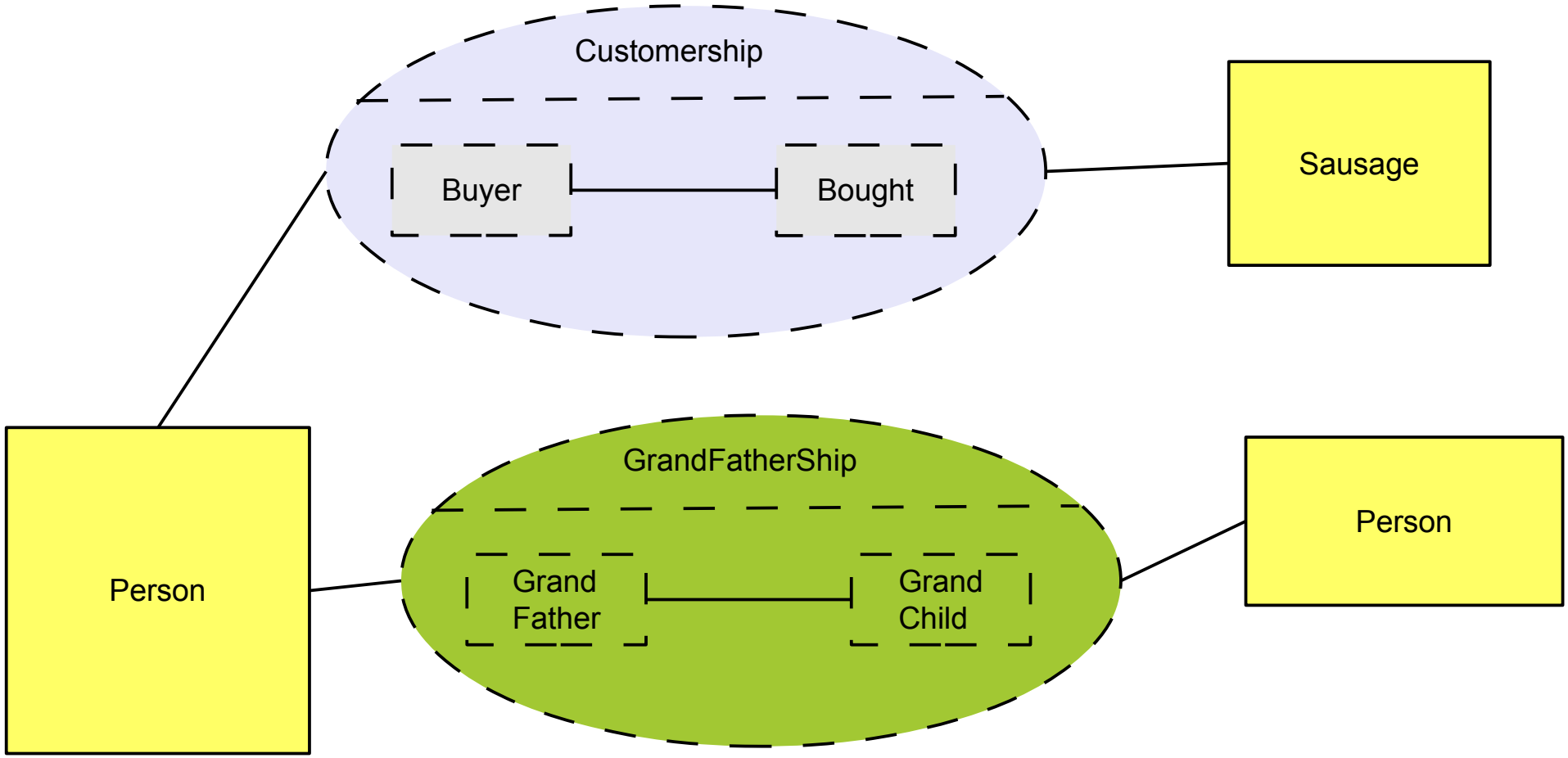
Querschneidende Erweiterung von Anwendungen mit Kollaborationen

- Analyse- und Entwurfsmodelle können sukzessive durch Kollaborationen **querschneidend** erweitert werden

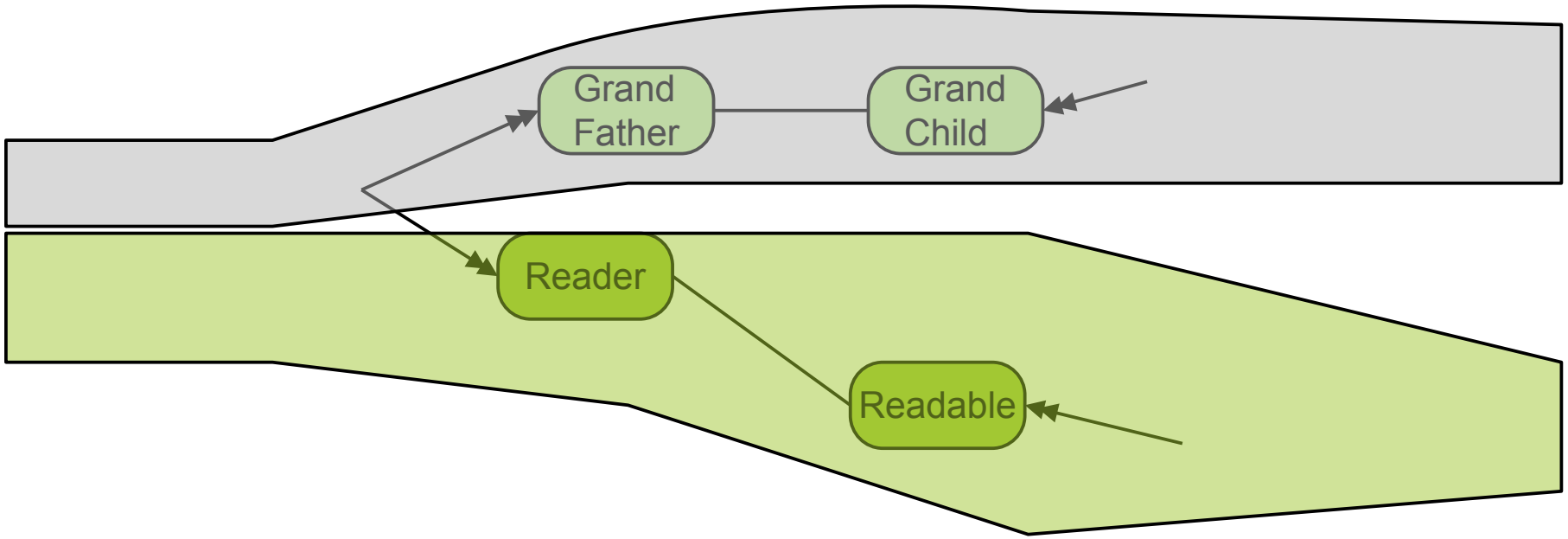


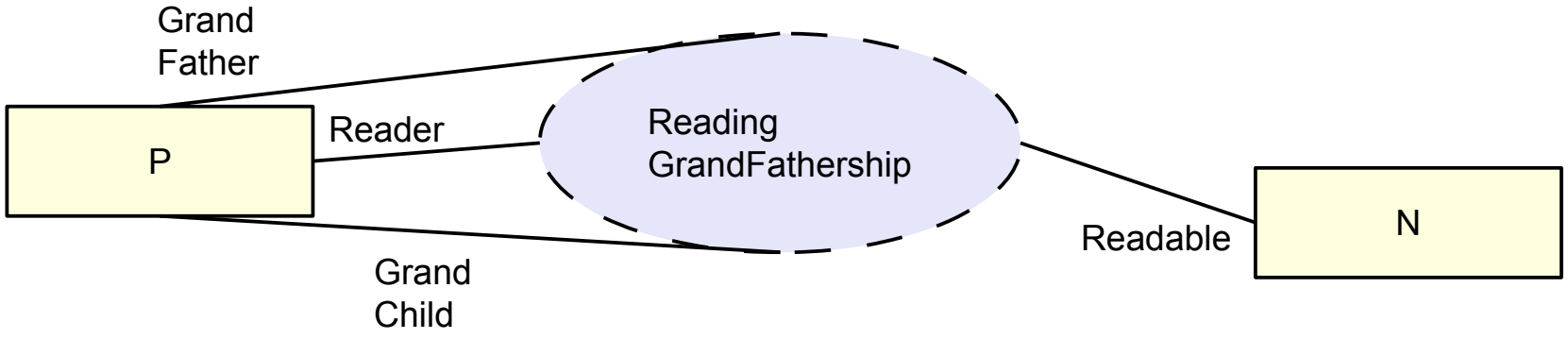
Verfeinerung mit Kollaboration-Superimposition in UML

- Das Überlagern von Kollaborationen und Konnektoren nennt man Superimposition **(Collaboration, connector superimposition)**
- Alternative Notation in UML: Kollaborationen *mit Abteilen*

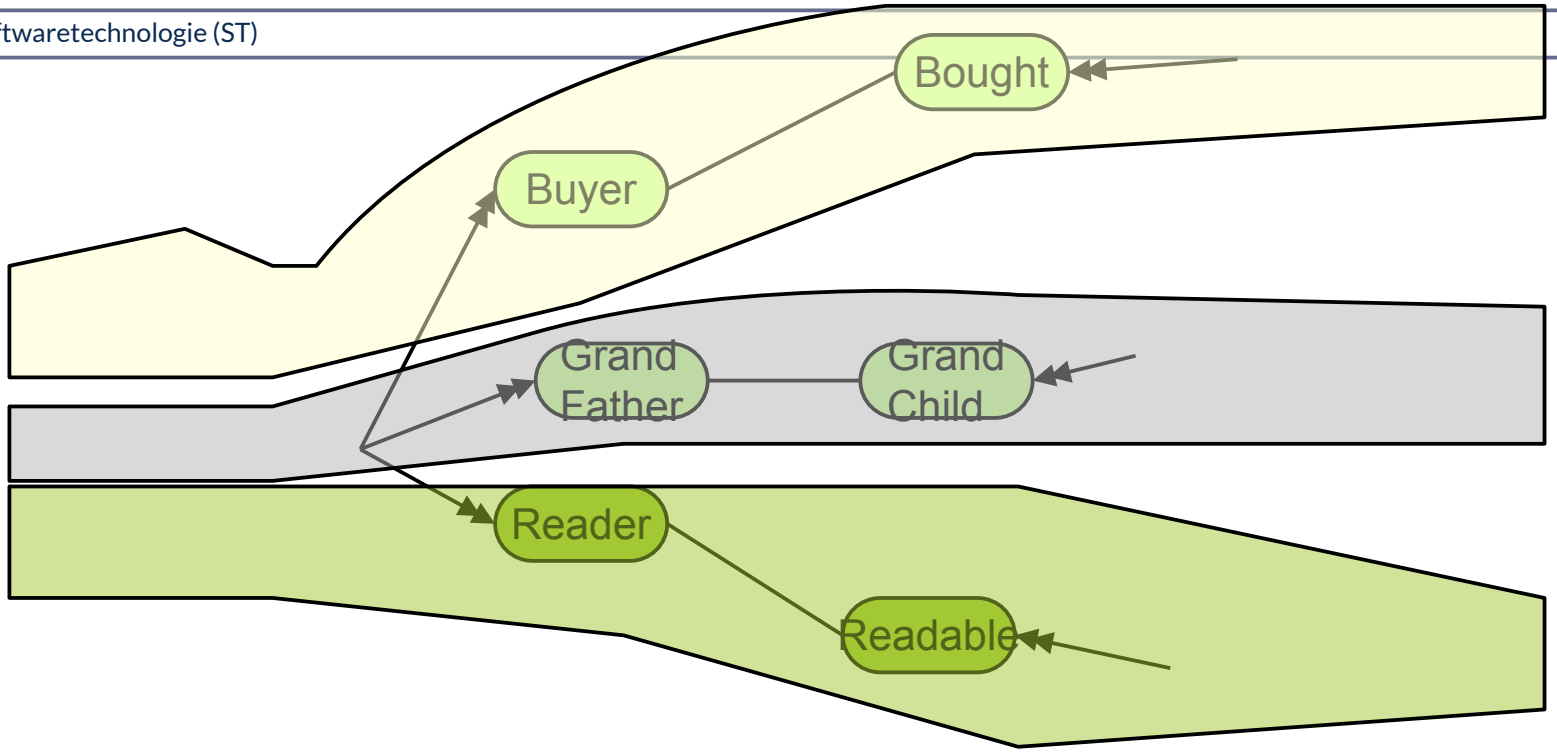


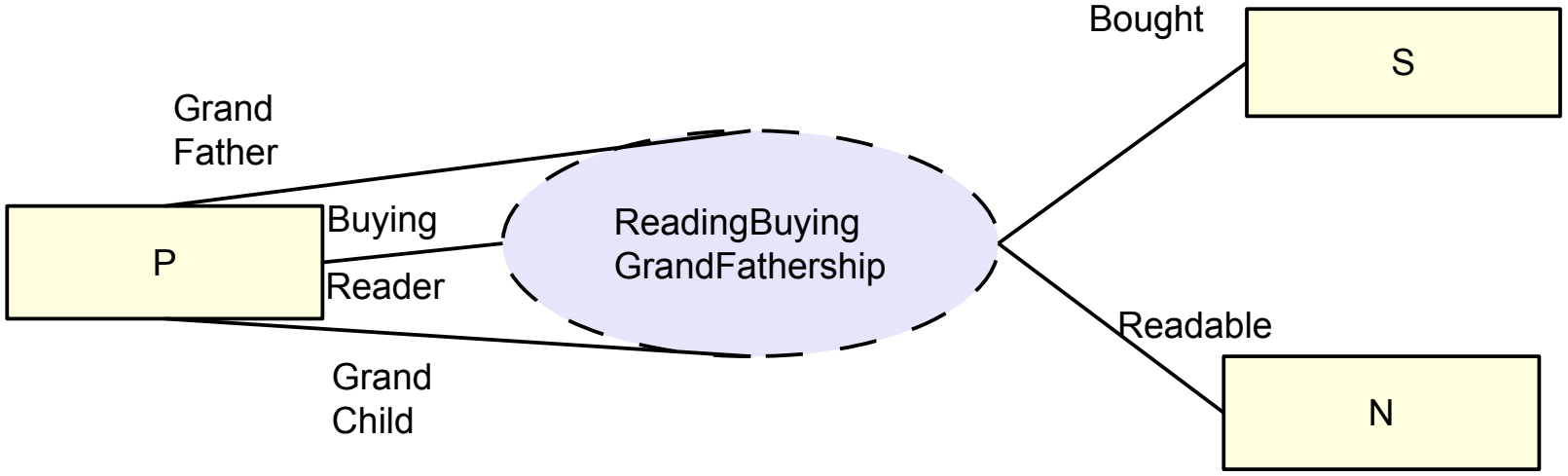
Verschmelzen von Kollaborationen: Newspaper-Reading GrandpaShip





Verschmelzen von Kollaborationen:
Newspaper-Reading Buying GrandpaShip





Kollaborationsbasierte Verfeinerung

- Kollaborationsbasierte (querschneidende) Verfeinerung bedeutet, Schritt für Schritt neue Kollaborationen in das Analyse- und das Entwurfsmodell zu integrieren,
  - d.h. neue Kollaborationen zu superimponieren
- In einer Programmiersprache wie ObjectTeams kann man das direkt umsetzen, in dem man zu einem Kern-Programm neue Teams hinzufügt
  - In Java ist es schwieriger