# 44. Querschneidende Verfeinerung mit Plattformkonnektoren

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 15-0.2, 11.07.15

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

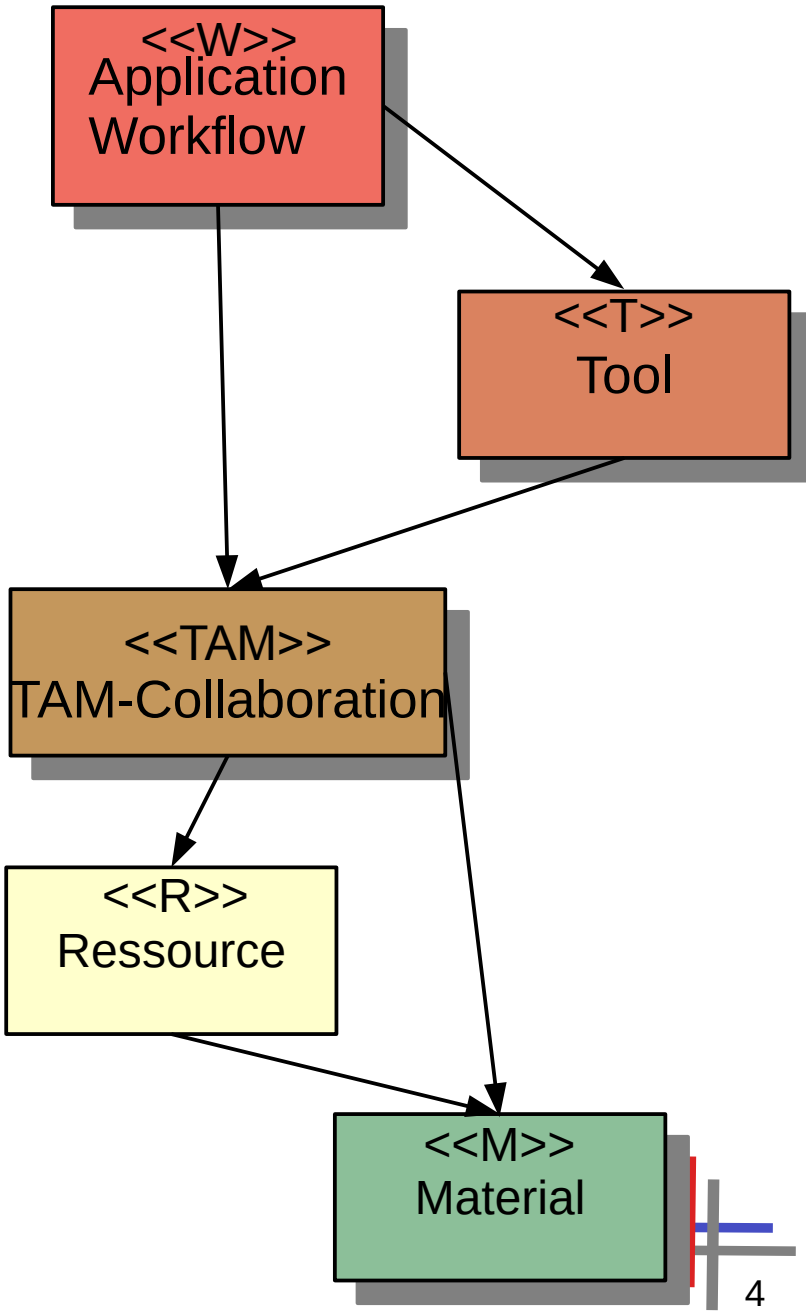
# Literatur

- Obligatorisch:
  - 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 definiert:

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



# 44.1 Identifikation von Tools, Materials, zur Einordnung von Klassen in die Schichten

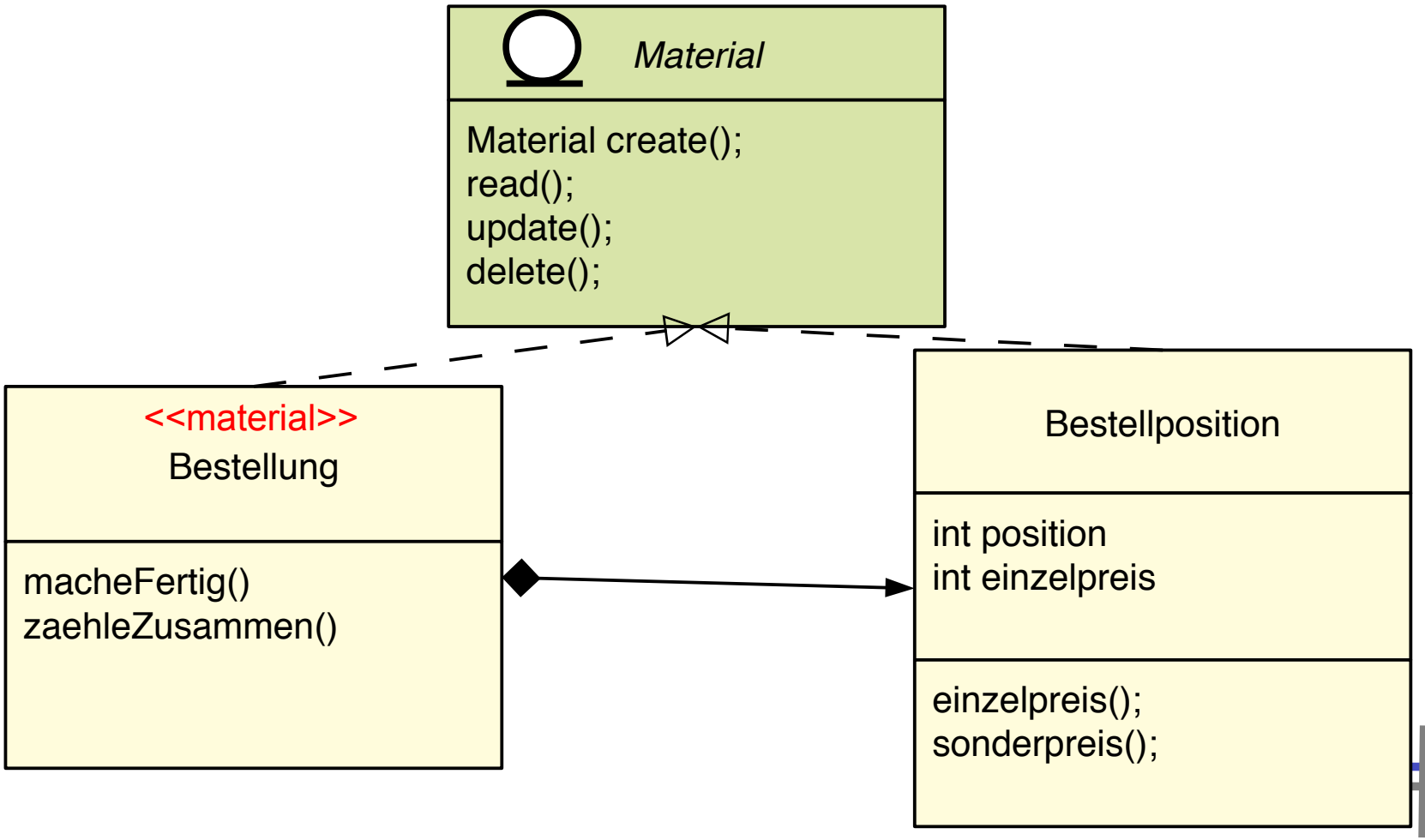
Was wird interaktiv (asynchron) aufgerufen?

Was ist passiv?

Was muss belegt werden?

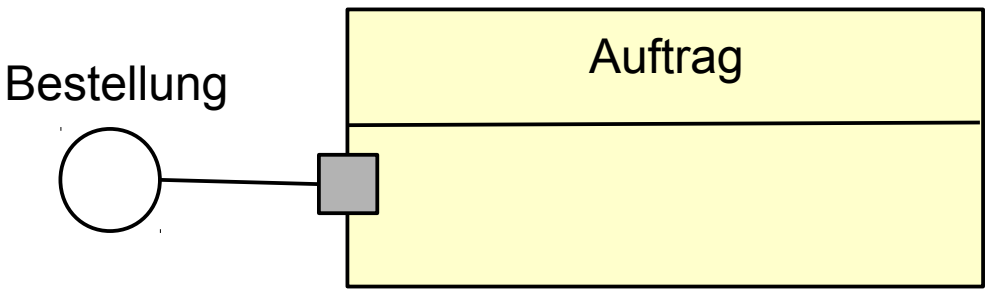
# Material-Klassen und -Schnittstellen

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



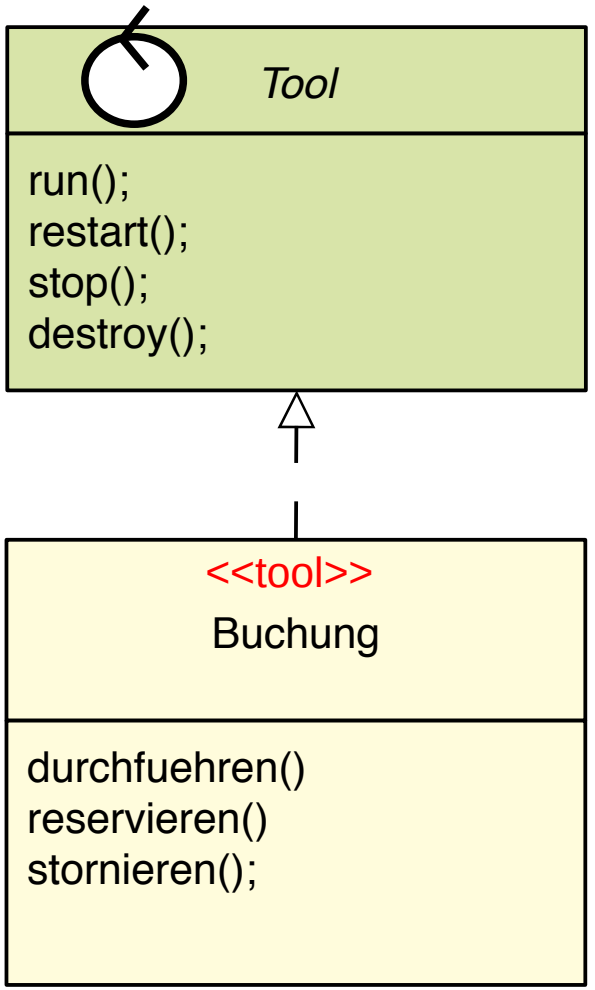
# Material-Klassen und -Schnittstellen

- Materialien können in Ports auftauchen



# Tool-Klassen und -Schnittstellen

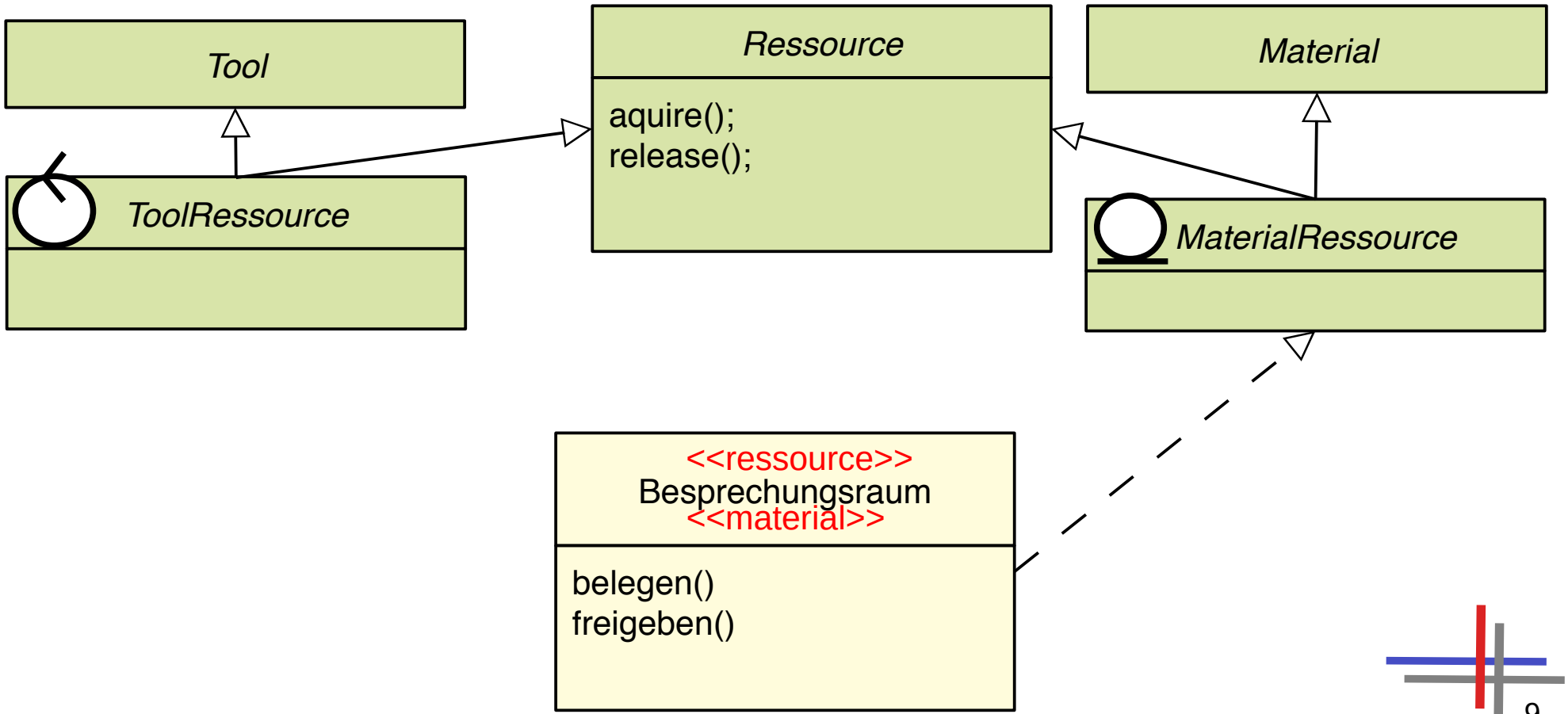
- Toolobjekte sind I.d.R. aktiv, besitzen eigenen Steuerfluss (thread, process)




[Züllighoven]

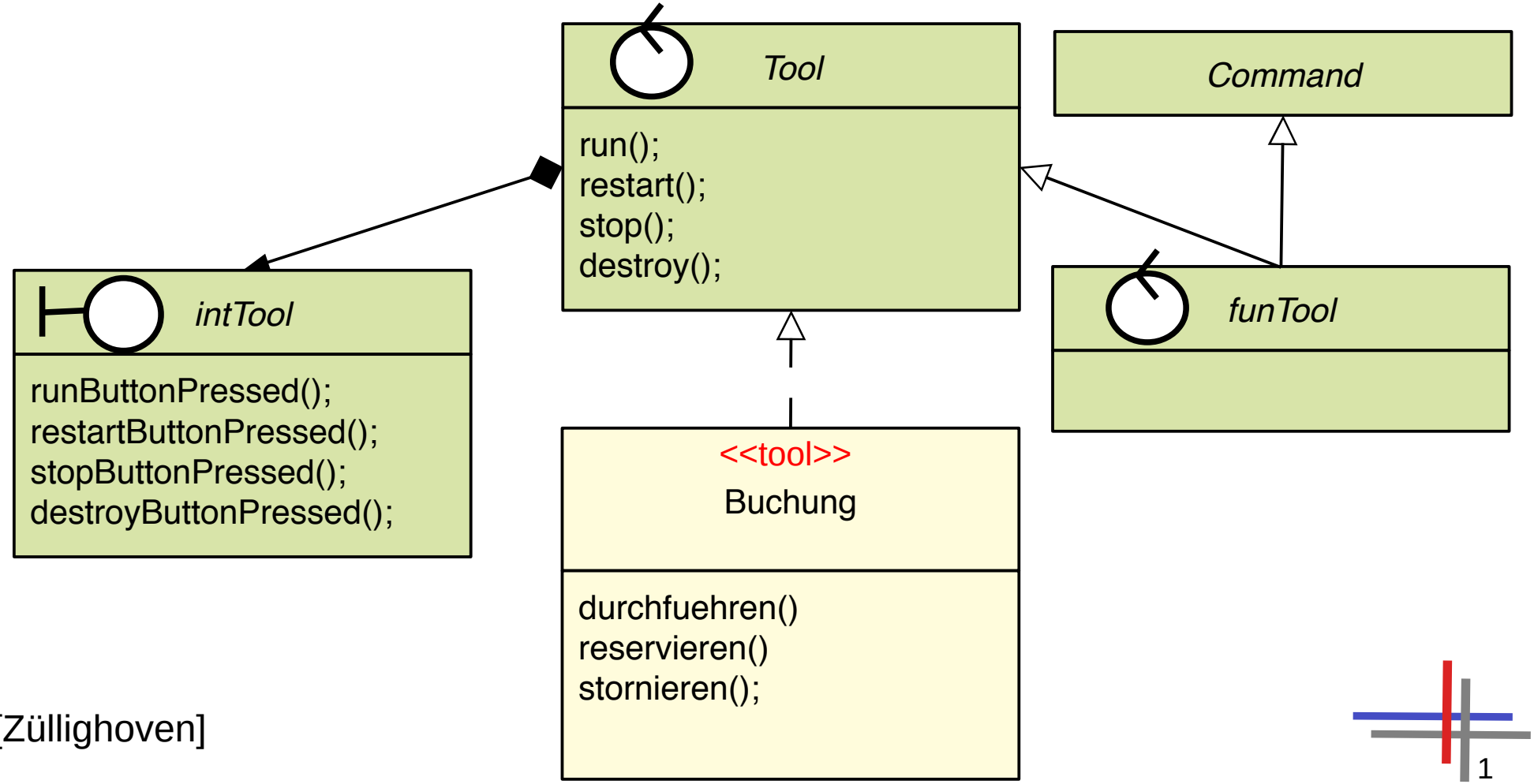
# Ressourcen-Klassen und -Schnittstellen

- **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

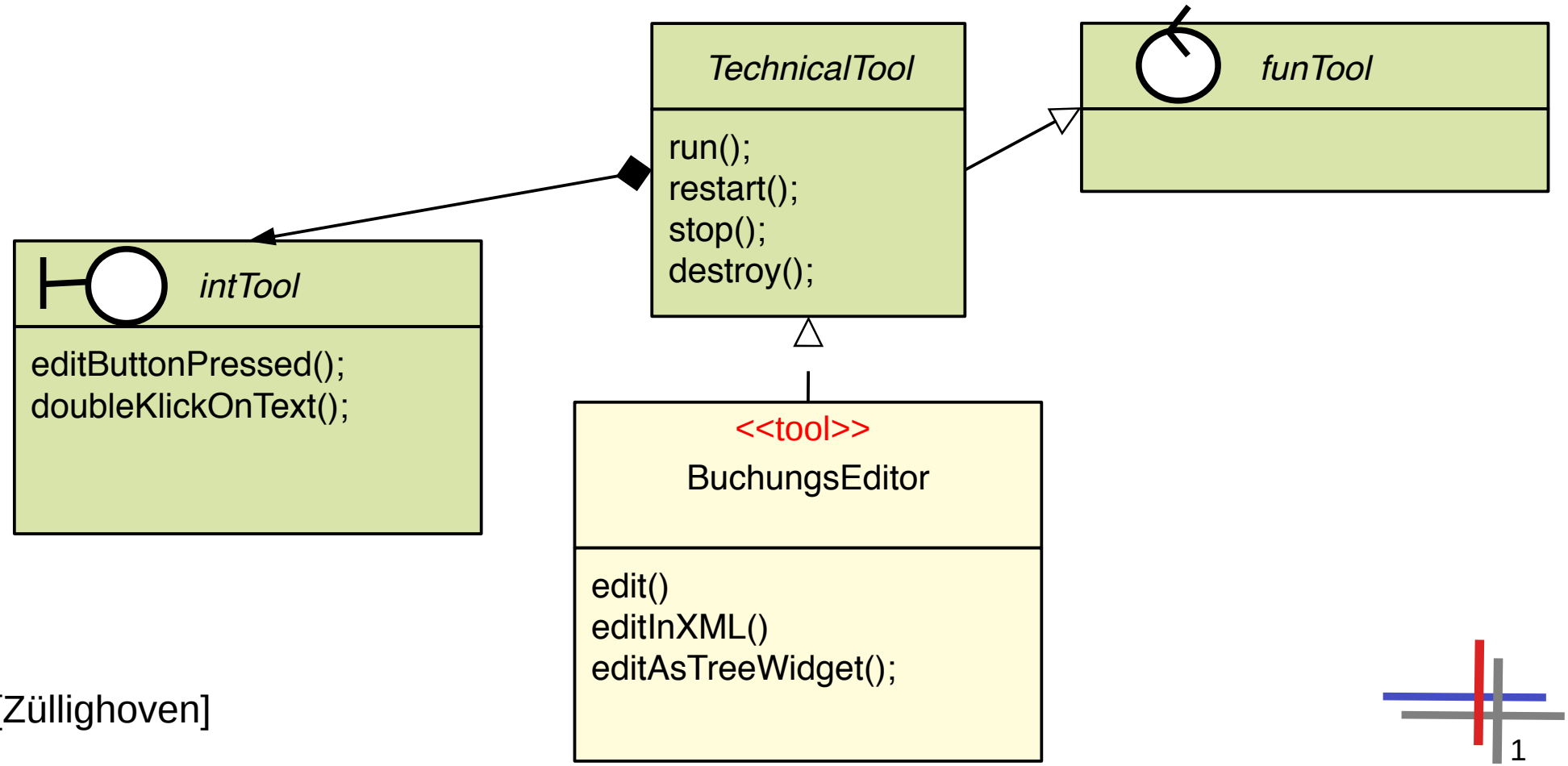
- Toolobjekte haben einen interaktiven Teil (intTool, boundary) und einen ausführenden, funktionalen Teil (funTool, control), der aus dem Command-Pattern abgeleitet ist
- **Interaktive Tools** stecken hinter den Menüeinträgen




[Züllighoven]

# Technische Tool-Klassen und -Schnittstellen

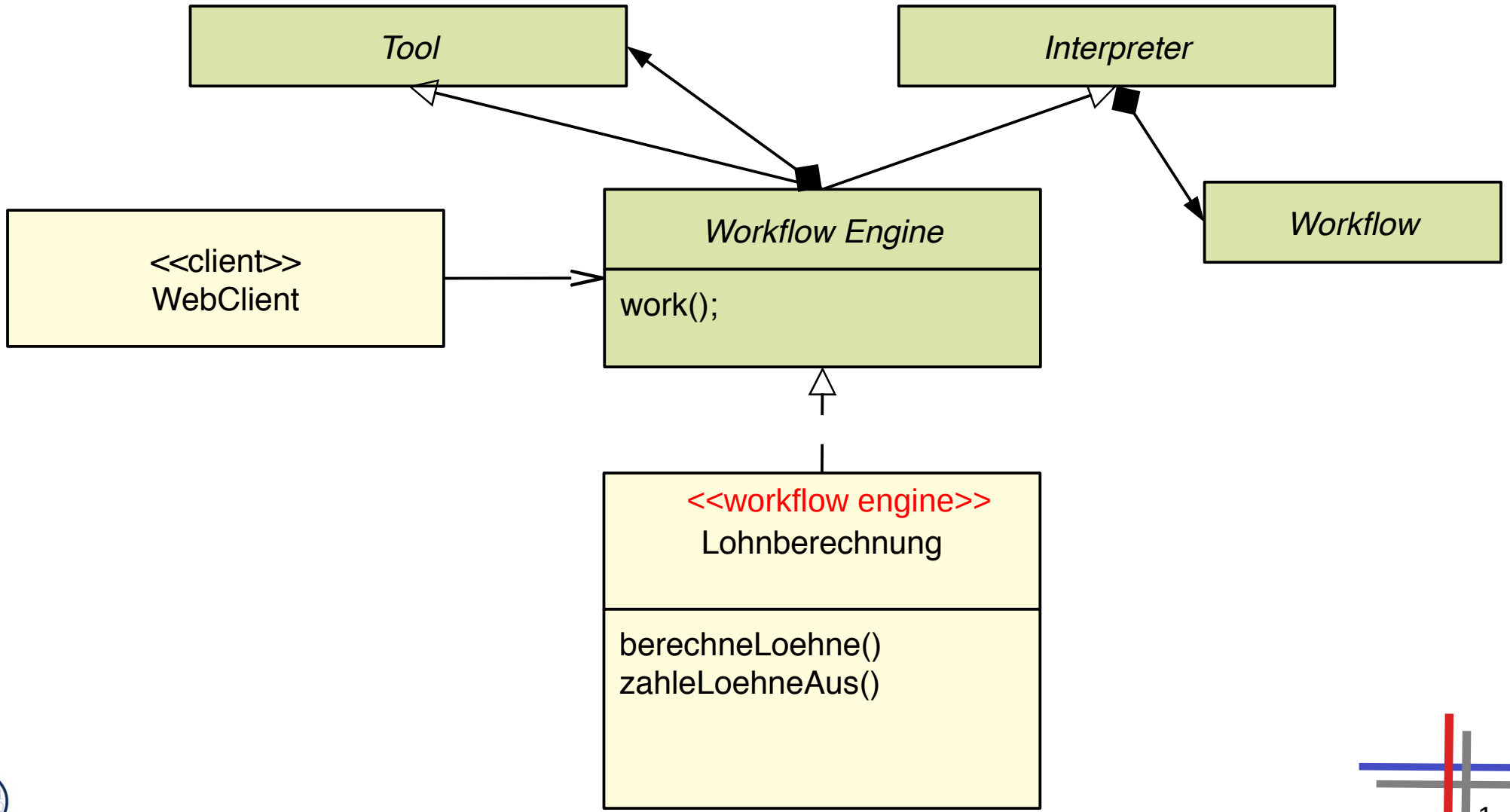
- Technische Tools tragen eine technische Funktionalität, die anwendungsunspezifisch ist
  - Bsp.: Editor, Lister, Inspektor, Browser, Verschlüsseler, Komprimierer, Optimierer
- Technische Tools liegen direkt über dem Material




[Züllighoven]

# Workflow-Engine-Klassen und -Schnittstellen

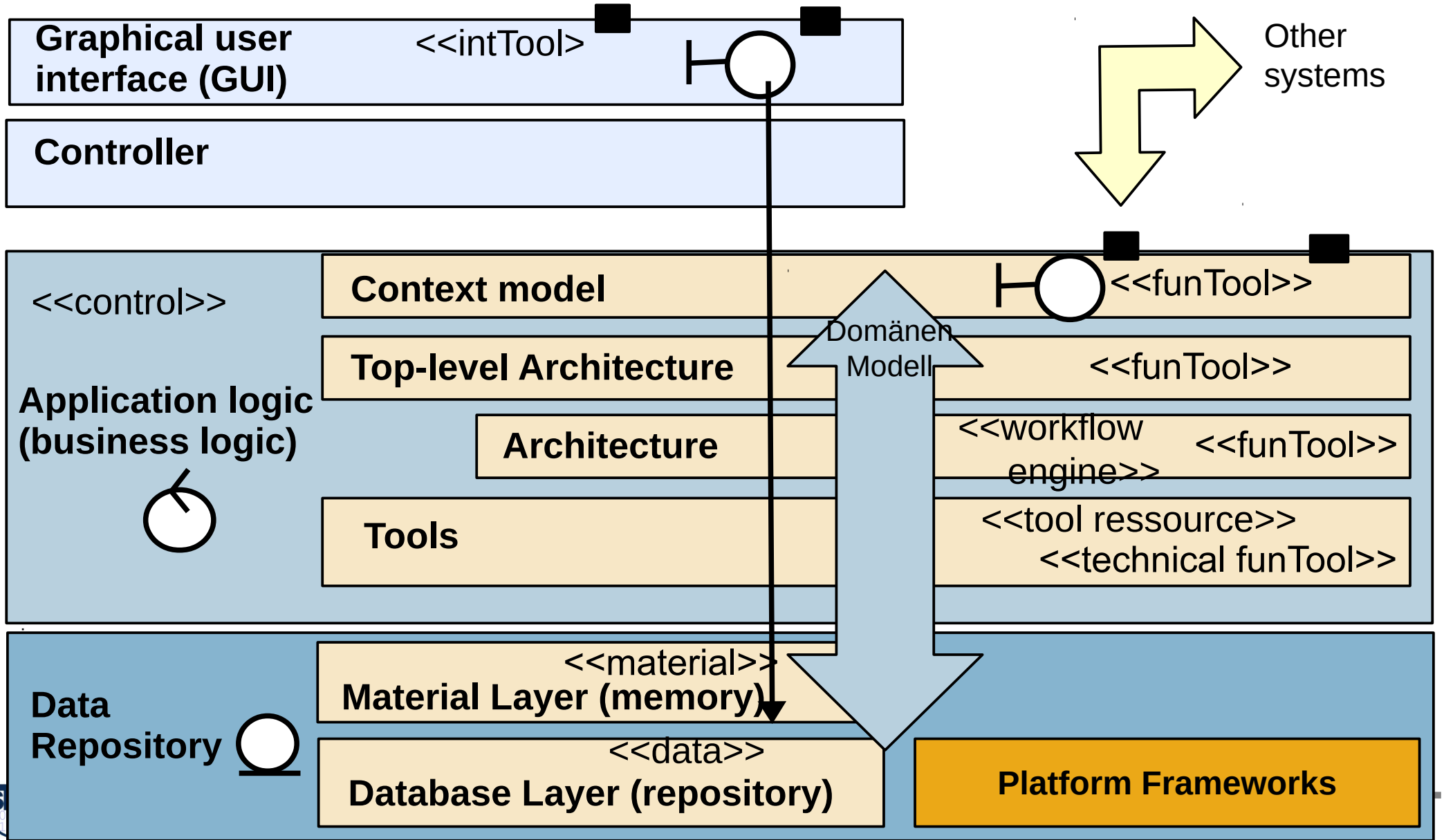
- Workflow-Engine arbeiten einen Arbeitsablauf (Workflow) ab
- Workflow-Engines rufen andere Tools auf, setzen also auf ihnen auf



# Schichten und TAM-Klassifikation

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

# Q7': Verfeinerte BCE-Schichtung eines Systems



# 44.2 Verfeinerungsbeispiel für Objektanreicherung in der Analyse

.. Verfeinerung durch Integration von Unterobjekten..

Teile und Rollen

# Objektanreicherung in der Analyse

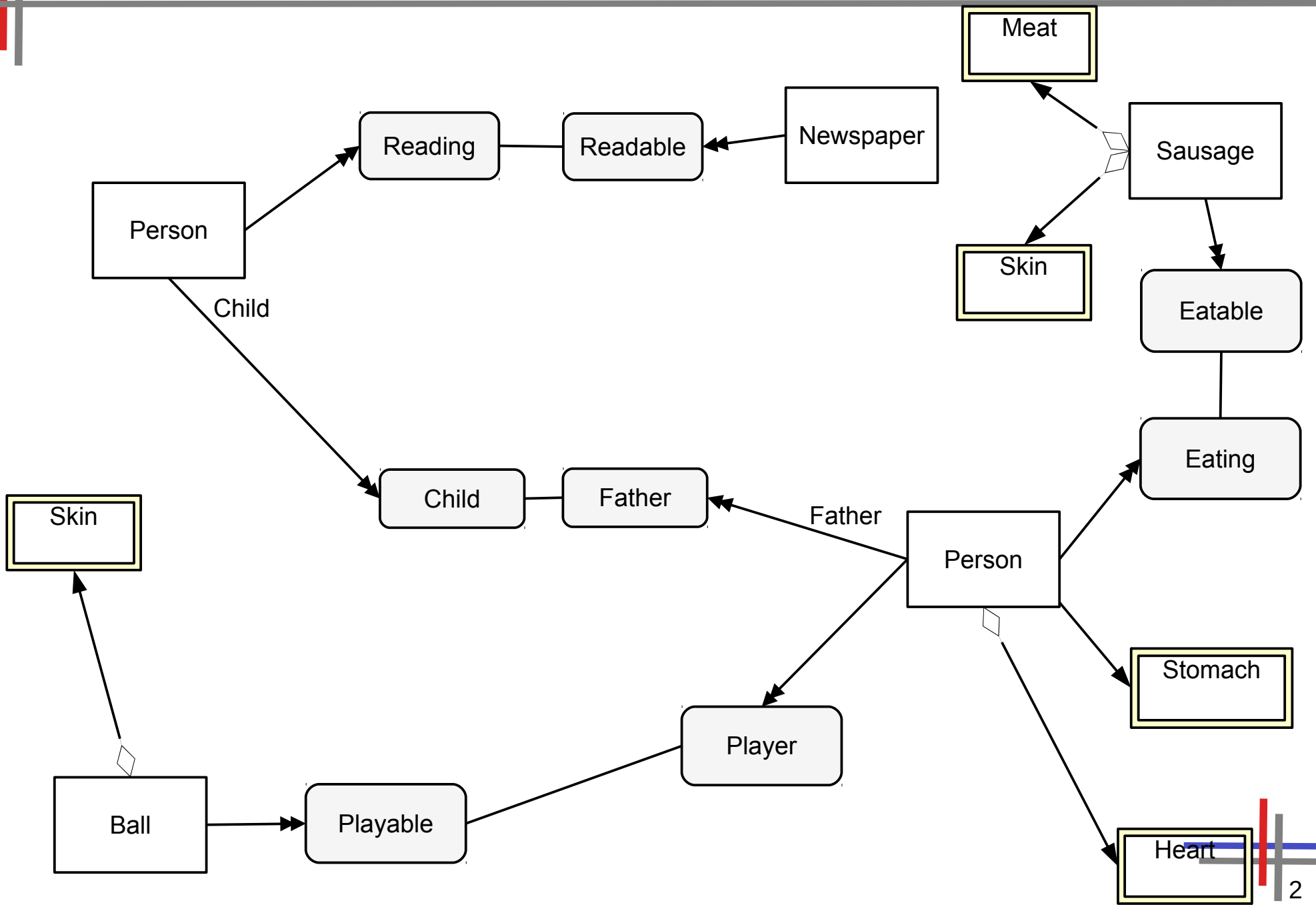
- **Objekt-Anreicherung (Object fattening) durch Unterobjekte** ist ein Verfeinerungsprozess, der an ein Kernobjekt aus dem Domänenmodell Unterobjekte anlagert, die
  - Teile ergänzen (Teile-Verfeinerung)
  - Rollen ergänzen (Konnektor-Verfeinerung), die Beziehungen klären zu
    - Plattformen (middleware, 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)

Person

Newspaper

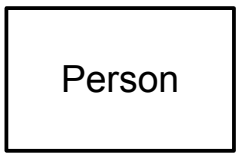
Sausage

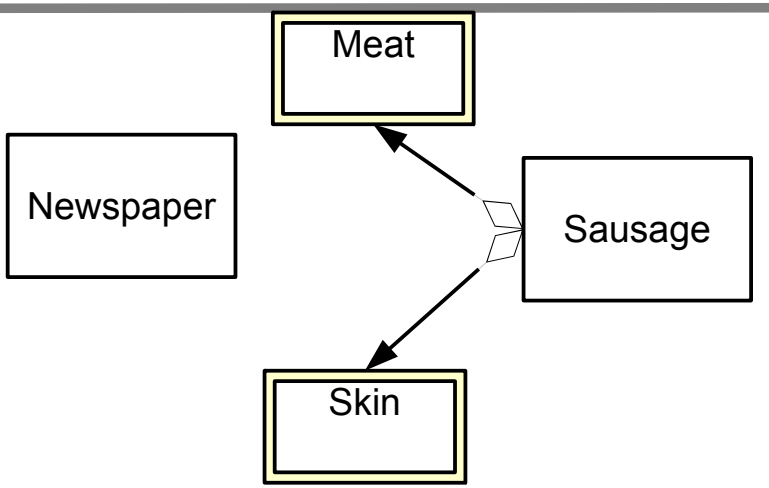
Person

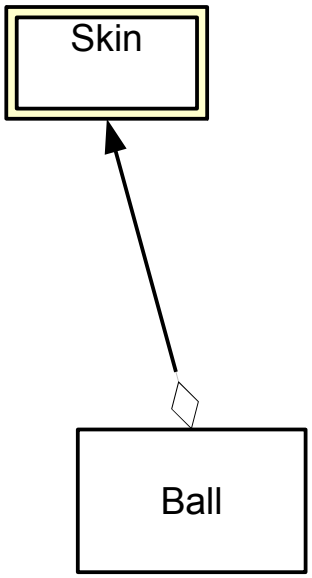
Ball

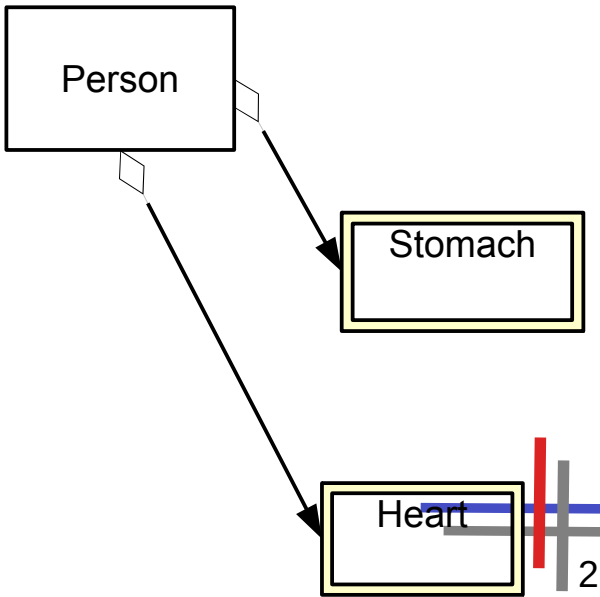
# Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 1: Teile-Verfeinerung



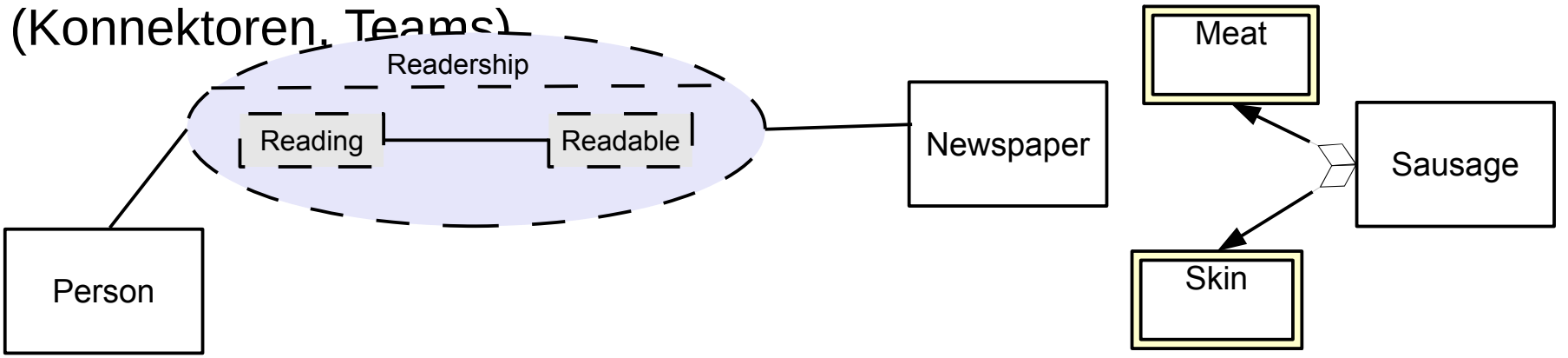



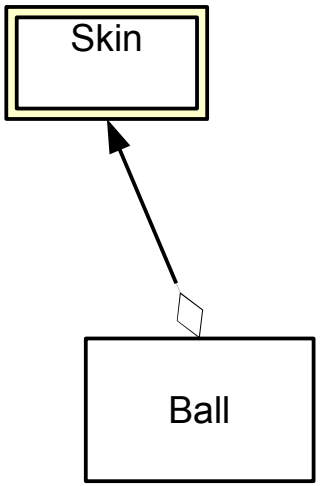



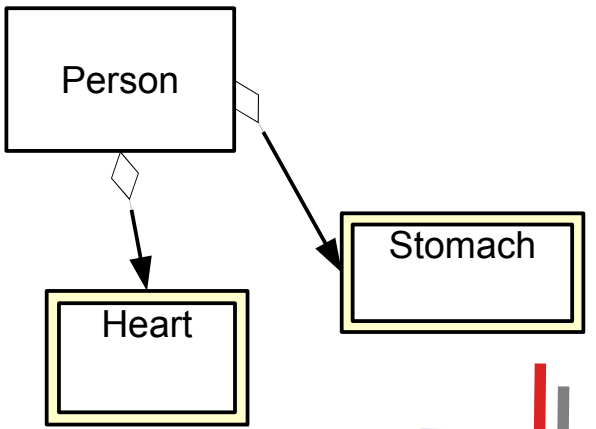



# Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

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

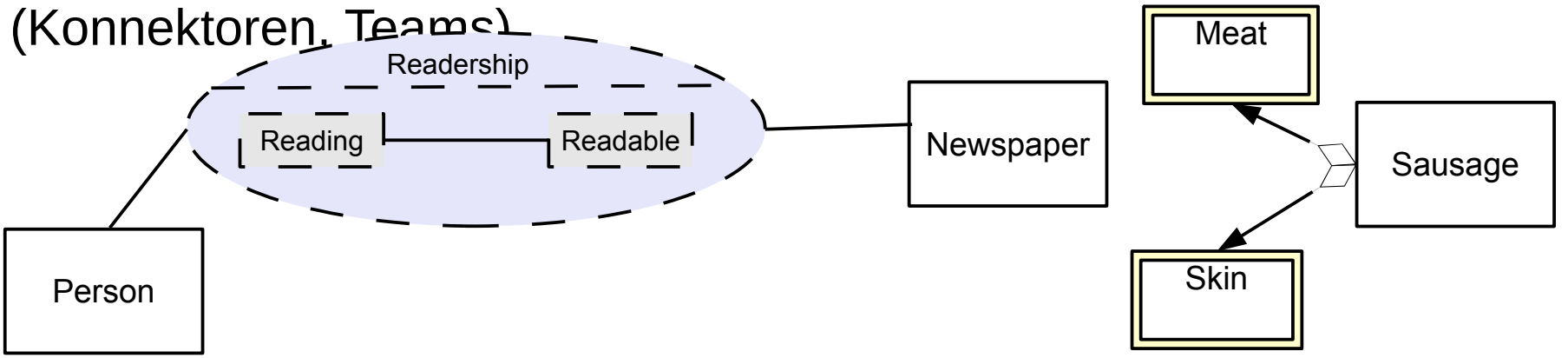


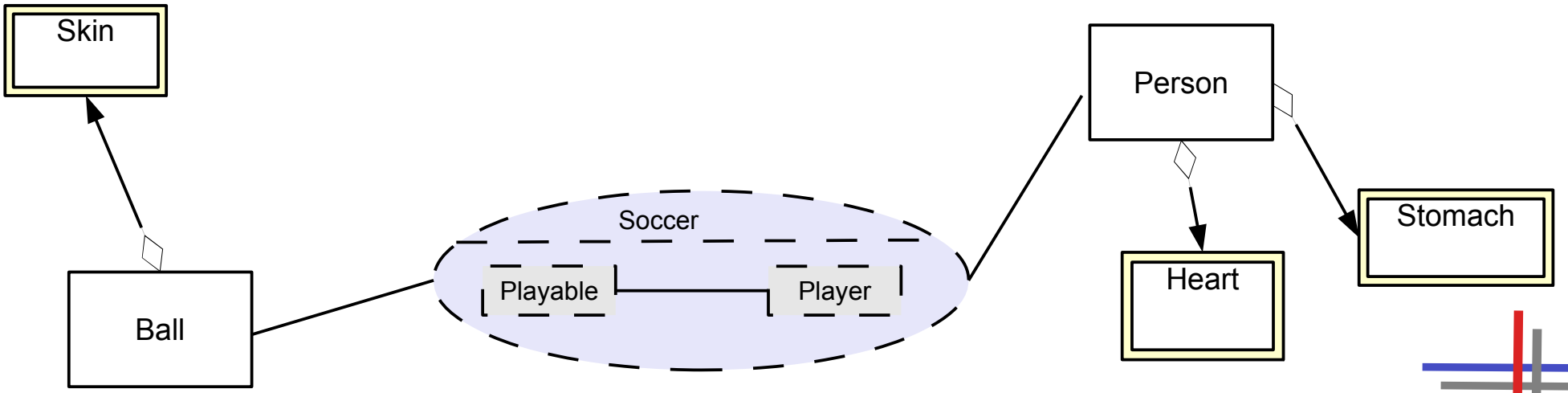



# Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

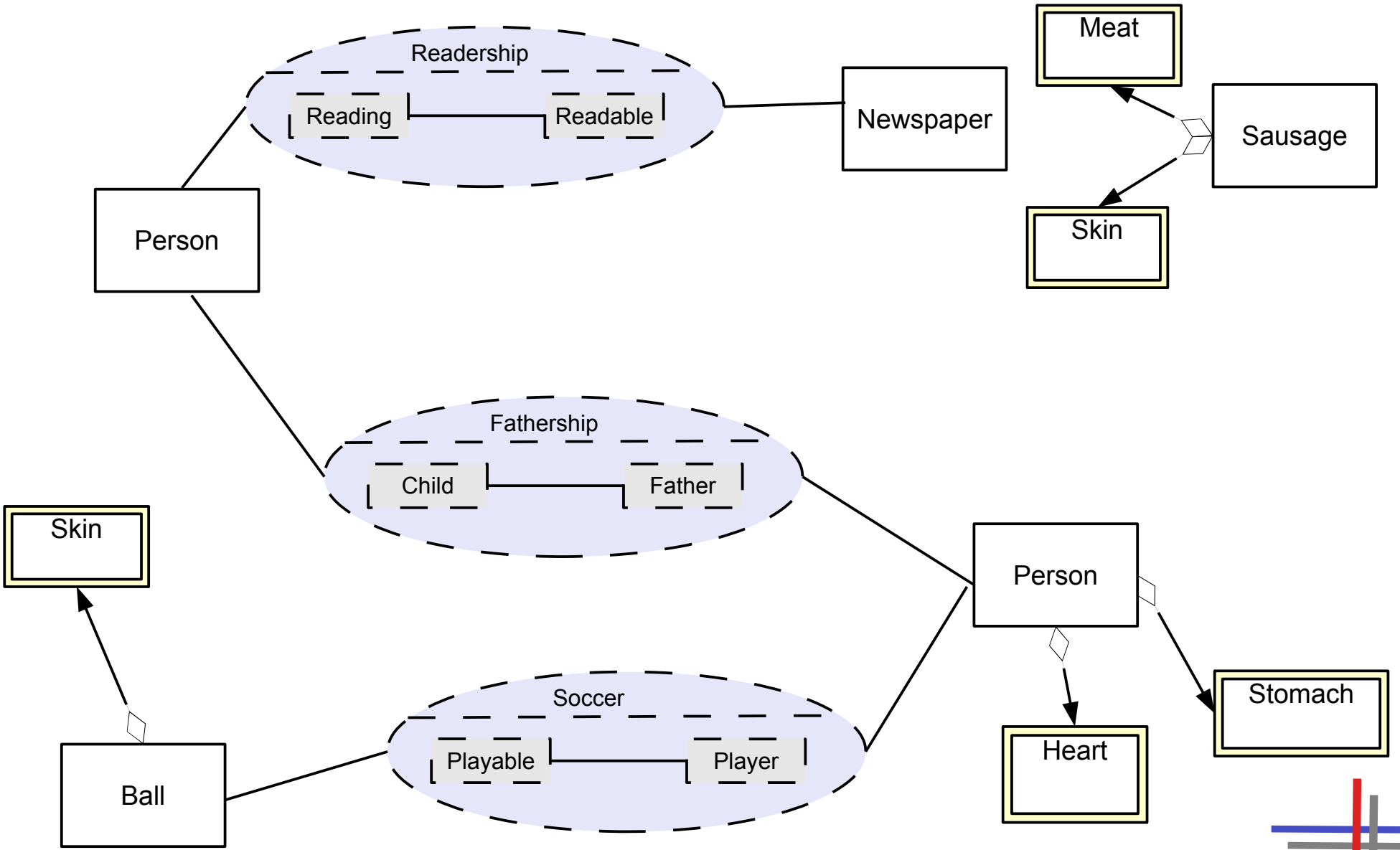
- Schritt 2: Schrittweise Erweiterung durch Kollaborationen (Konnektoren, Teams)



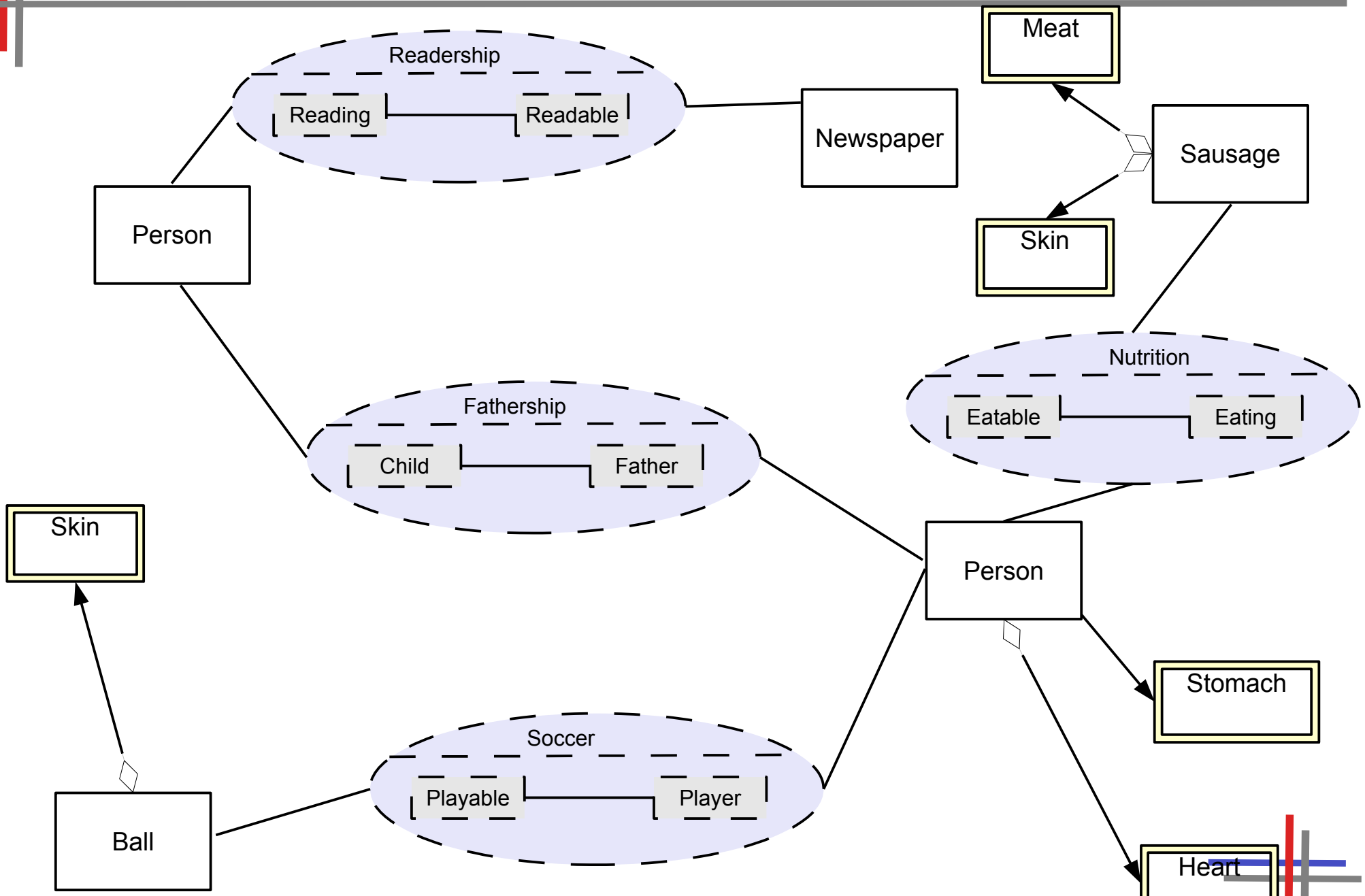



# Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 2: final: alle Kollaborationen



# Personen-Analysemodell – Angereichert durch Einziehen von querschneidenden Kollaborationen

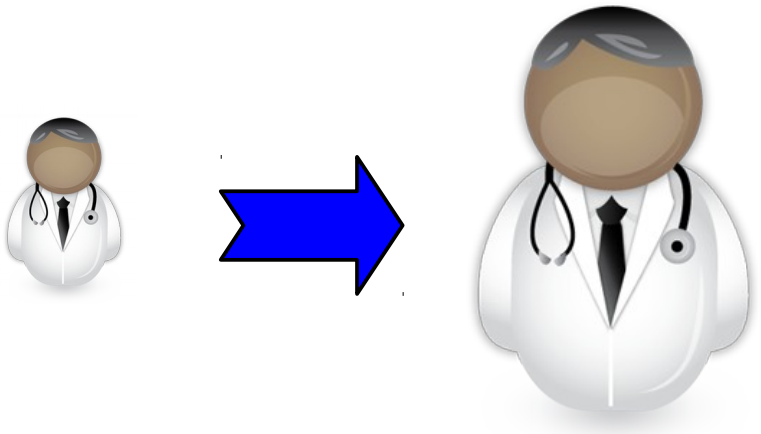


# 44.3 Objektanreicherung mit Plattforminformation (Querschneidende Verfeinerung für Plattformen)

.. 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

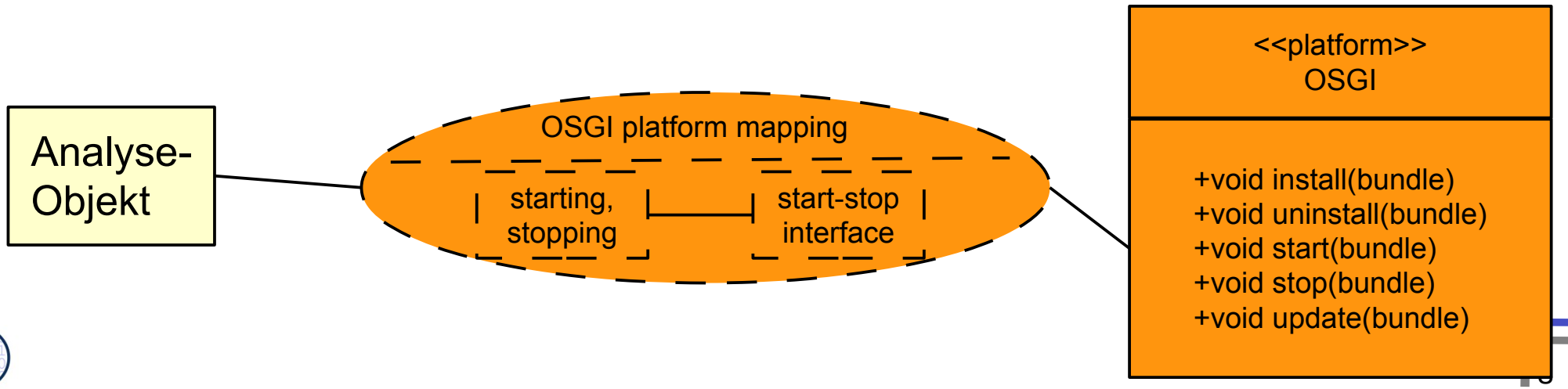


# Plattformanreicherung – Weitere Schritte im Entwurf

- 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 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

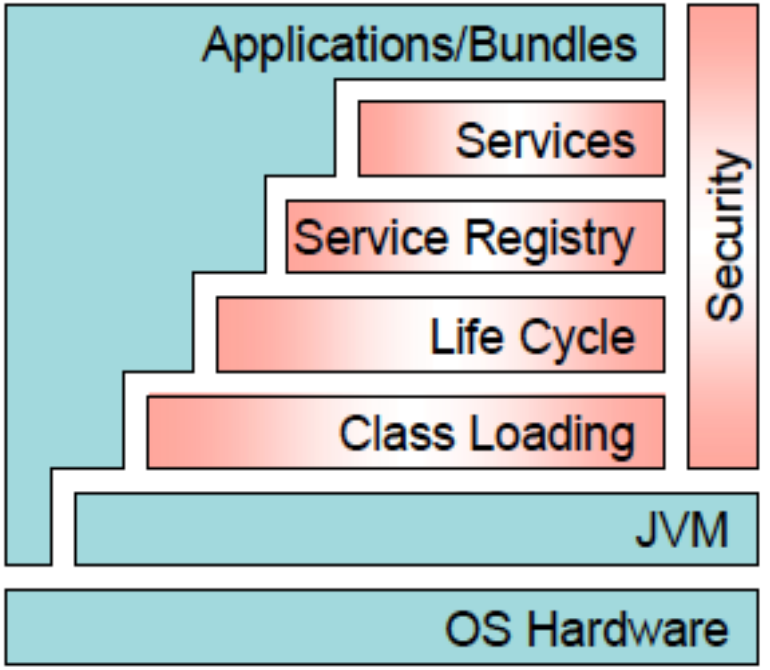
# Plattformobjekte und -konnektoren

- 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



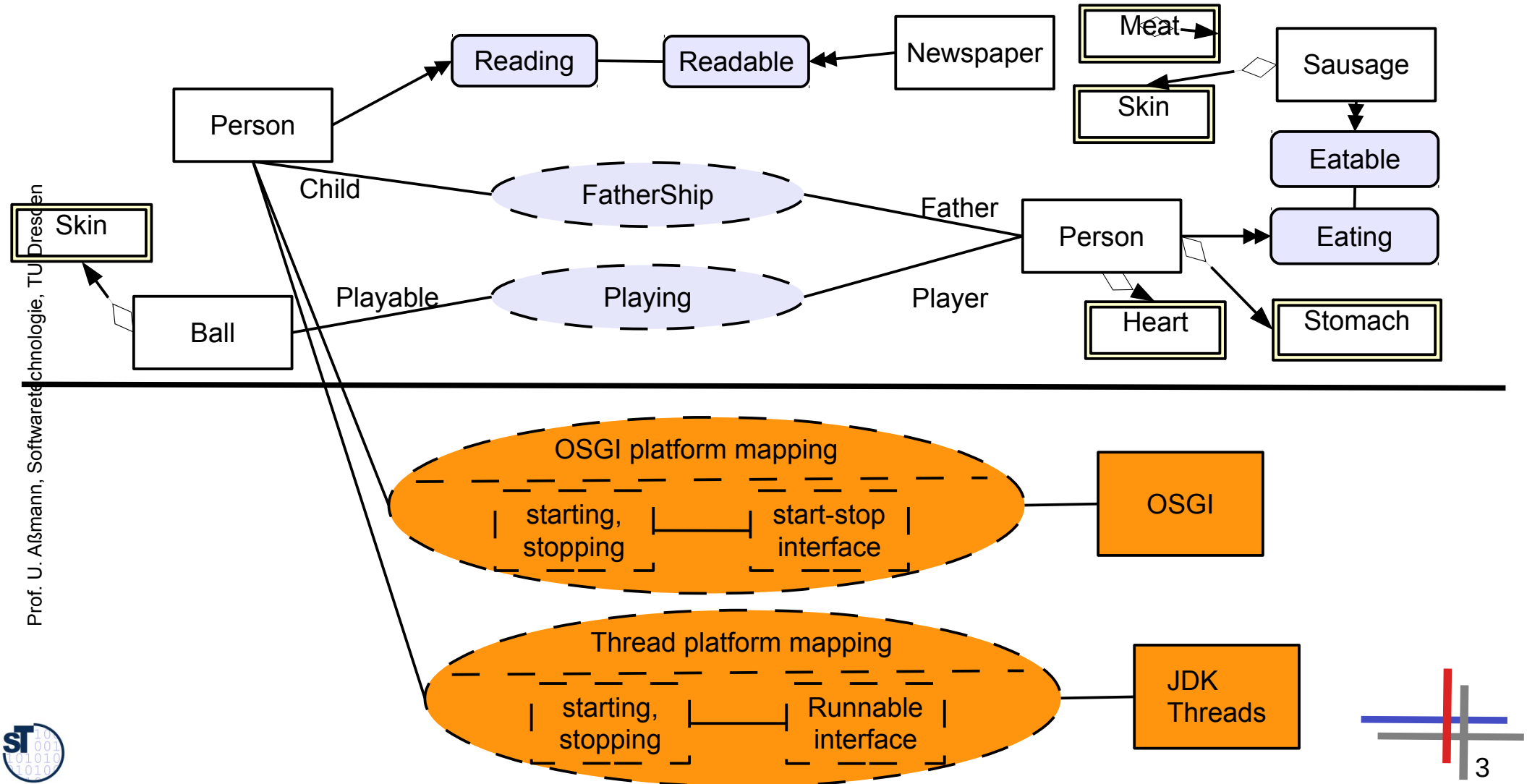
# Plattformobjekt OSGI

- 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]



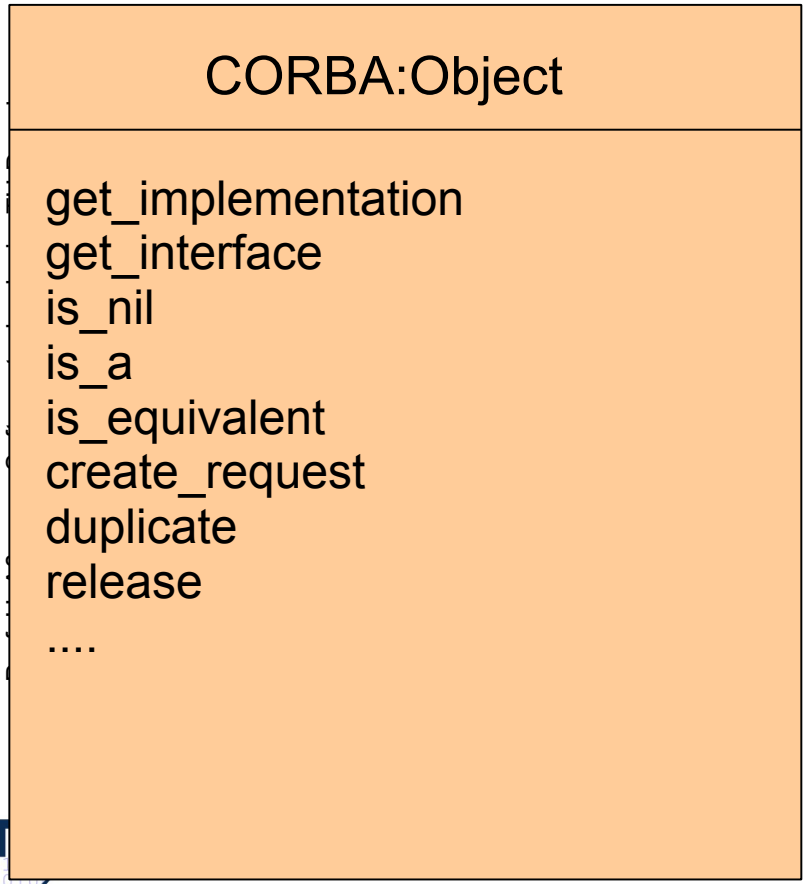
# Mit Verfeinerung durch Plattform-Konnektoren (platform fattening)

- 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

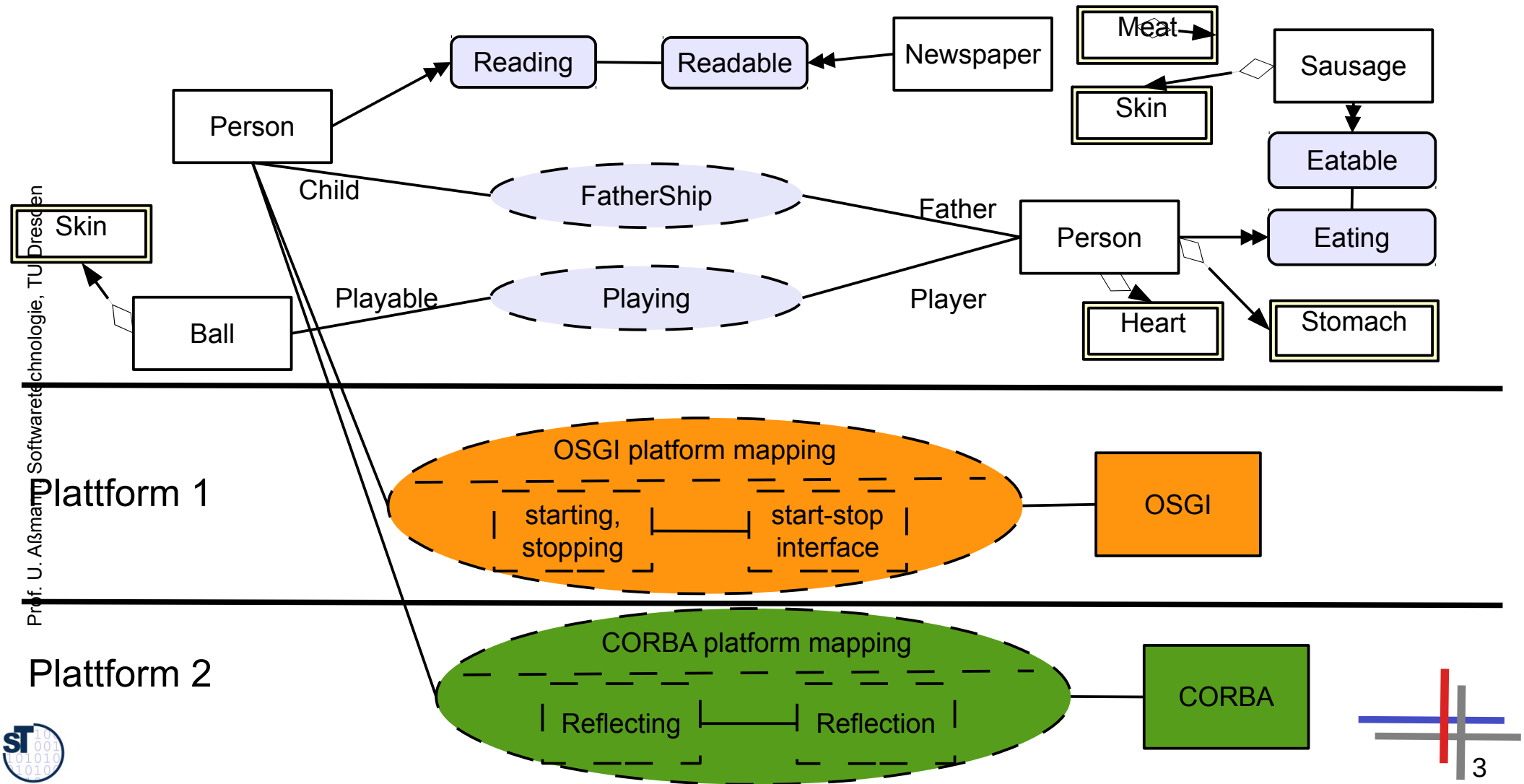
- CORBA bildet eine Komponentenplattform für heterogen programmierte Systeme




- 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

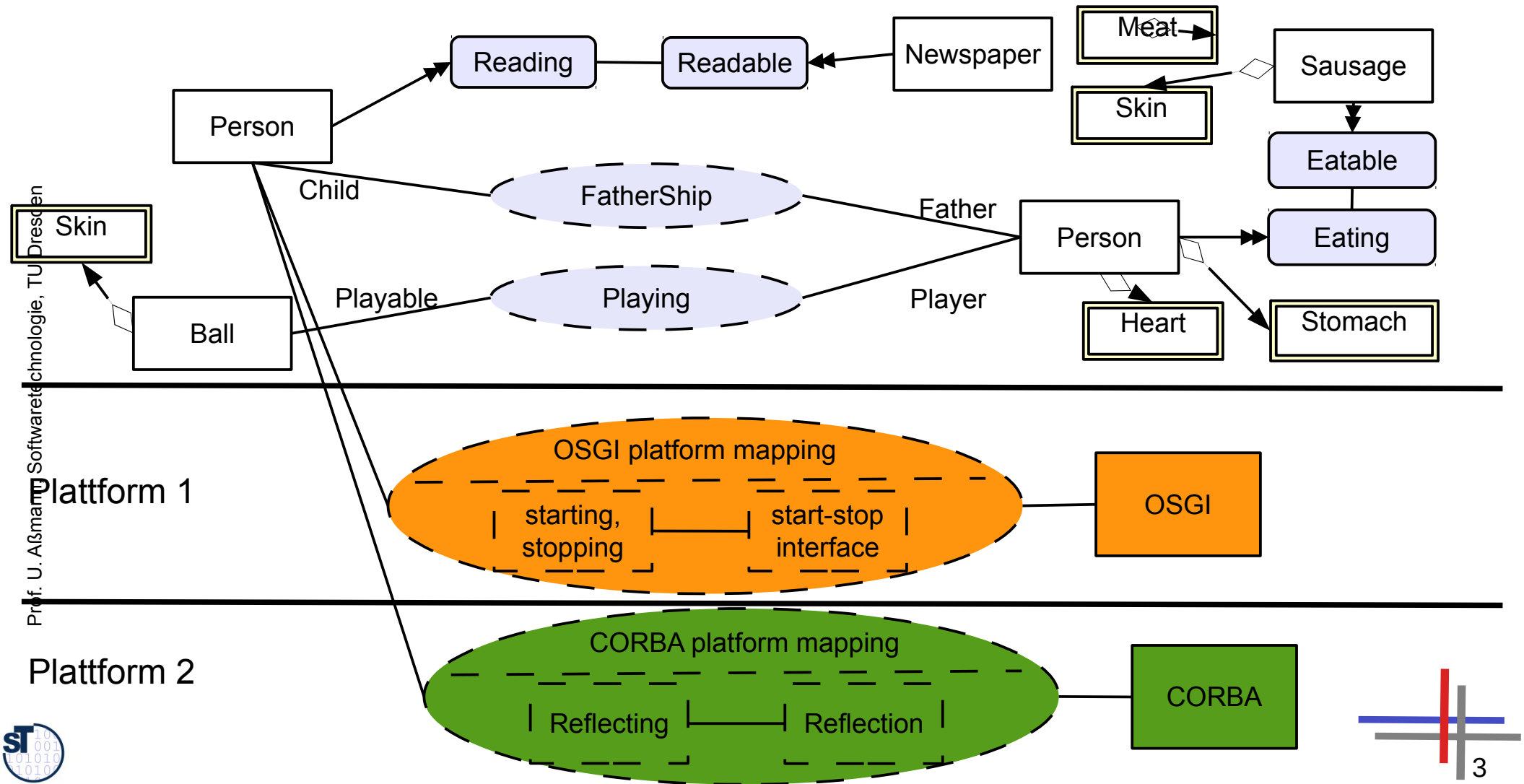
Kapselt man Plattformabhängigkeiten in einen Plattformkonnektor, können sie leicht ausgetauscht werden und die Software wird portabel.

# 44.4 Abbildung der Integrationsrelation auf klassische Programmiersprachen

.. in der Implementierung ..

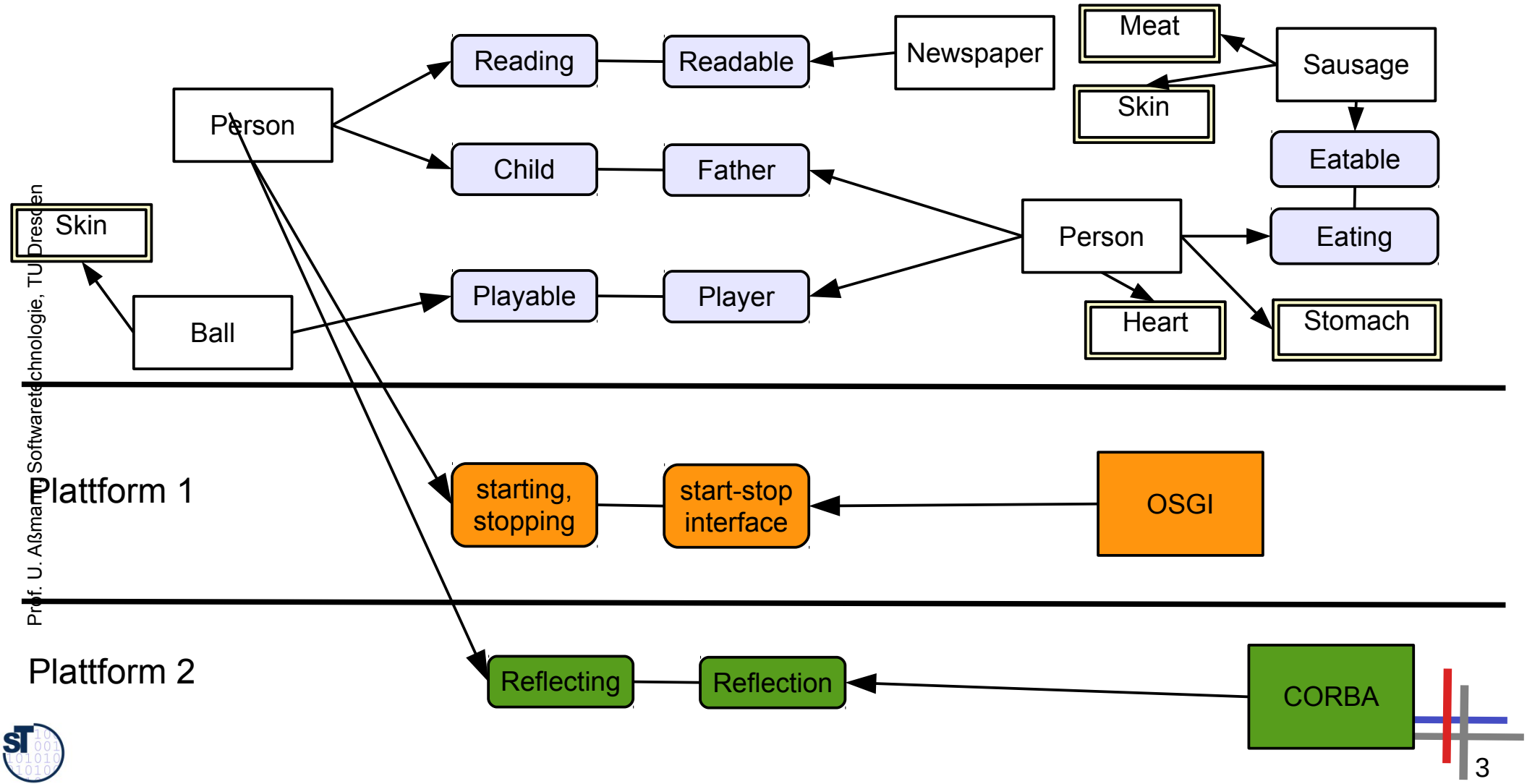
# Wie bilde ich “integrates-a” ab?
# a) mit einer Rollen-Programmiersprache

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



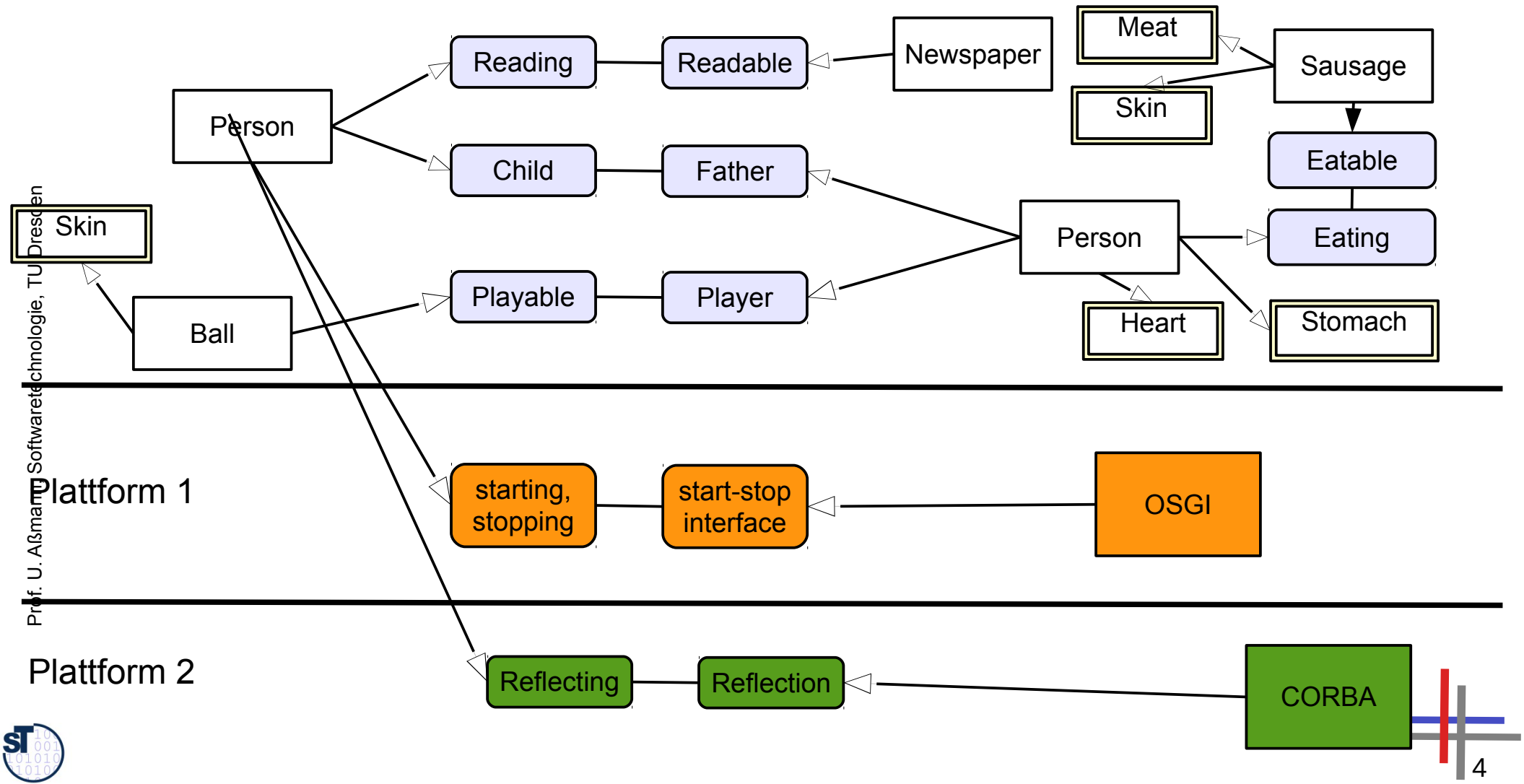
# b) Wie bilde ich “integrates” durch Delegation ab?

- 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



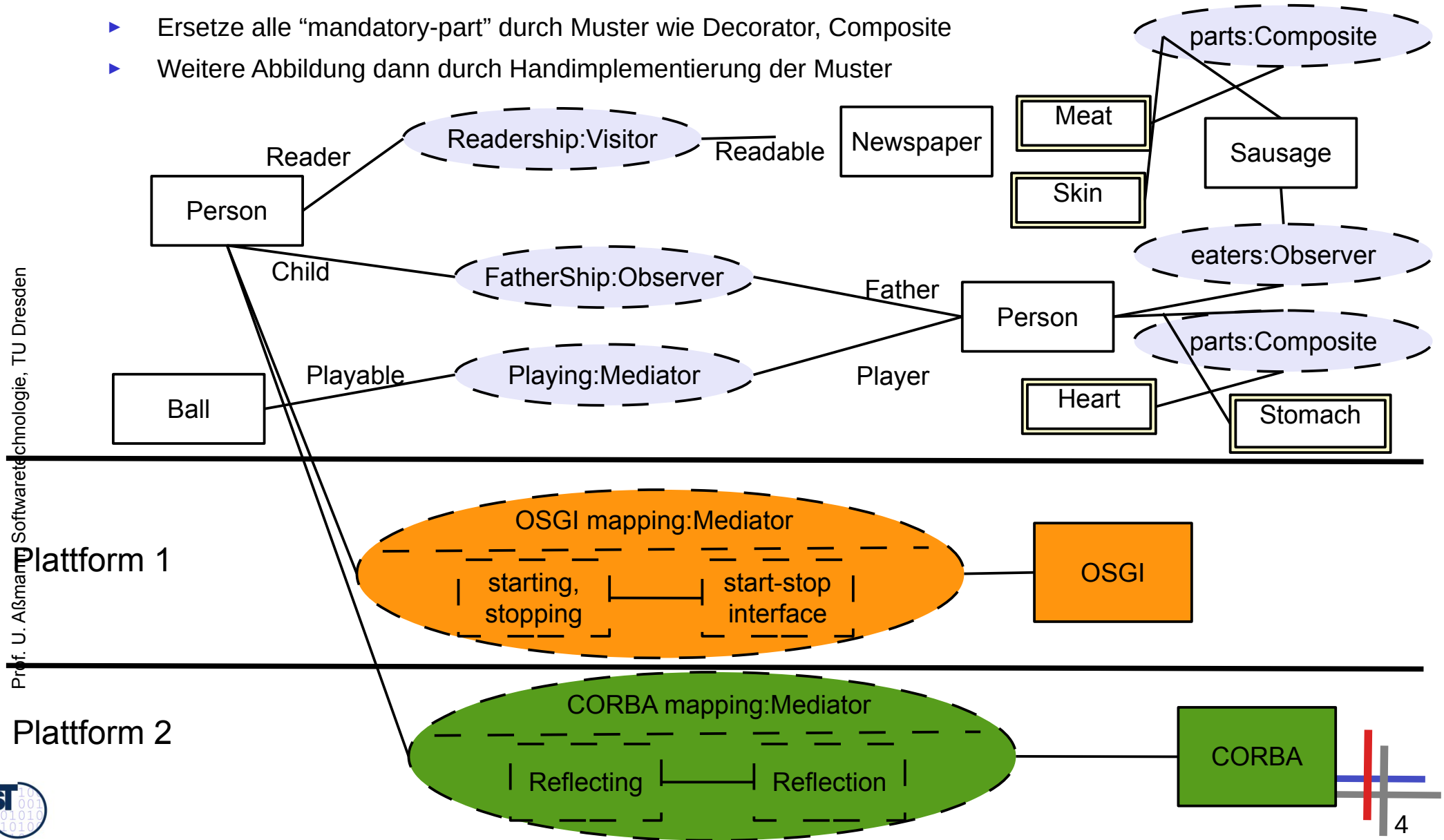
# c) Wie bilde ich “integrates” durch Vererbung ab?

- 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?

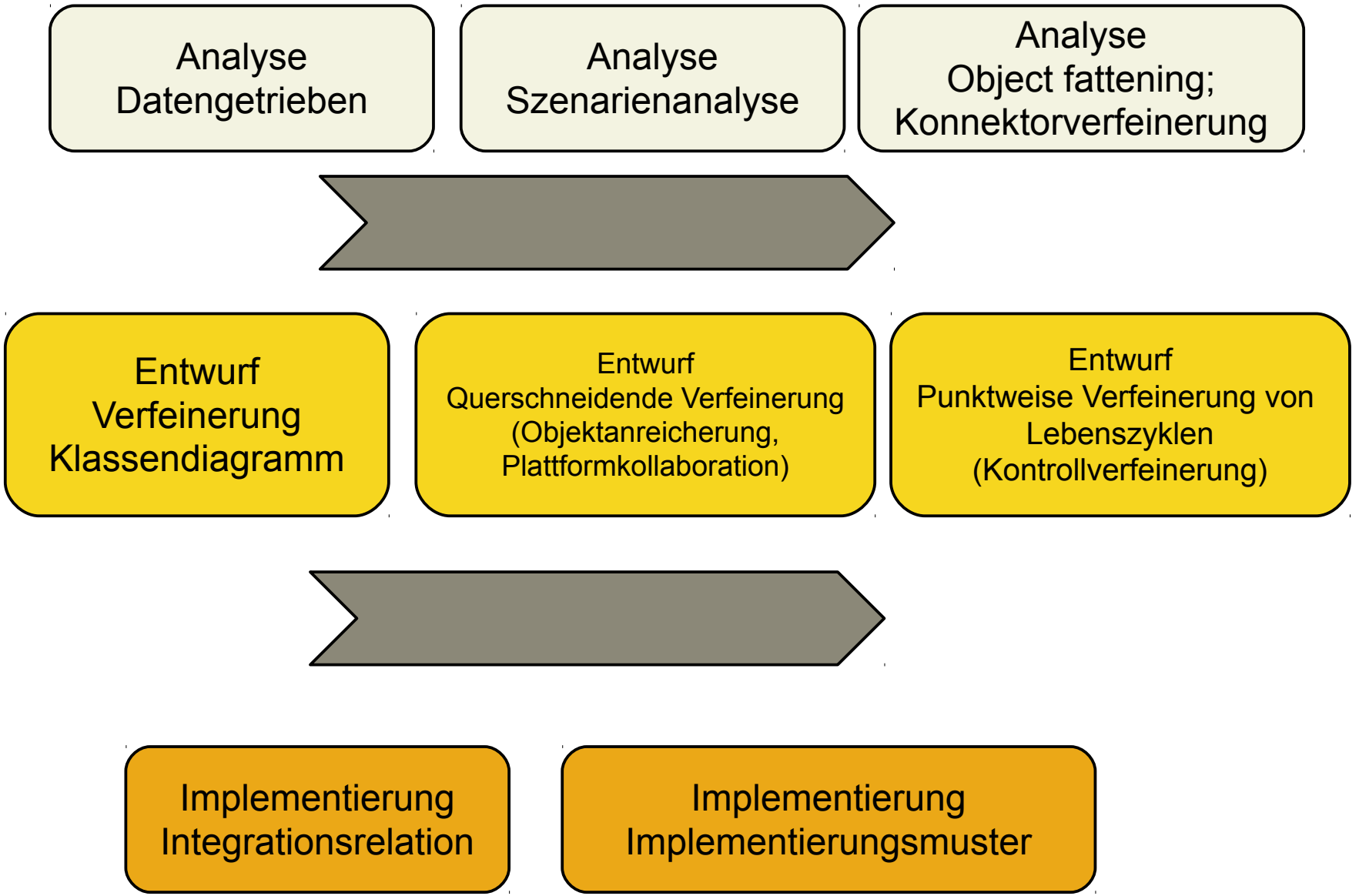
- Ersetze alle “integrates”, “plays”,  etc. durch Muster wie Observer, Visitor
  - Dynamische Komposition der Initial- und Terminal-Botschaften
- Ersetze alle “mandatory-part” durch Muster wie Decorator, Composite
- Weitere Abbildung dann durch Handimplementierung der Muster



# 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

# 44.5 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