42. Die Schichtenarchitektur im Detail

Weitere Tipps zur Gestaltung der Anwendungslogik- und Datenhaltungs-Schicht mit TAM-Kollaborationen und Plattformkonnektoren

Prof. Dr. rer. nat. Uwe Aßmann

Institut für Software- und Multimediatechnik

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

TU Dresden

Version 17-0.4, 03.07.17

1) Perspektivenmodell TAM
2) Verfeinerung mit Kollaborationen
3) Plattformverfeinerung mit Plattform-Konnektoren
4) Abbildung der plays-Relation
5) Gesamtbild der Verfeinerung

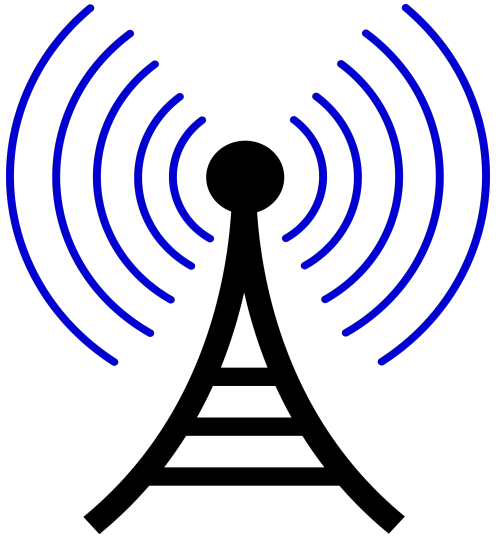
DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Einschreibung in das Softwarepraktikum WS 2017/18

- Einschreibung erfolgt für ALLE Studenten (auch für ISTler) ab 03. Juli 2017 über www.jexam.de !
- Einschreibung **befristet bis 31. Juli 2017**
  - **Internes (SalesPoint)-Praktikum** (Standardfall)
  - **Externes Praktikum** (Firma oder andere Institution)
- Achtung!
  - Entweder in das externe oder in das interne Praktikum einschreiben! Wer keinen externen Praktikumsplatz erhält, bekommt automatisch einen internen Praktikumsplatz.
  - Wer sich in das Praktikum bis zum 31. Juli 2017 nicht eingeschrieben hat, kann wahrscheinlich nicht teilnehmen, sich jedoch auf die Nachrückliste einschreiben!
- Teamwünsche können im August abgegeben werden. Verfolgen Sie dazu die News auf der Webseite der LV!

Teil IV - Objektorientierter Entwurf (Object-Oriented Design, OOD)

1) 40: Überblick
2) 41: Einführung in die objektorientierte Softwarearchitektur
    1) Architekturprinzipien, Architekturstile, Perspektivenmodelle
    2) Modularität und Geheimnisprinzip
    3) BCD-Architekturstil (3-tier architectures)
3) **42: Schichtenarchitektur im Detail**
    1) **TAM-Kollaborationen**
    2) **Plattform-Konnektoren: Verfeinerung mit querschneidender Objektanreicherung**
4) 43: Architektur interaktiver Systeme
5) 44: Punktweise Verfeinerung von Lebenszyklen
    - Verfeinerung von verschiedenen Steuerungsmaschinen



Waltzing Matilda, Waltzing Matilda,

You'll come a-Waltzing Matilda with me,

And he sang as he watched and waited till his billy boiled,

You'll come a-Waltzing Matilda with me.

https://de.wikipedia.org/wiki/Waltzing_Matilda

Tippelei

Walz

Kundschaft

Zunftlade

Meisterstück

Bundesarchiv, Bild 183-1990-1210-001 / CC-BY-SA 3.0

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.

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

Ein Vorschlag für die Konnektion von Anwendungslogik und Datenhaltung

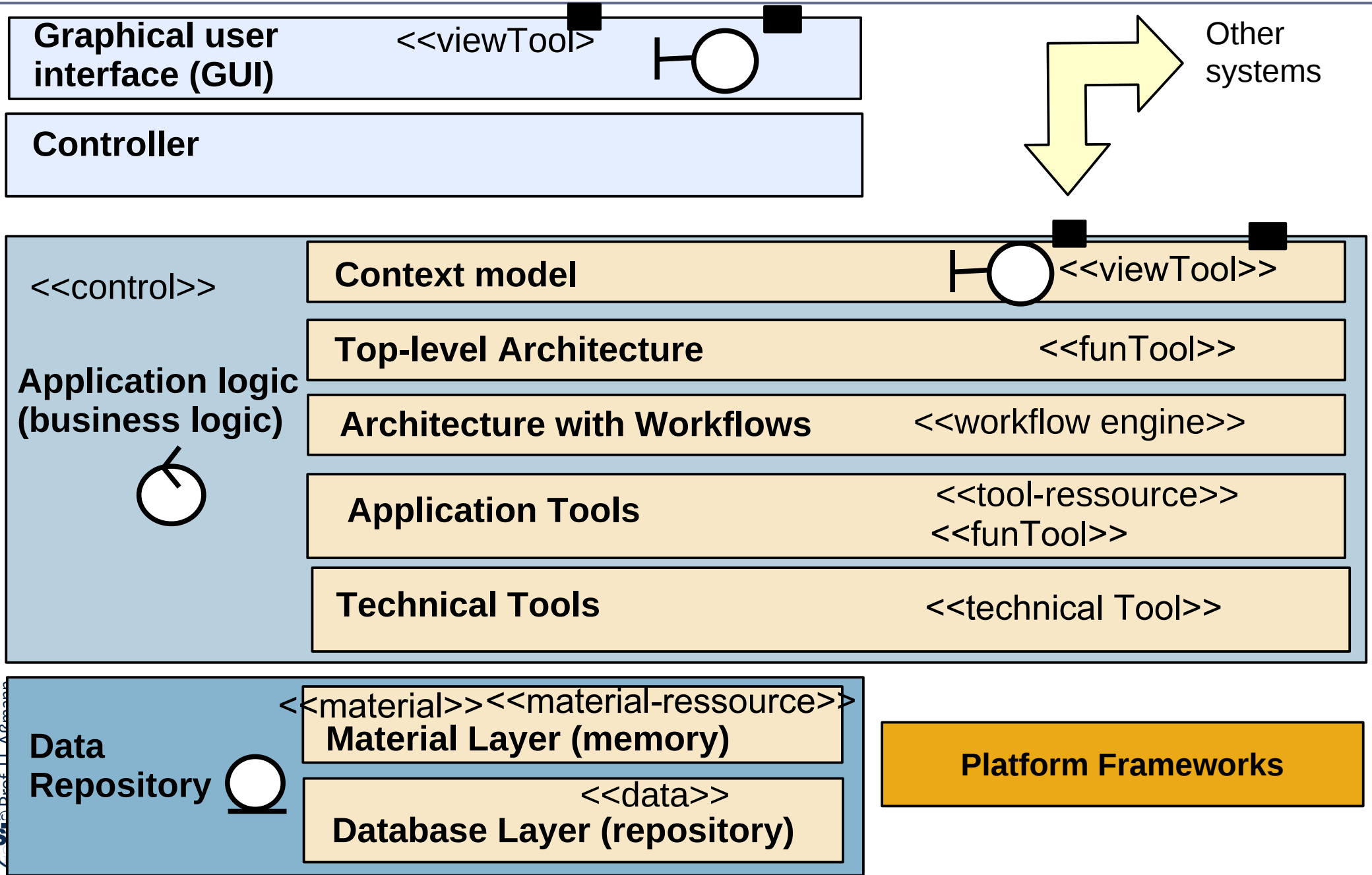
Was wird interaktiv (asynchron) aufgerufen?

Was ist passiv?

Was muss belegt werden?

Welche Klasse wird in welche Schicht eingeordnet?

Vorausschau:
Q7': Verfeinerte BCE-Schichtung eines Systems mit TAM

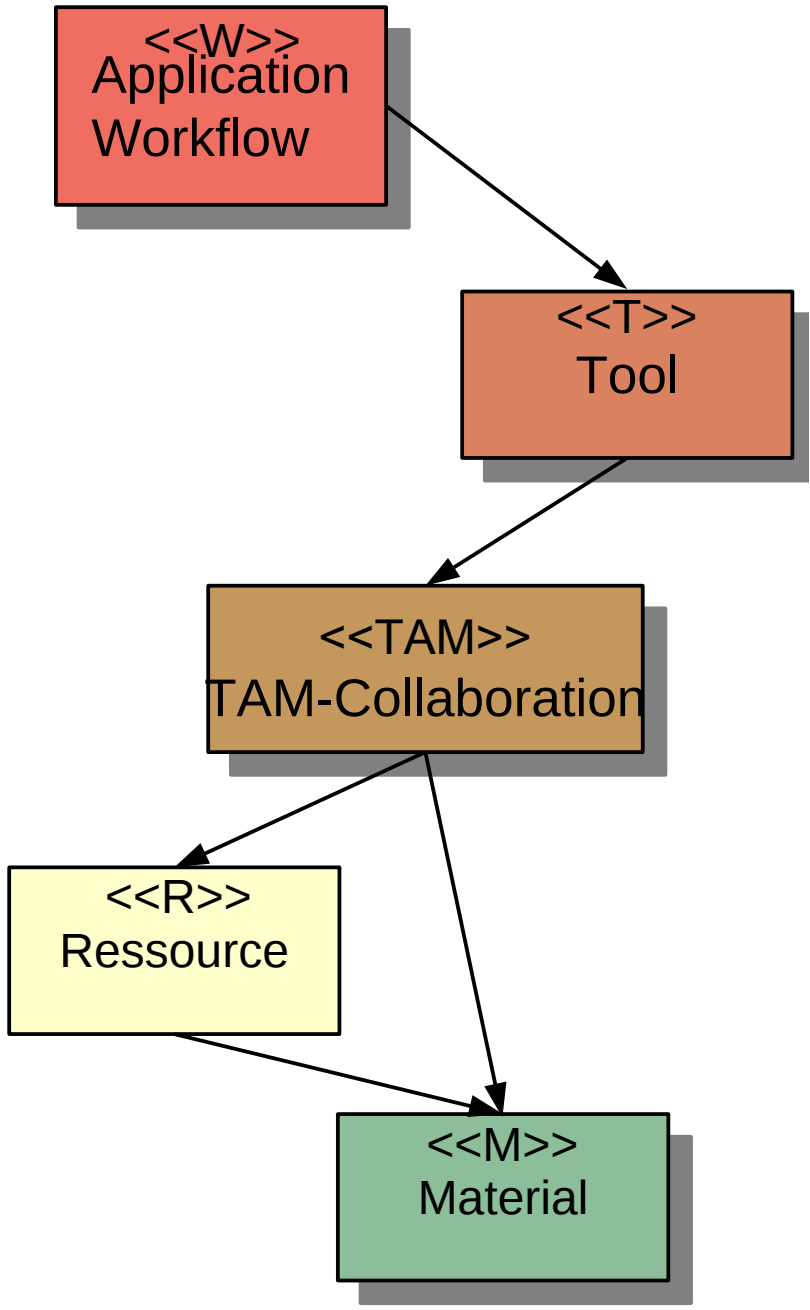


Perspektivenmodell TAM: Trennung von aktiven und passiven Komponenten

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

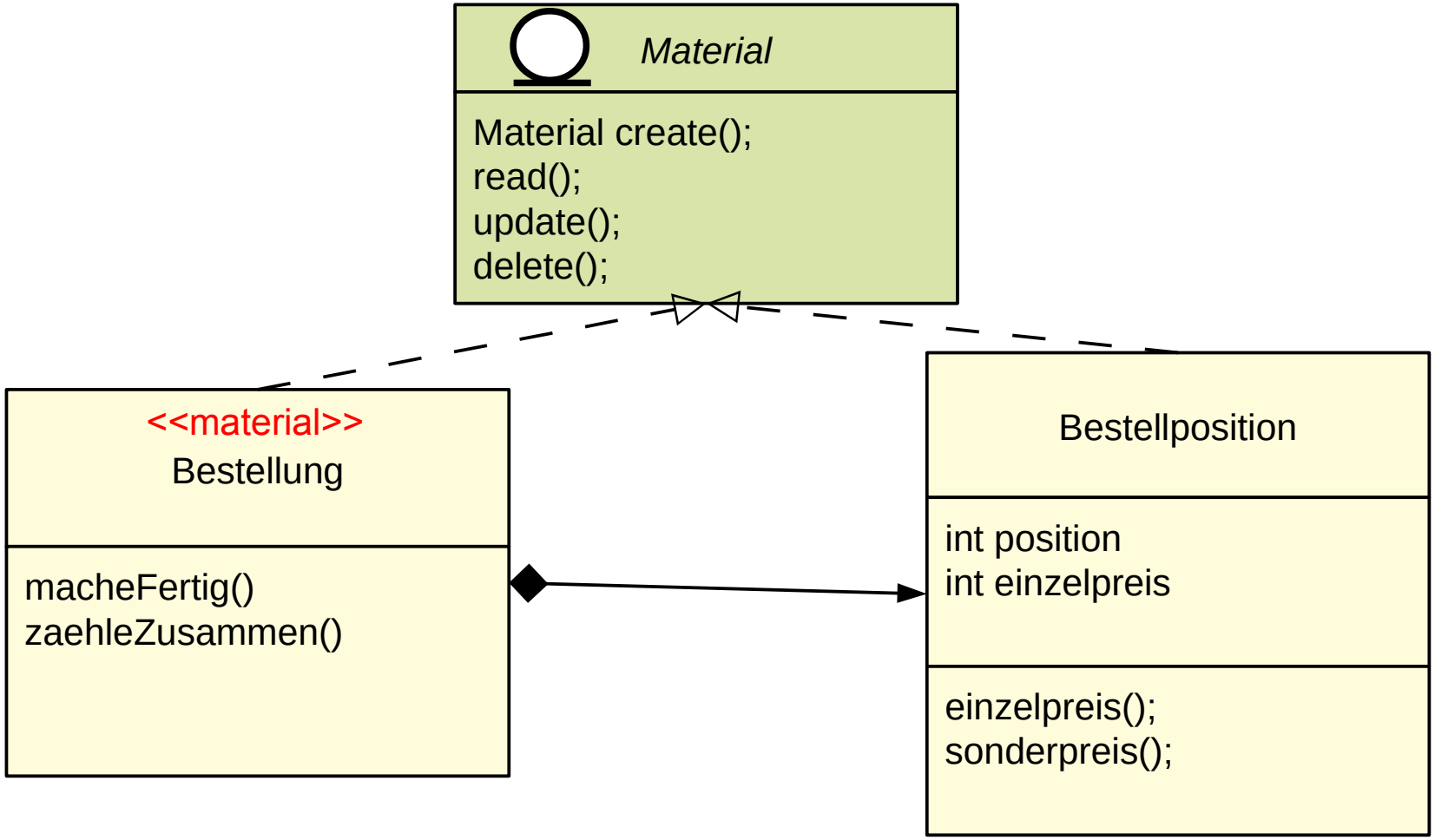
1) Tools (aktive Prozesse)
2) Ressources (belegbar)
3) Materials (passive Daten, Schicht E)
4) TAM-Collaboration
5) Workflows koordinieren Tools

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



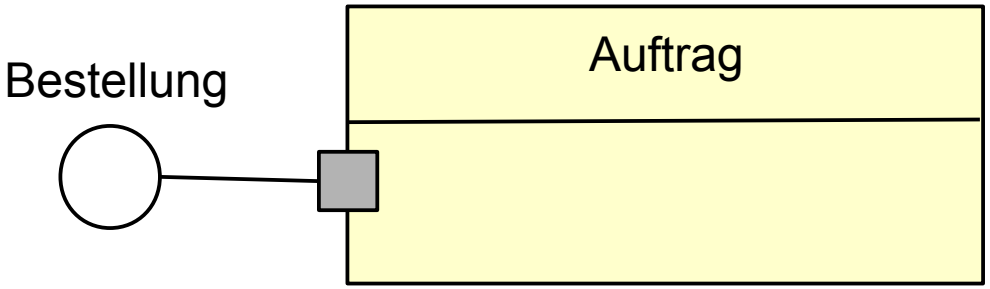
Material-Klassen und -Schnittstellen

- **Materialobjekte** sind passiv, d.h. werden von außen aufgerufen und geben den Steuerfluss nach außen hin zurück
  - Liegen in der Datenablage-Schicht (D)
- Materialobjekte können komposit sein (Muster Composite)
- Materialien folgen der CRUD-Schnittstelle (create, read, update, delete)



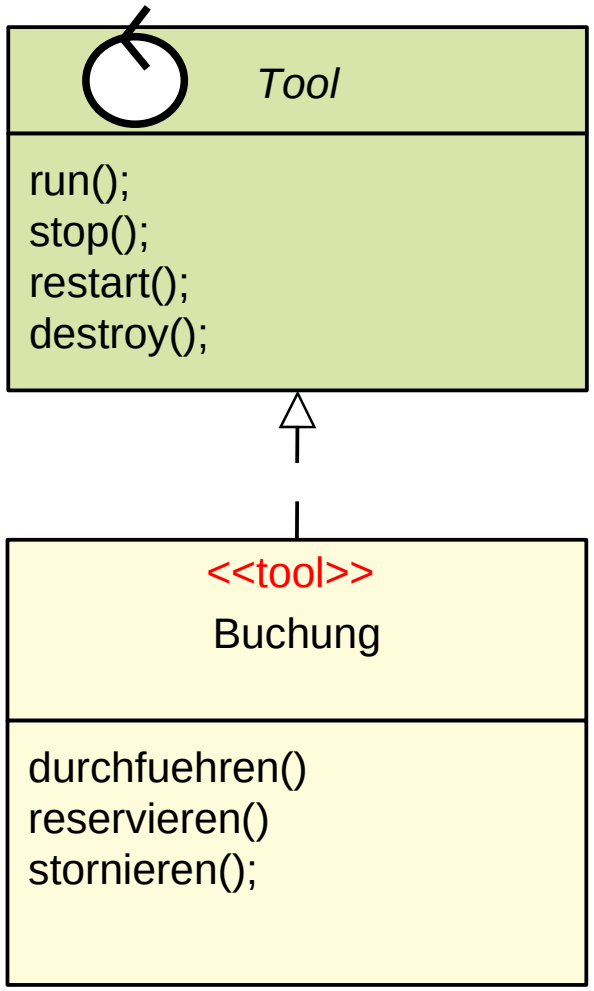
Material-Klassen und -Schnittstellen

- Materialien können in Ports auftauchen



Tool-Klassen und -Schnittstellen

- Toolobjekte sind I.d.R. aktiv, besitzen eigenen Steuerfluss (thread, process)
- Tools liegen in der Anwendungslogik (C-Schicht)



[Züllighoven]

Das TAM-Gesetz (Tools-Materials Gesetz)

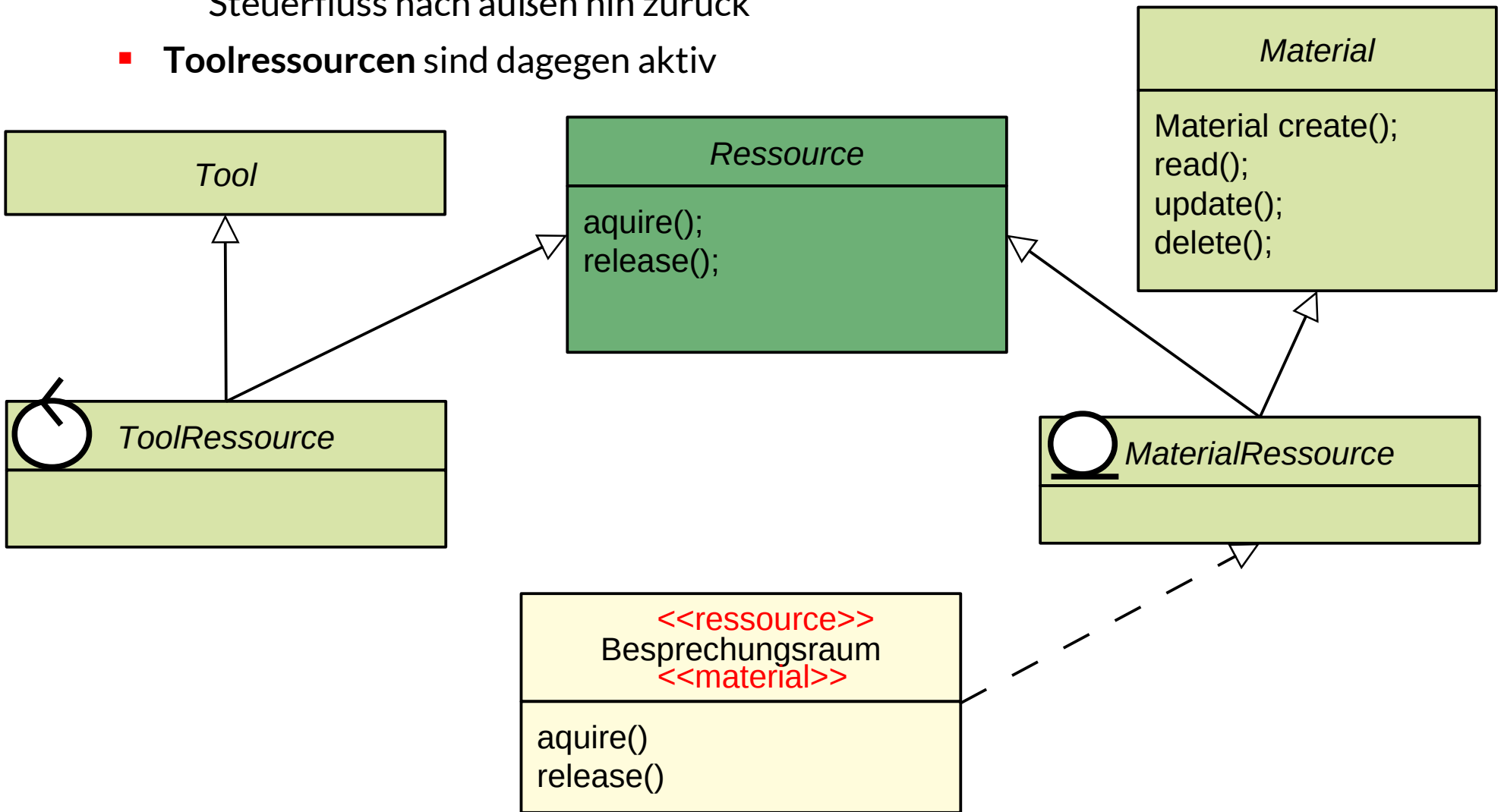
Trenne Tool- von Material-Klassen, denn sie gehören auf verschiedene Schichen des Systems.

<<Tool>>
<<A-Schicht>>

<<Material>>
<<D-Schicht>>

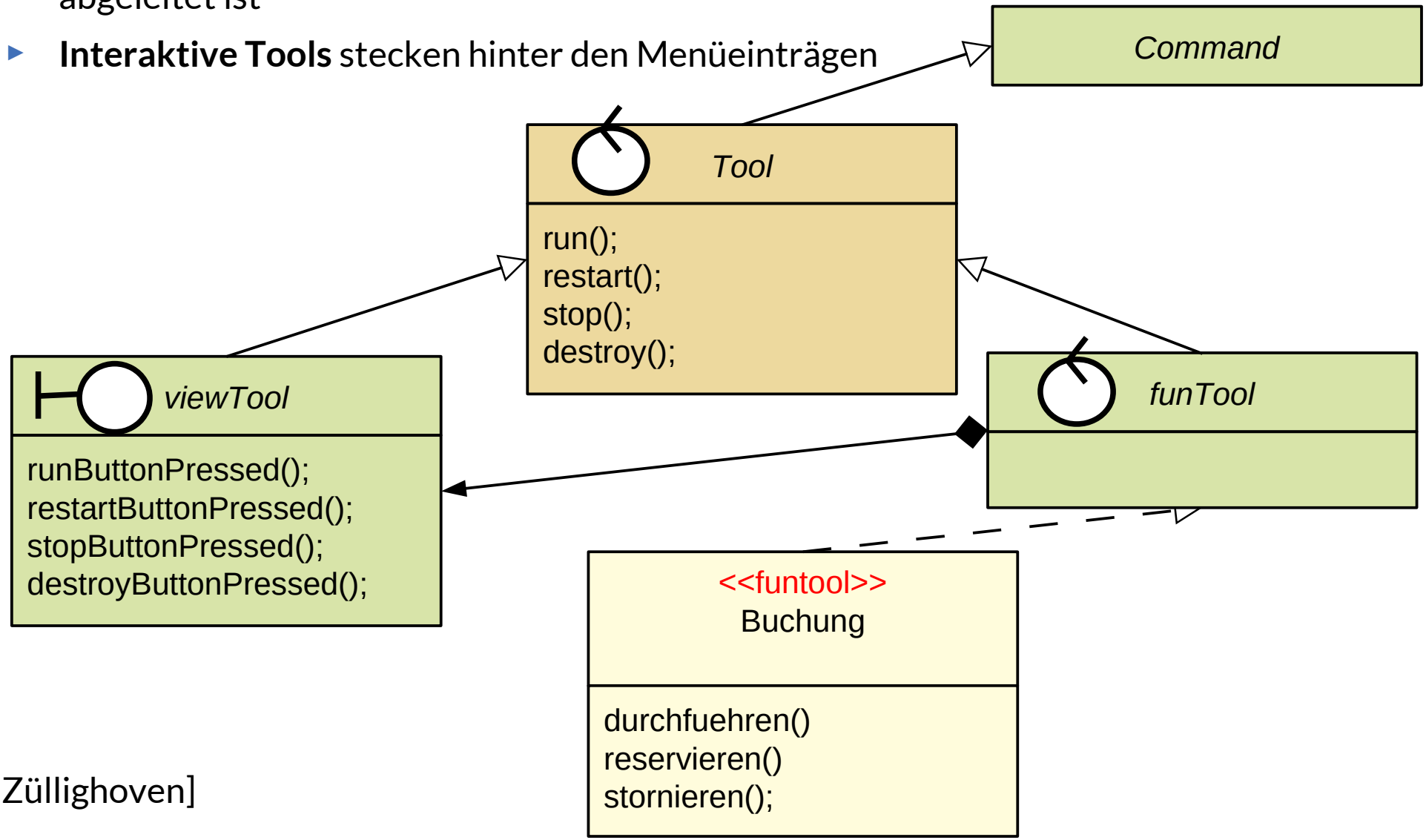
Ressourcen-Klassen und -Schnittstellen

- **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

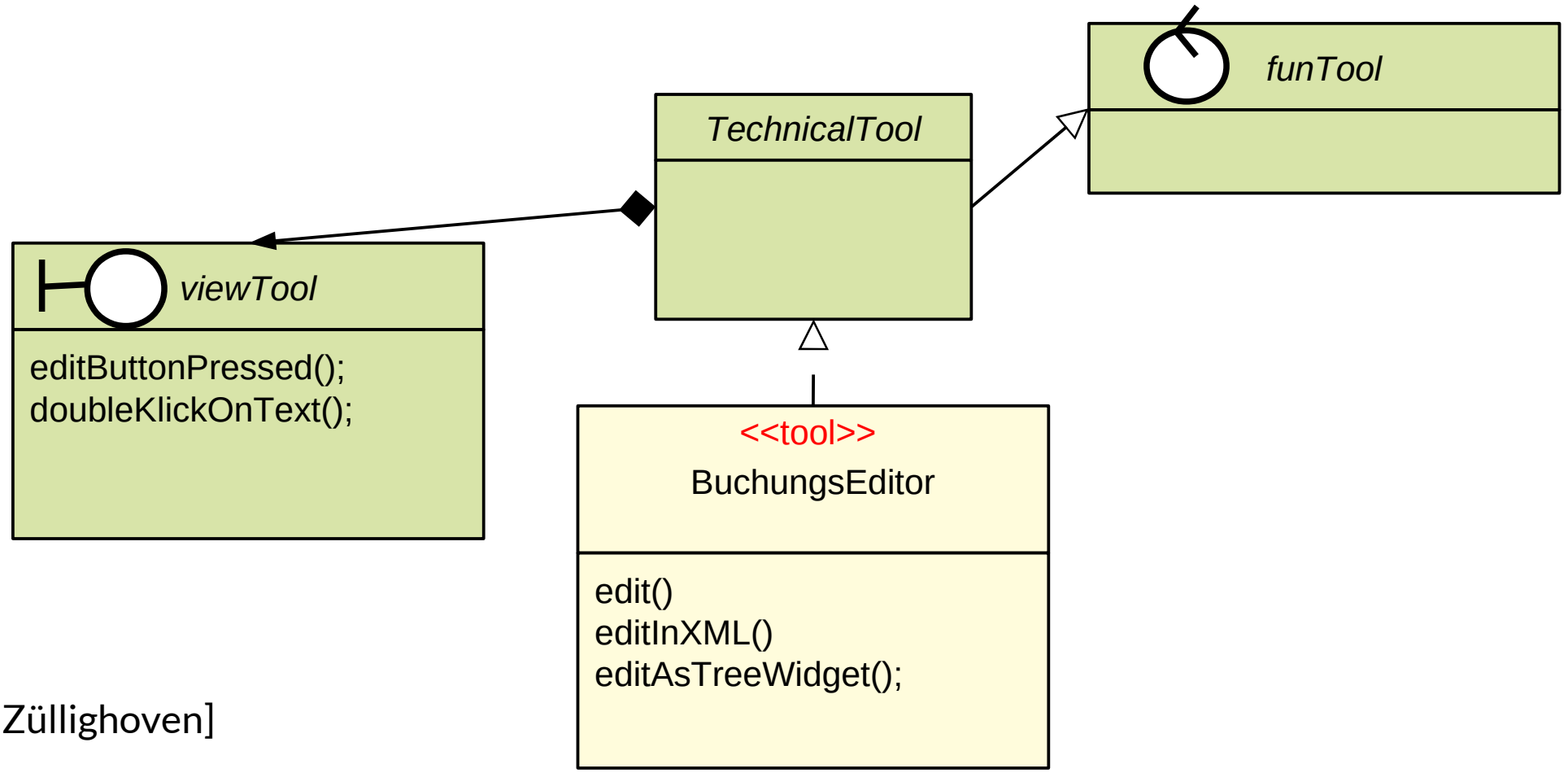
- Toolobjekte haben einen interaktiven Teil (viewTool, boundary, view) und einen ausführenden, funktionalen Teil (funTool, "model"), der aus dem Command-Pattern abgeleitet ist
- **Interaktive Tools** stecken hinter den Menüeinträgen



[Züllighoven]

Technische Tool-Klassen und -Schnittstellen

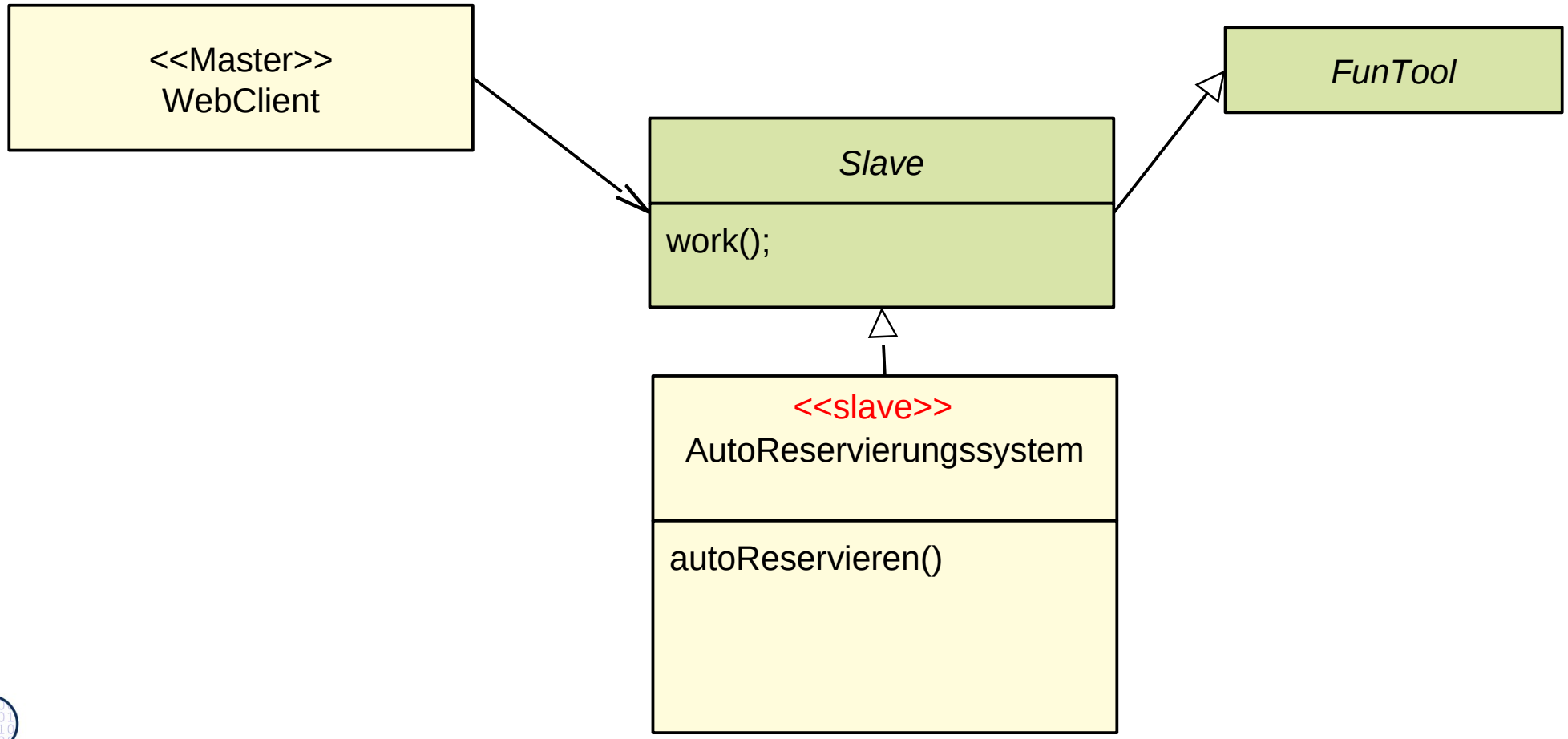
- **Technische Tools** sind funktionale Tools, die eine technische Funktionalität tragen, die nicht anwendungsunspezifisch ist
  - Bsp.: Editor, Lister, Inspektor, Browser, Verschlüsseler, Komprimierer, Optimierer
- Technische Tools verwalten das Material und bilden eine C-Teilschicht direkt über der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



[Züllighoven]

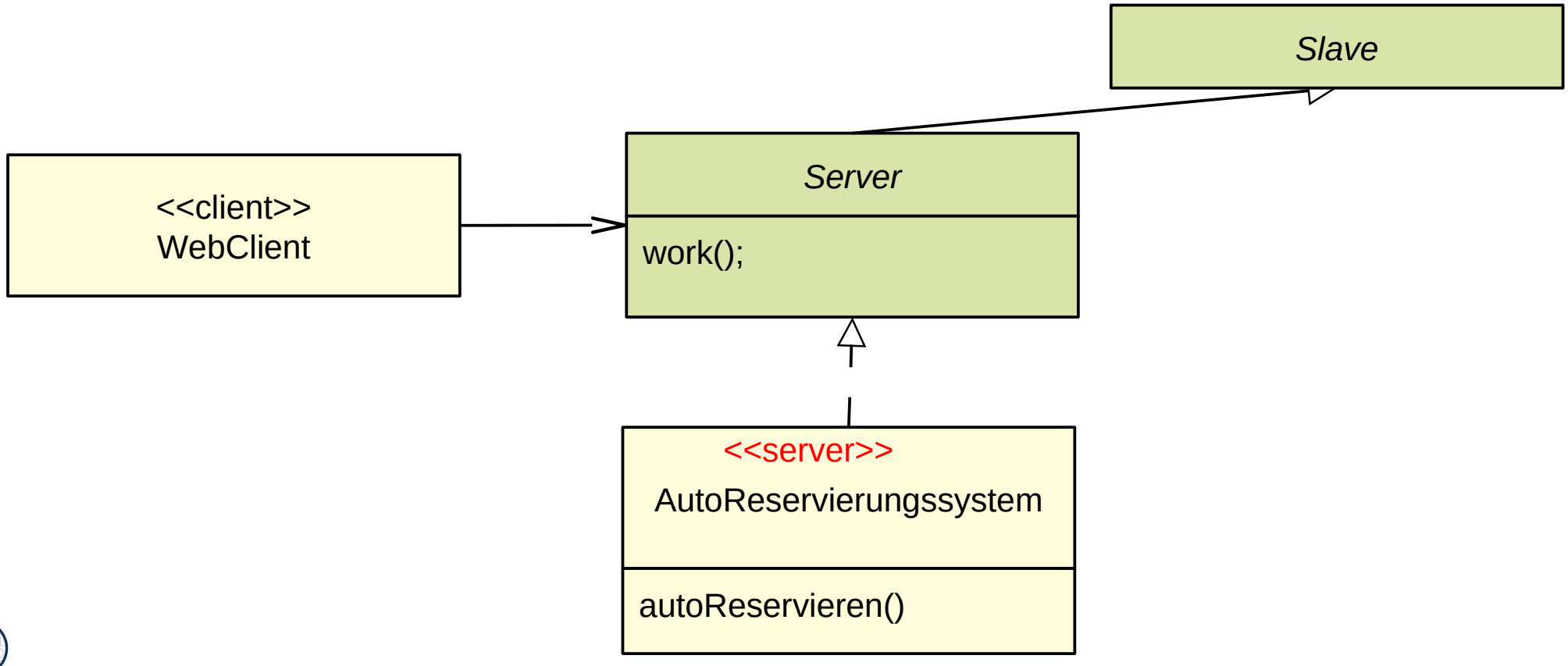
Slave-Klassen und -Schnittstellen

- 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)



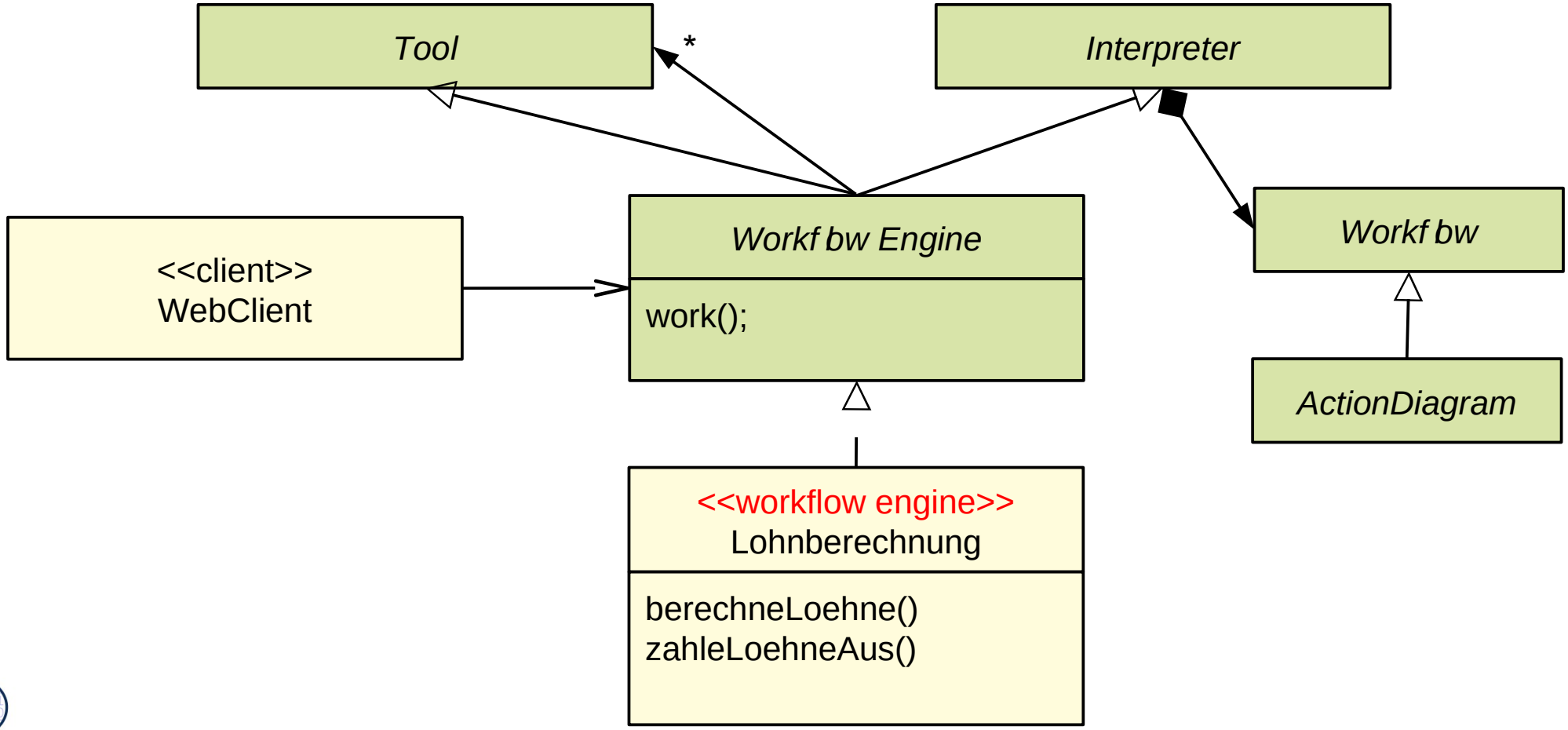
Server-Klassen und -Schnittstellen

- 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

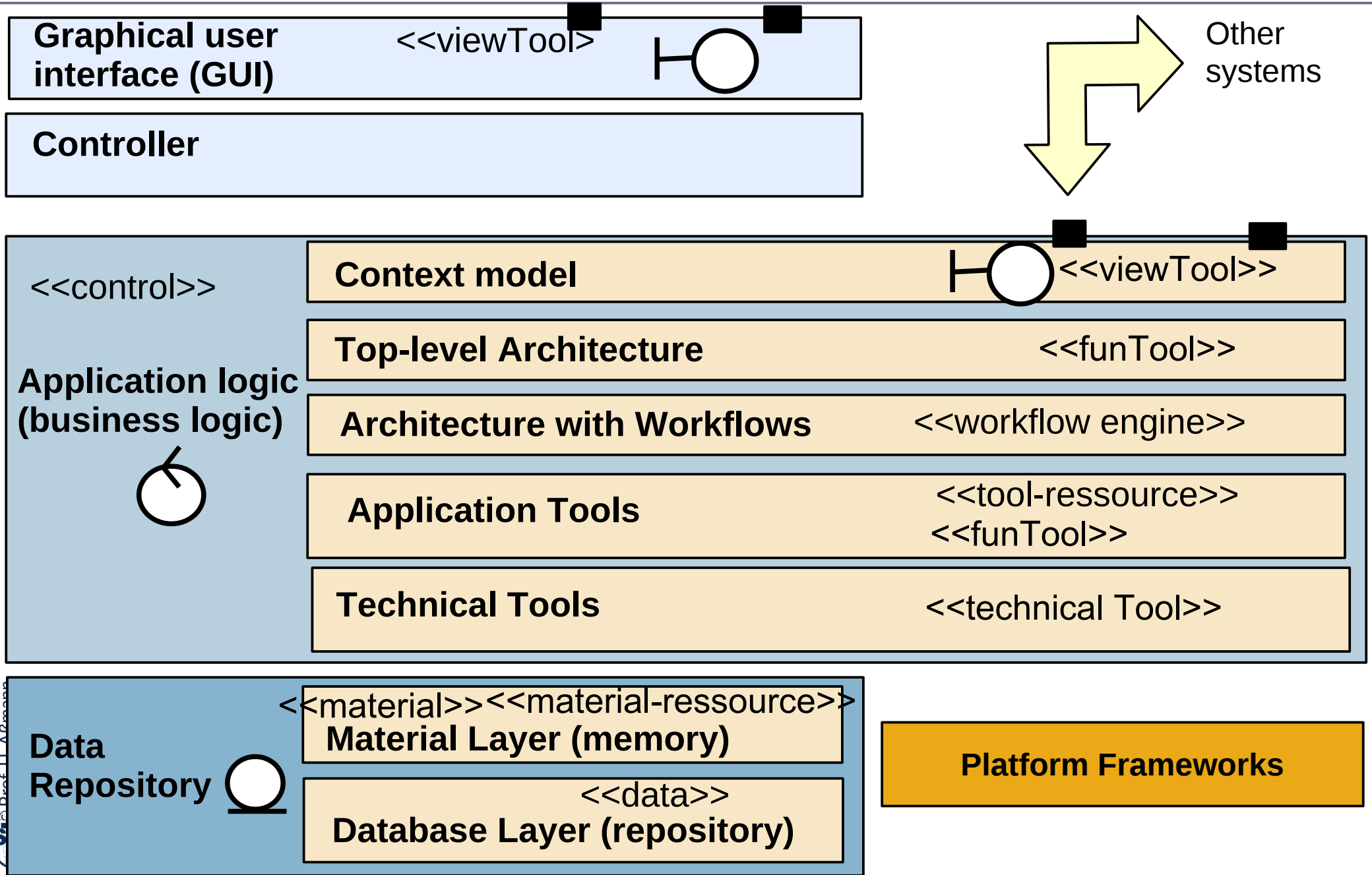
- Eine Workflow-Engine ist ein funktionales Tool, das einen komplexen Arbeitsablauf (Workflow) abarbeitet (interakiv oder batch) und andere Tools ansteuert
  - Workflow-Engines rufen andere Tools auf, setzen also auf ihnen auf
- Workflows können durch Aktionsdiagramme (Aktivitätendiagramm, Statechart,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



Schichten und TAM-Klassifikation

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

Q7': Verfeinerte BCE-Schichtung eines Systems mit TAM



42.2 Plattformanpassung mit Plattformkonnektoren

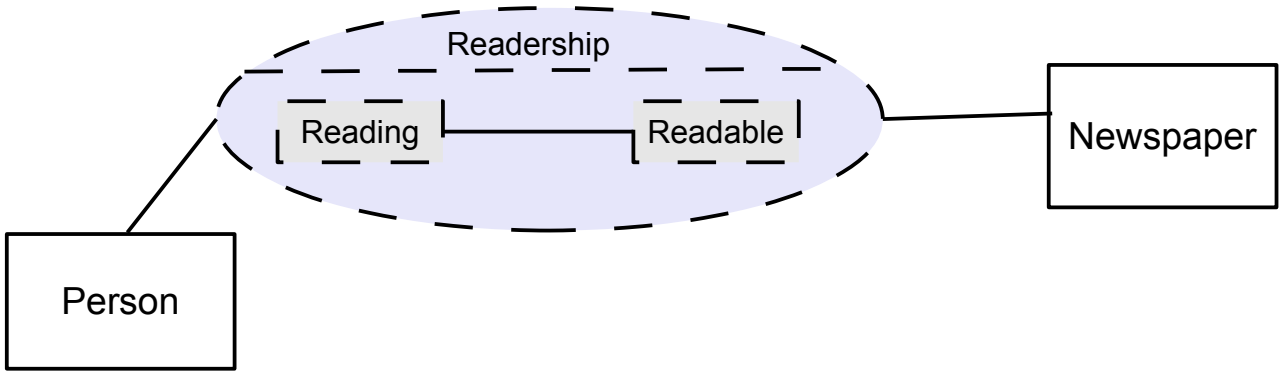
Verfeinerungsbeispiel für Objektanreicherung in der Analyse

.. Verfeinerung durch Integration von Unterobjekten..

Teile und Rollen

Objektanreicherung in der Analyse

- **Objekt-Anreicherung (Object fattening) durch Unterobjekte, die von Konnektoren angelagert werden**
- 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

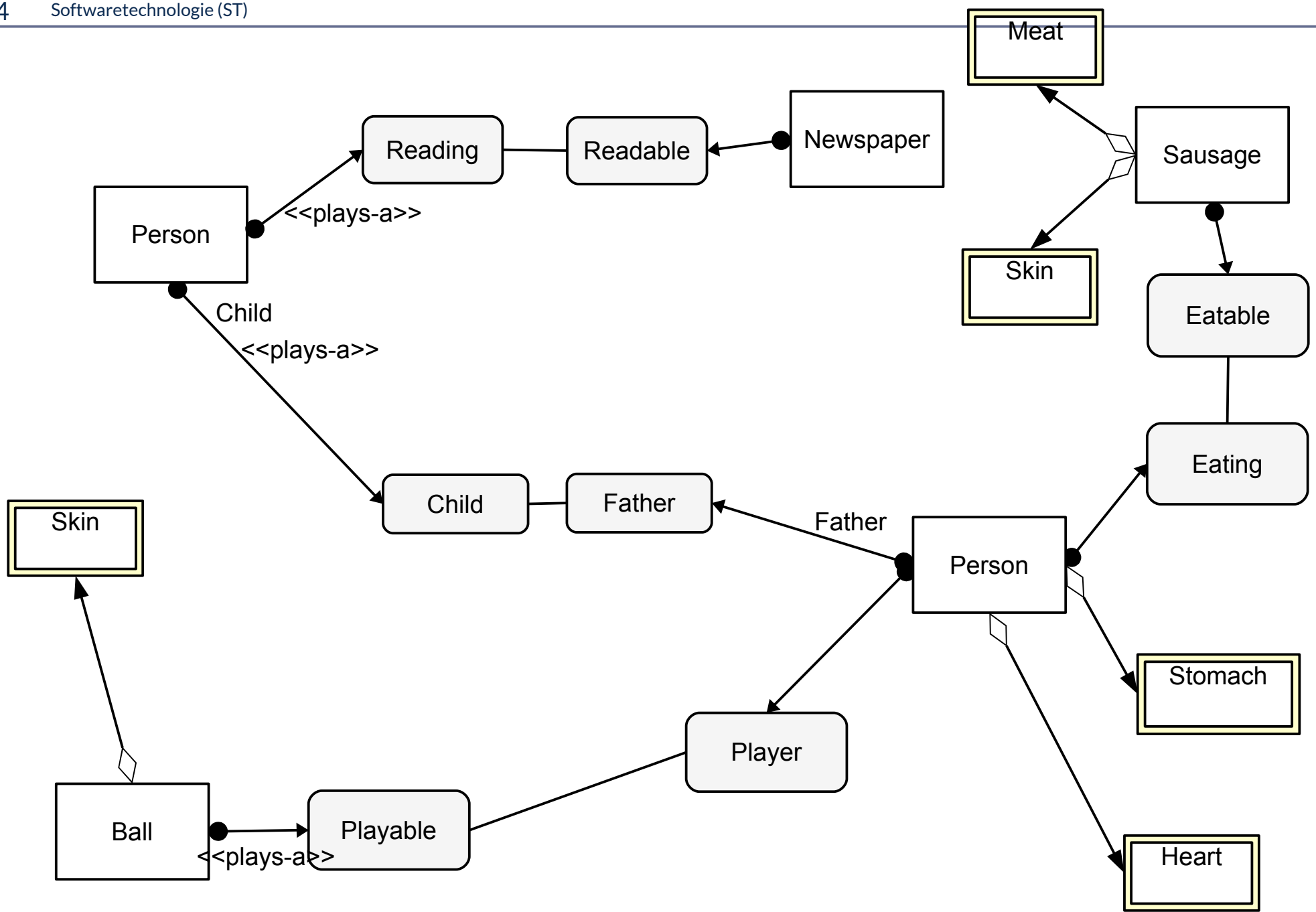


Wdh. Eine **Rolle** ist ein Unterobjekt, das auf ein anderes Objekt bezogen ist.

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

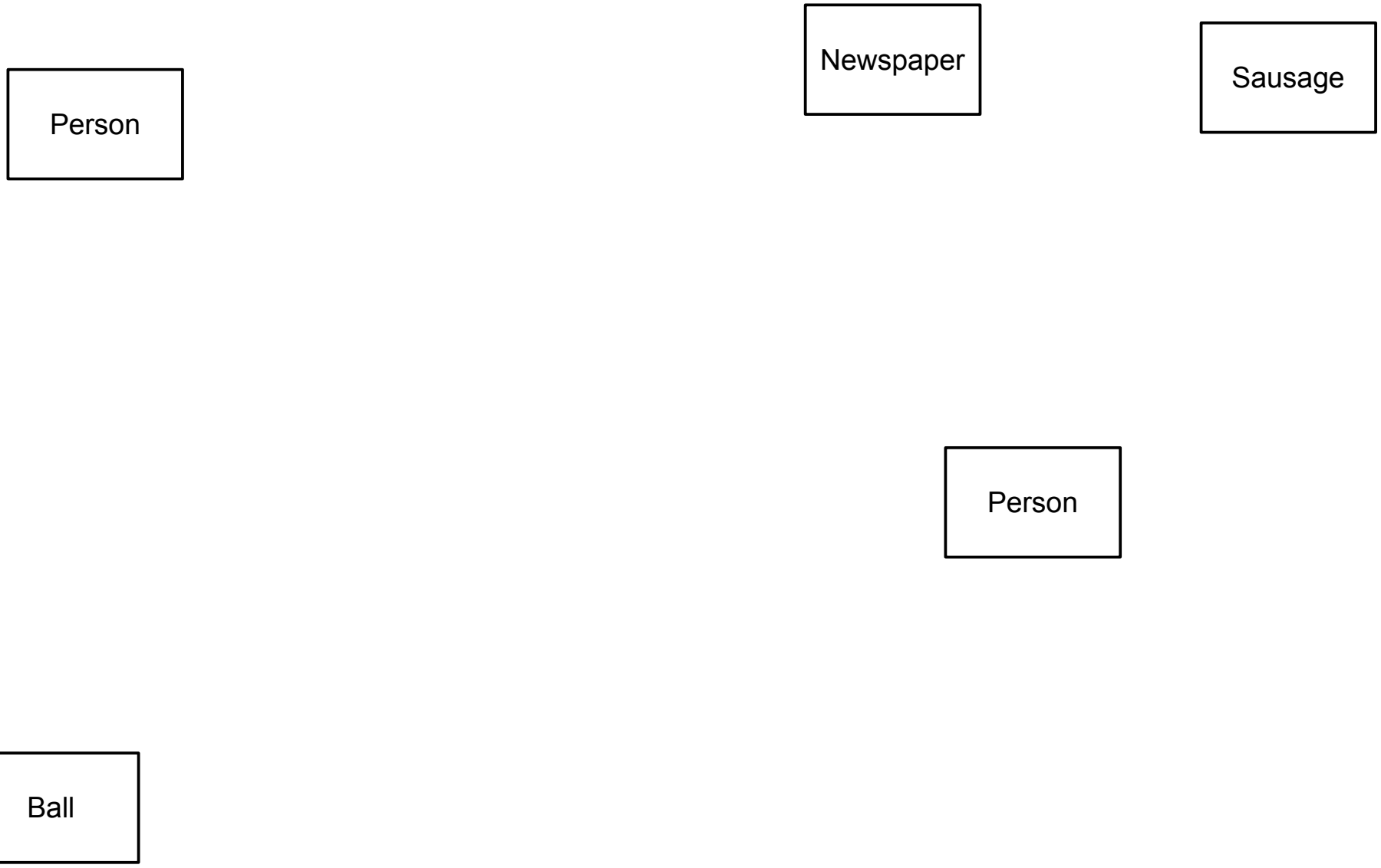
- Gründe:
  - Der objekt-orientierte Software-Entwicklungsprozess startet mit einer Simulation der realen Welt durch Objekte, die zu Systemobjekten erweitert werden und dabei durch technische Informationen angereichert werden müssen

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



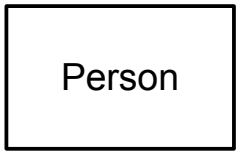
Mit Verfeinerung durch Integration von Unterobjekten (Objektanreicherung, Object Fattening)

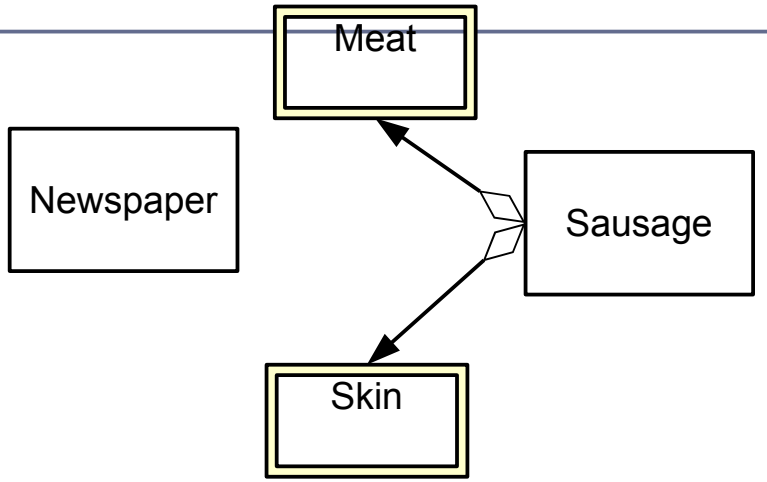
- Rohzustand: Identifikation der natürlichen Typen (in dem Domänenmodell)

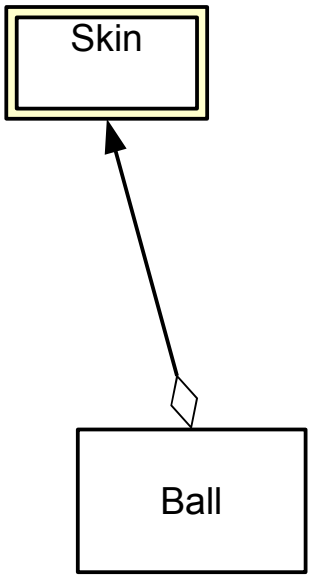


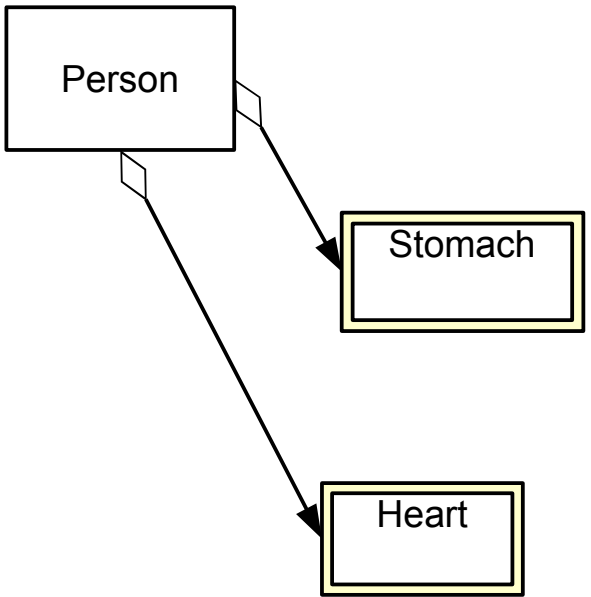
Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 1: Teile-Verfeinerung



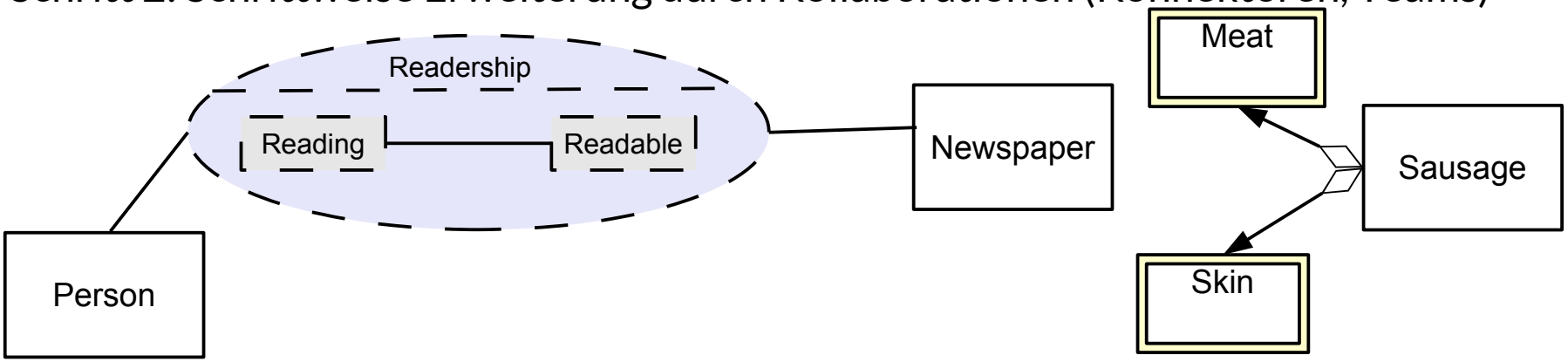


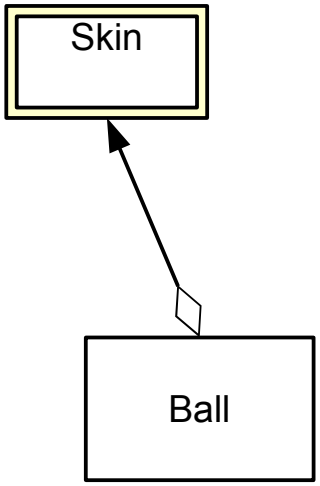


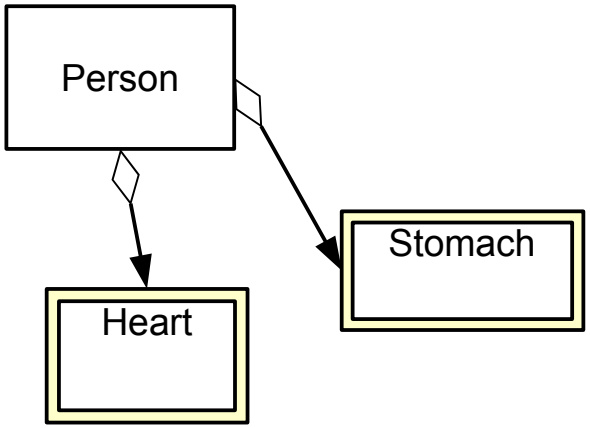


Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

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

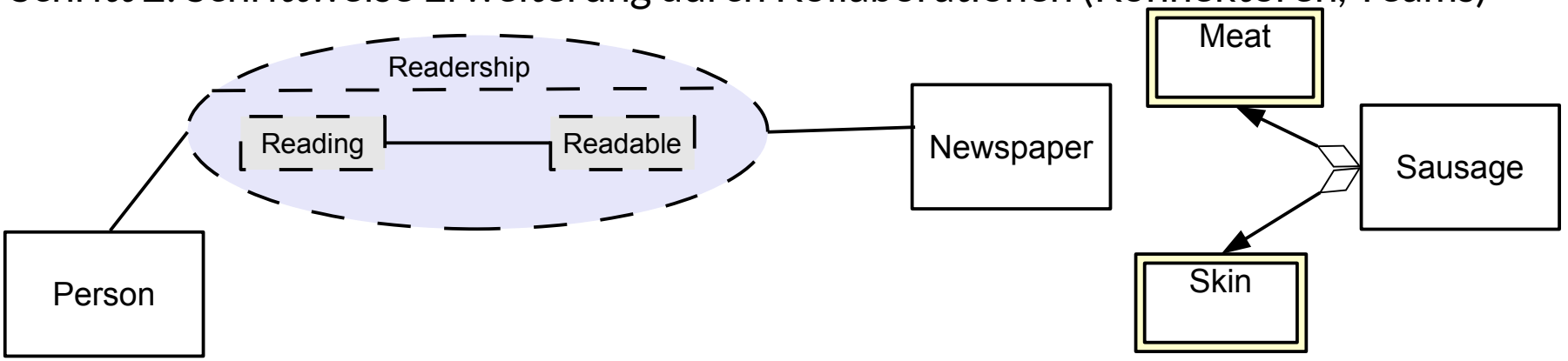


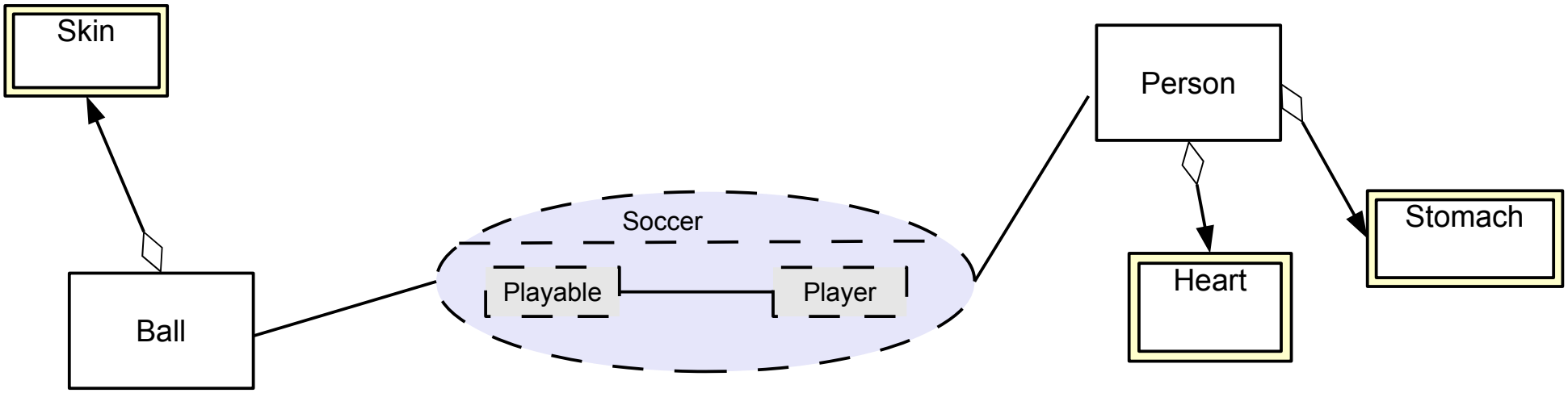




Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

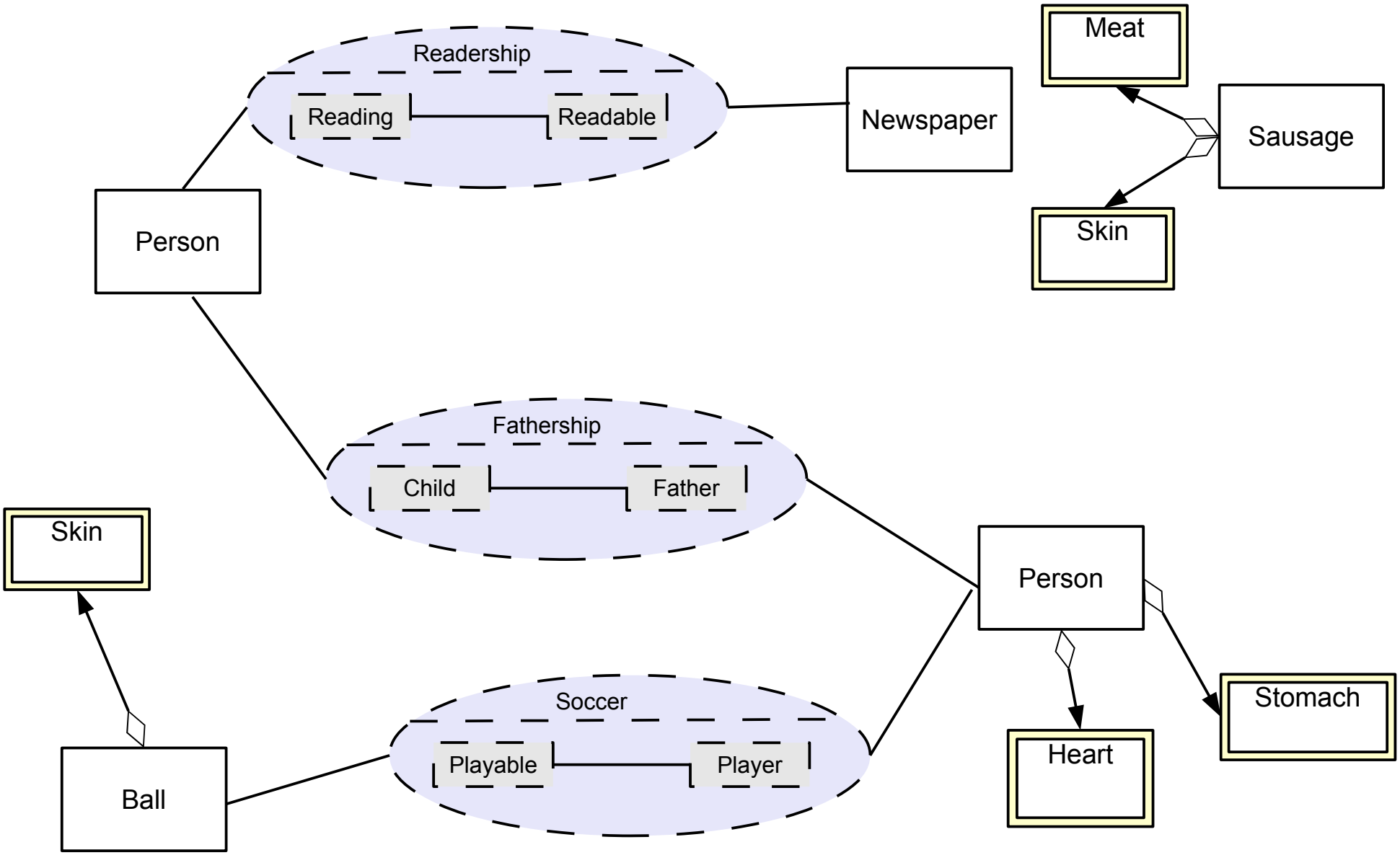
- Schritt 2: Schrittweise Erweiterung durch Kollaborationen (Konnektoren, Teams)



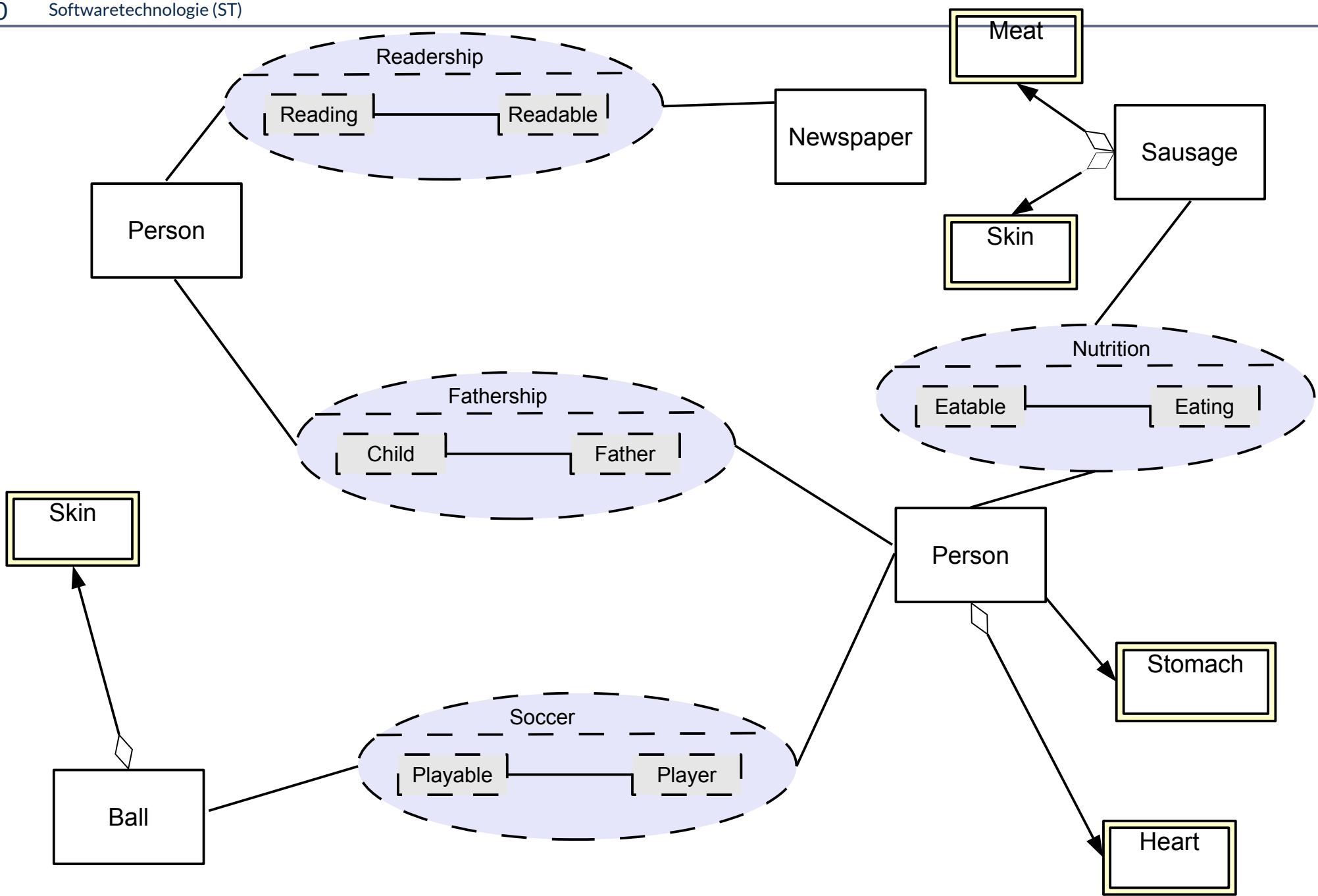


Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 2: final: alle Kollaborationen



Personen-Analysemodell – Angereichert durch Einziehen von querschneidenden Kollaborationen

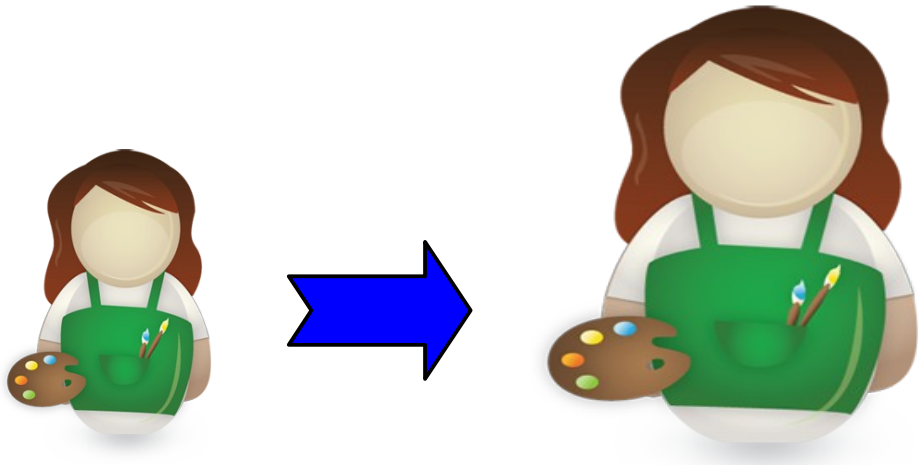


42.3 Anpassung von Material-Objekten an Plattformen
Objektanreicherung mit Plattforminformation (Querschneidende Verfeinerung für Plattformen)

.. 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

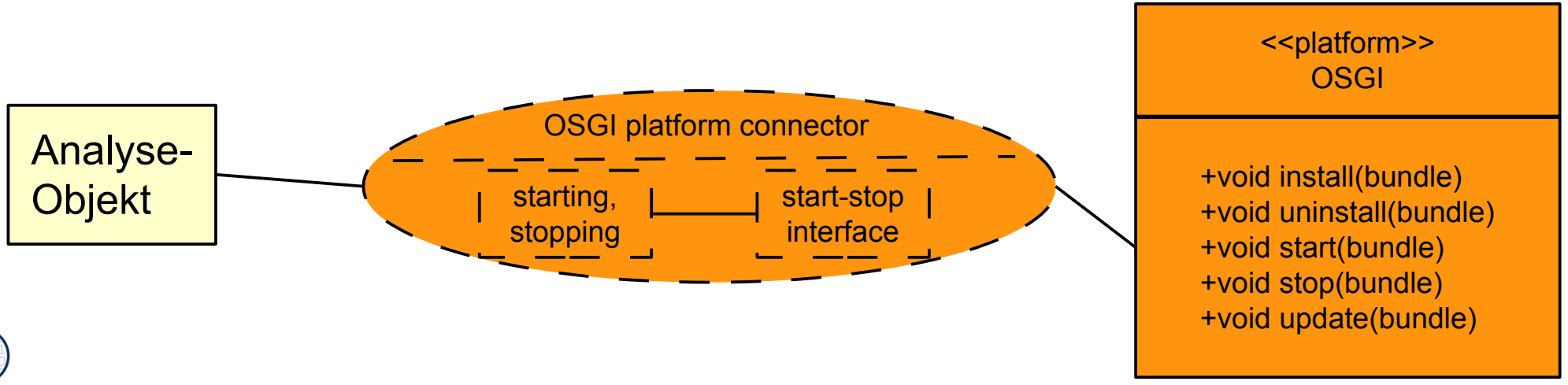


Plattformanreicherung – Weitere Schritte im Entwurf

- Bei Entwurfsobjekten kommt **Plattformanreicherung** hinzu:
  - Identifiziere jede **Plattform**
    - Bibliotheken und Frameworks, auf denen aufgebaut wird
    - Betriebssystem, Datenbanken, Middleware
  - Finde **Plattform-Konnektoren,** mit fundierten Unterobjekte, 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
- **Realisierung** der Konnektoren und der Integrationsrelation
  - Einfache Implementierung durch Konnektoren oder Entwurfsmuster

Plattformobjekte und Plattform-Konnektoren

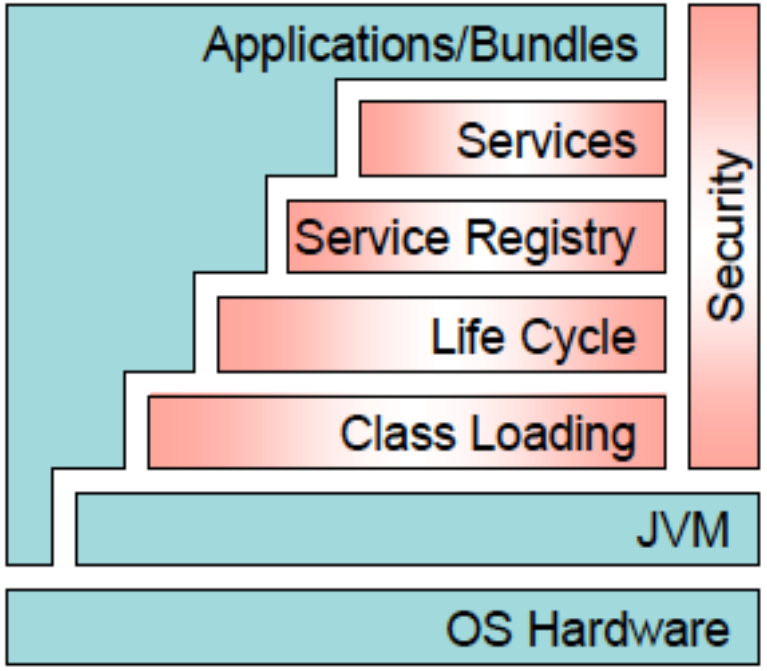
- Ein **Plattformobjekt** ist ein Objekt eines Plattform-Frameworks, das wesentliche Laufzeitfunktionalität bietet und auf die eine Software angepasst werden muss
  - Bietet Schnittstelle an bzgl. bestimmter Funktionalität, z.B. abstrakte Maschine (Interpretierer)
  - Variabel: je nach Maschine, Middleware, 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

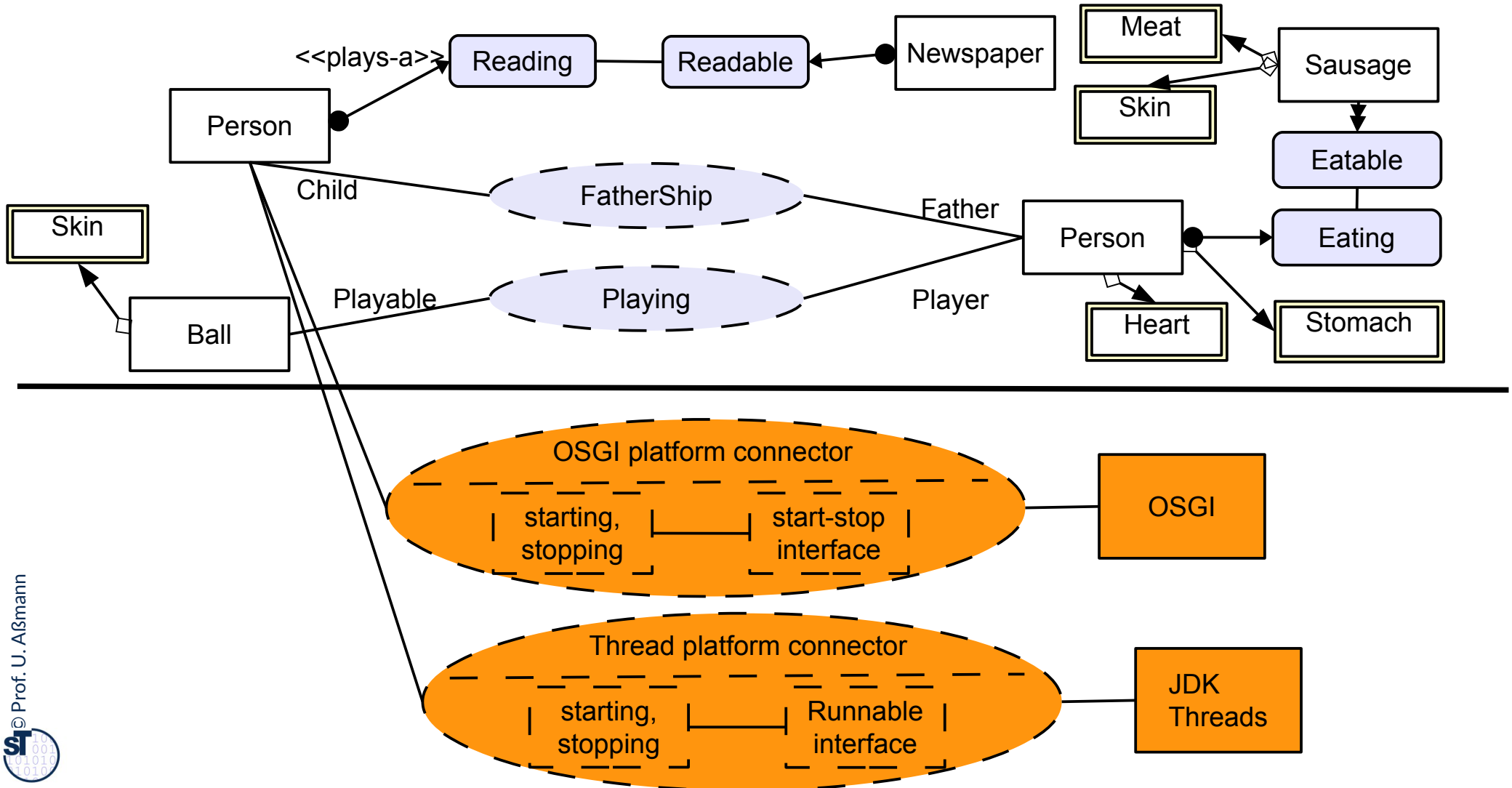
- OSGI bietet 5 Schnittstellen (rot)
  - Klassenlader (für Ersetzung von bundles)
  - Lebenszyklus (life cycle) von *bundles* (Paketen von Klassen, mit zip gepackt und verschickt)
  - Register (service registry): dient zum Registrieren von Bundles und ihren Zuständen
  - Dienste (services) verschiedener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

- [OSGI Technical White Paper]



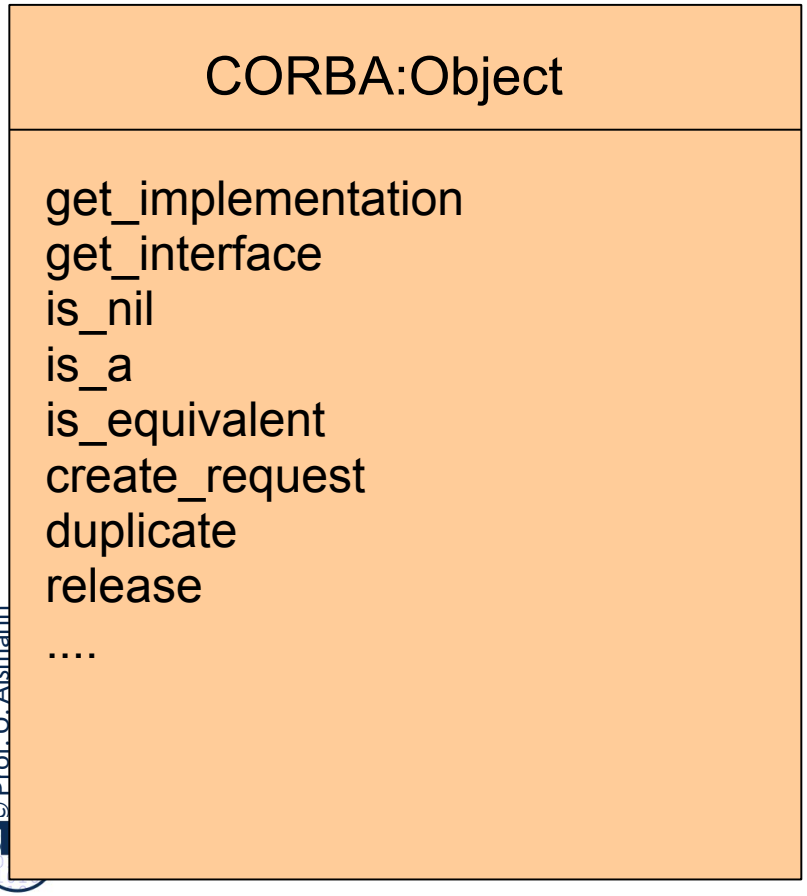
Mit Verfeinerung durch Plattform-Konnektoren (platform fattening)

- 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

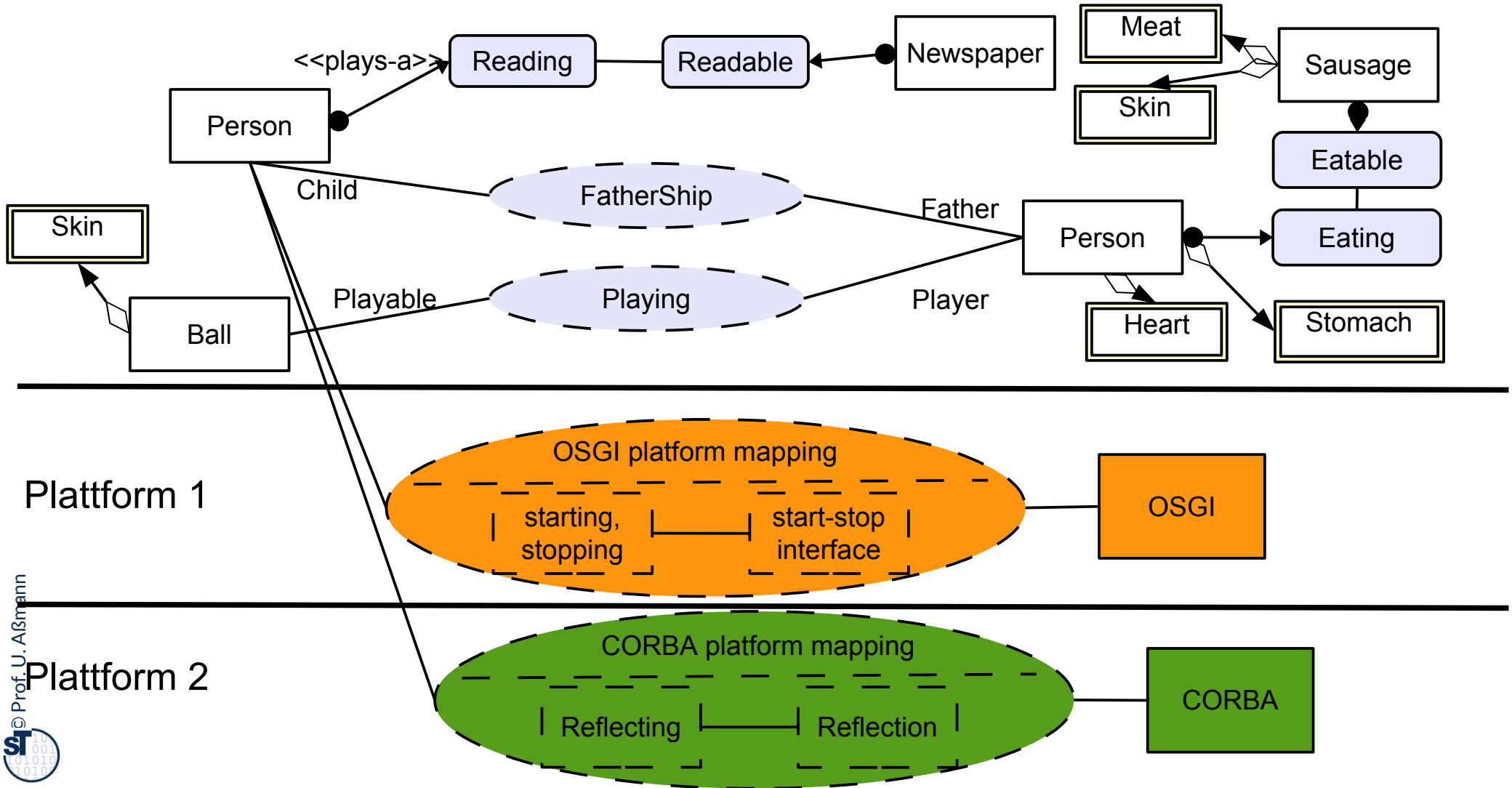
- CORBA bildet eine Komponentenplattform für heterogen programmierte Systeme



- 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.

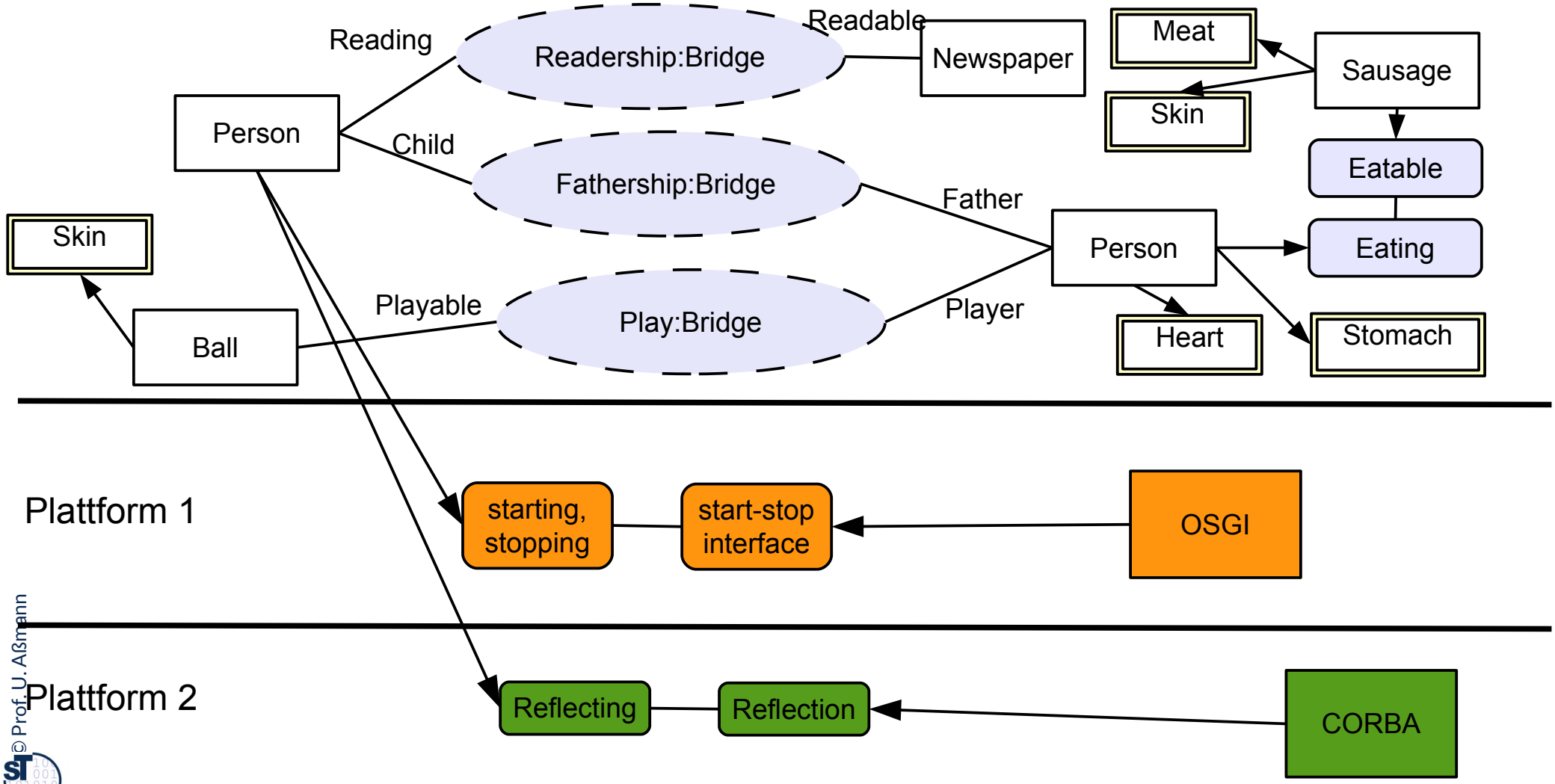
Bei einer Portierung auf eine andere Plattform müssen I.d.R. für Datenhaltung und Anwendungslogik getrennt Plattformkonnektoren entwickelt werden.

42.4 Abbildung der Integrationsrelation auf klassische Programmiersprachen

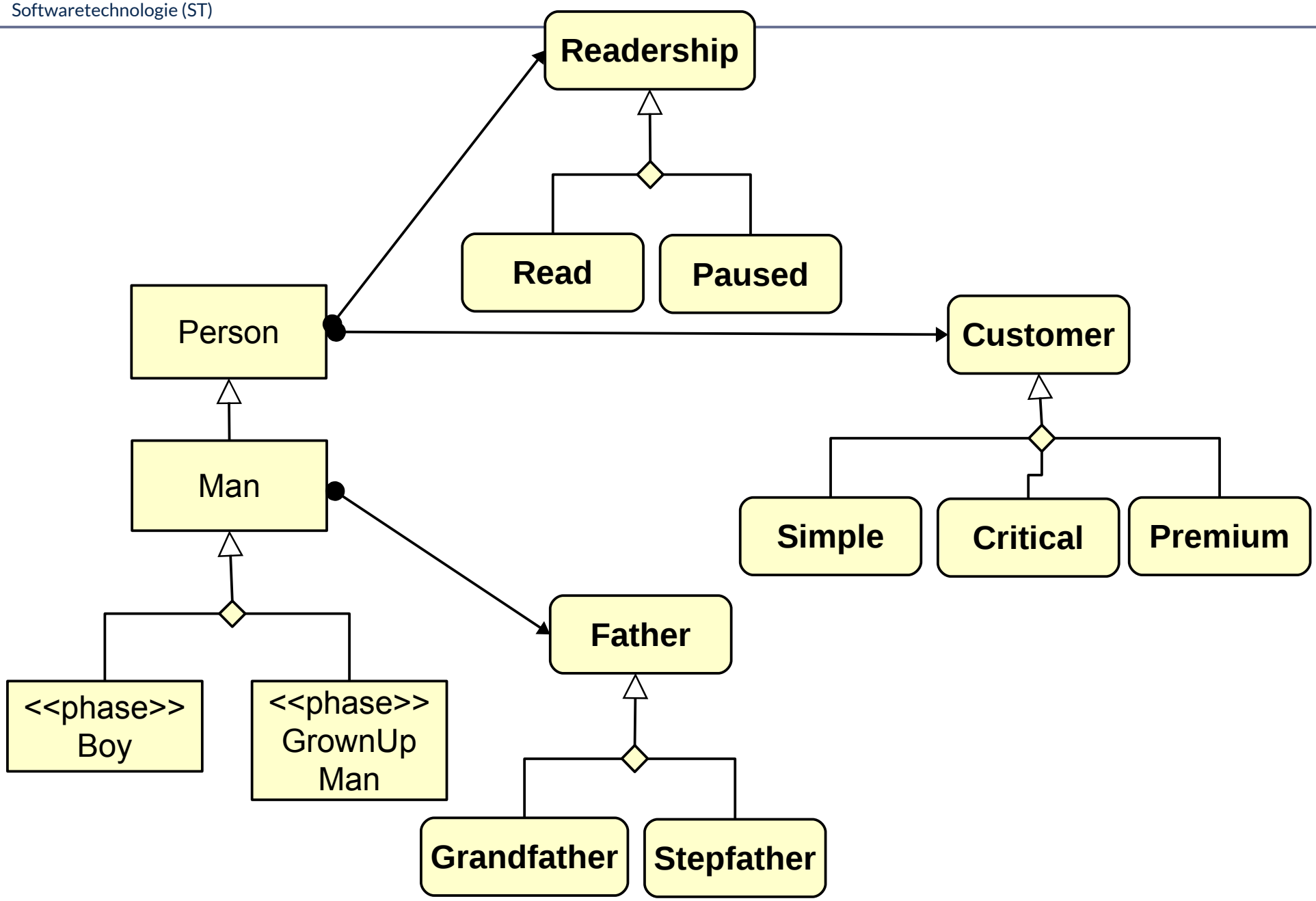
.. in der Implementierung ..

a) Wie bilde ich “plays-a” durch Multi-Bridge (Delegation) ab?

- Ersetze alle “plays”, “mandatory-part”, etc. durch Entwurfsmuster Bridge und Multi-Bridge (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

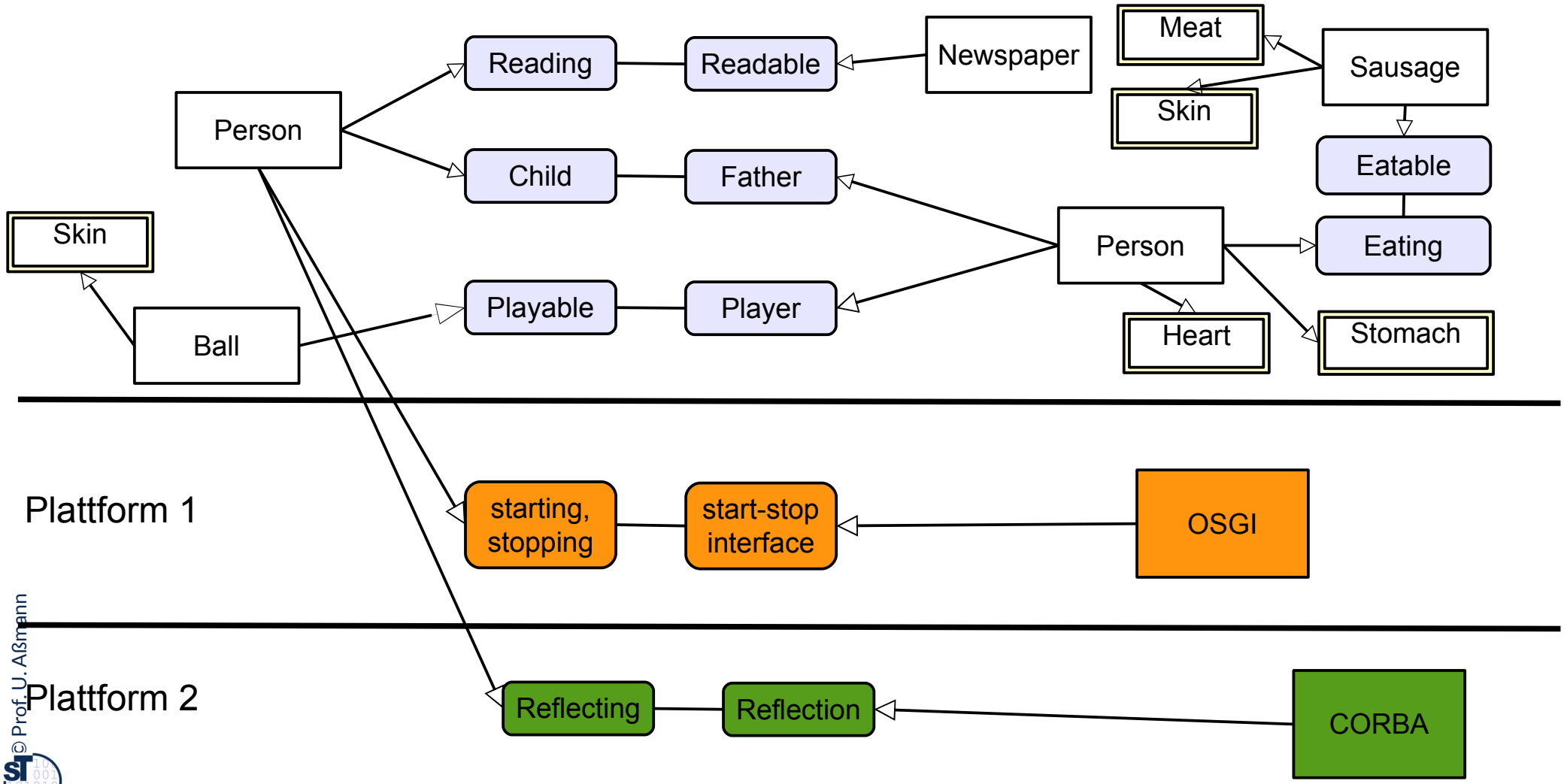


Erinnerung: Realisierung von Rollen mit Multi-Bridge



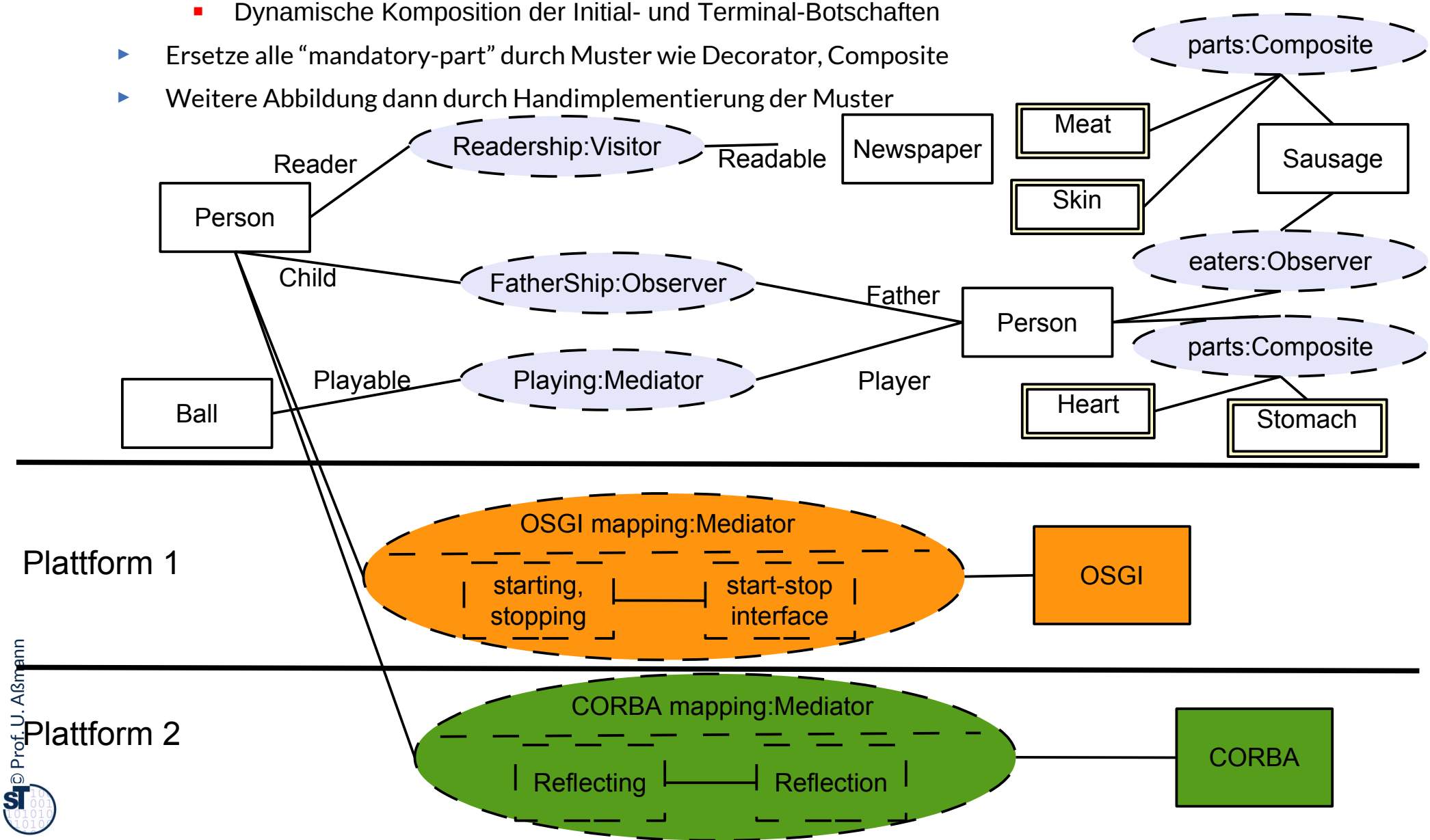
b) Wie bilde ich “integrates” durch Vererbung ab?

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



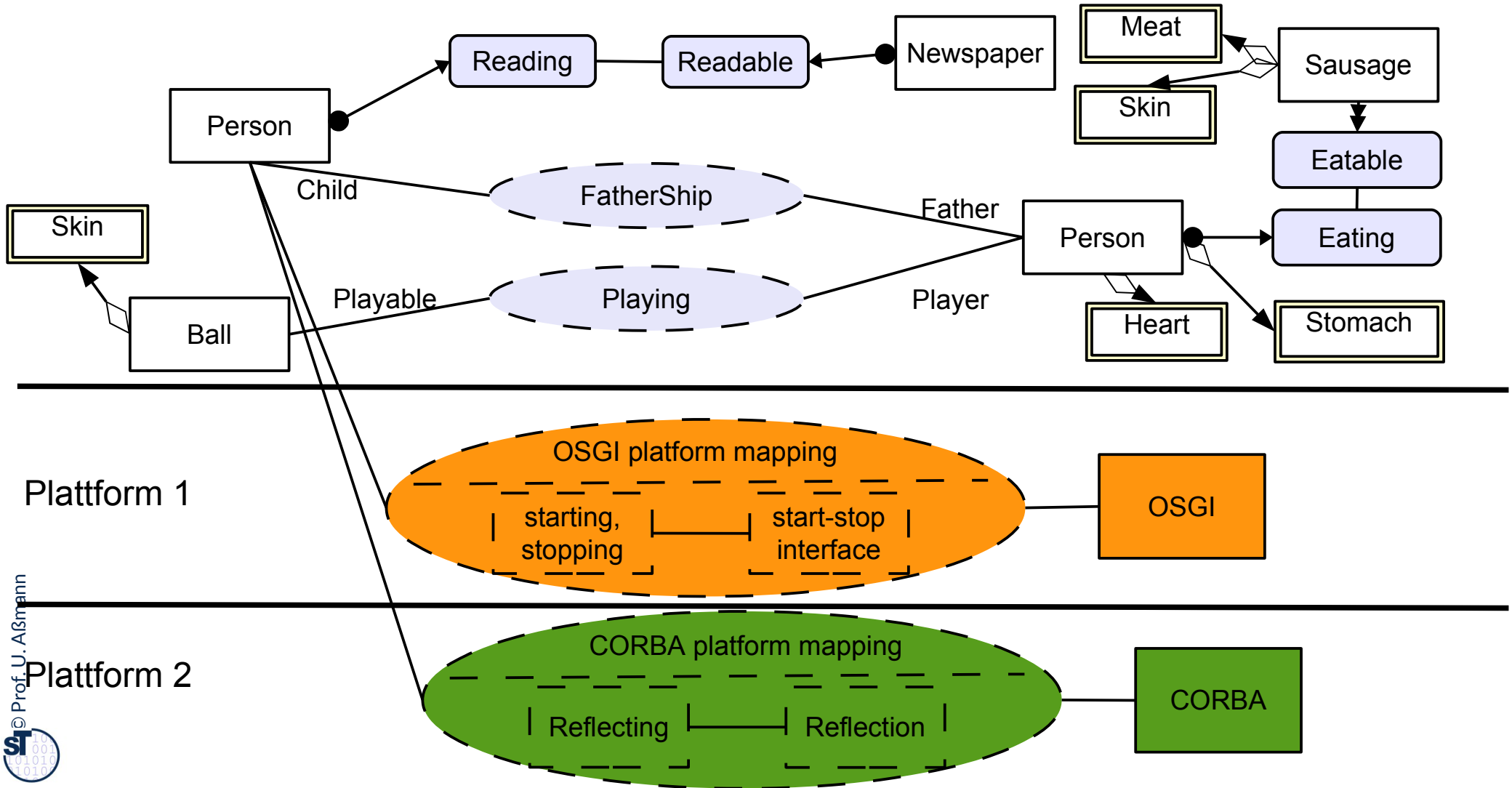
c) Wie bilde ich “plays-a” durch Implementierungsmuster ab?

- Ersetze alle “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



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

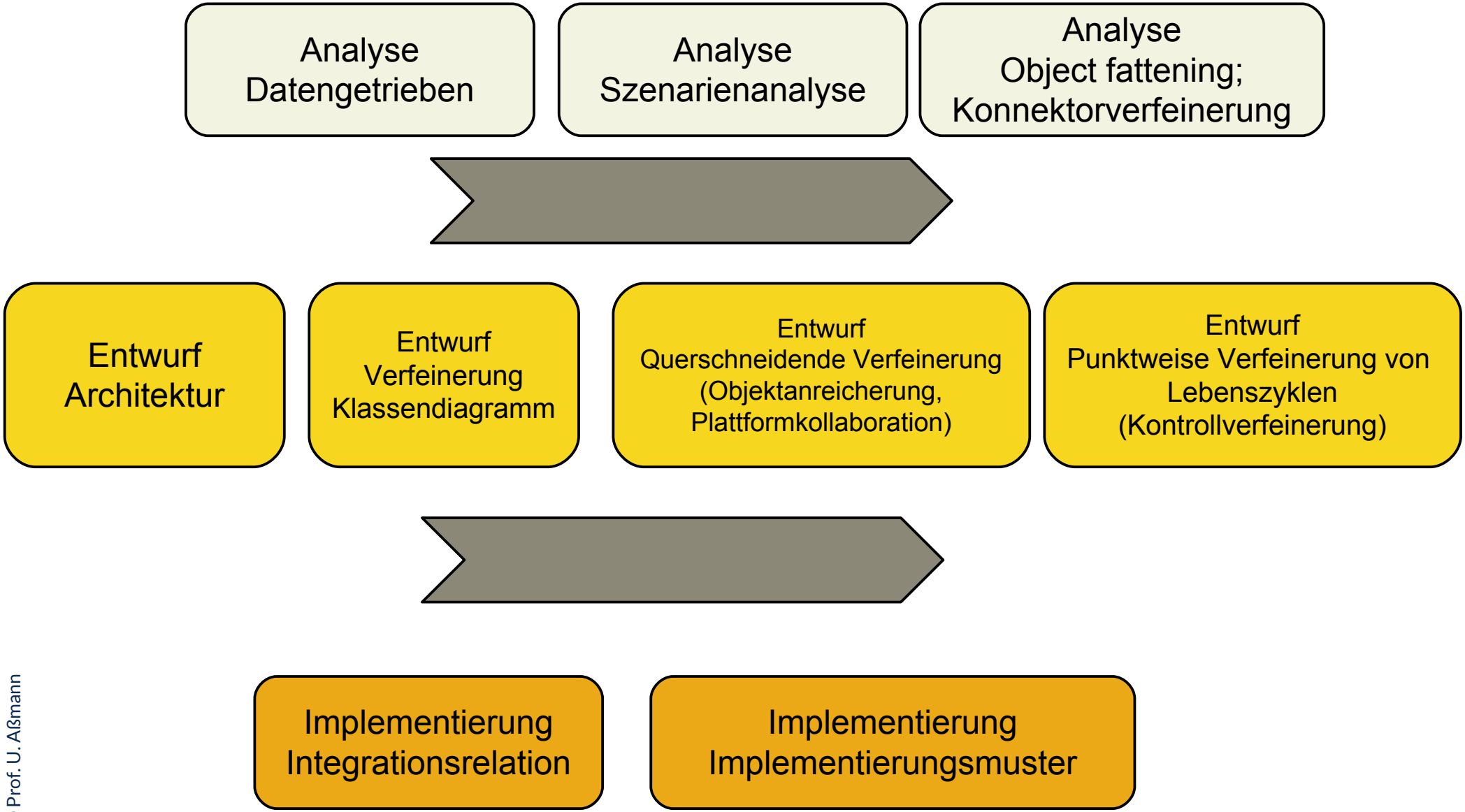
- Kollaborationen/Konnektoren und die “plays-a”-Relation können verschieden auf eine Programmiersprache abgebildet werden
  1) Durch Rollensprachen wie ObjectTeams.org; dann liegt die Abbildung im Übersetzer



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

42.5 Gesamtbild der Verfeinerung



The End

- Gehen Sie im Geiste zurück auf die Szenarienanalyse aus Teil III. Nutzen Sie die TAM-Stereotypen, um die dort analysierten Klassen in eine Schicht einzuordnen. Was werden Tools? Was Materials? Was Workflows?
- Warum ist eine Trennung von Tool und Material auf verschiedene Objekte in verschiedenen Schichten sinnvoll?
- Wie kann man die Materialien testen? Wie die Tools?
- Wieso muss man Ressourcen-Materialen zuteilen und sperren?
- Wann entsteht aus einer Bridge einer Collaboration eine Multi-Bridge?
- Wieso braucht man Plattformkonnektoren?
- Wieso will man eine Software auf andere Plattformen portieren?
- Geben Sie zwei Realisierungen für eine UML-Kollaboration mit “plays-a”-Links an. Vergleichen Sie deren Vor- und Nachteile

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