Details zur Klausur auf OPAL

- Probeklausur: Montag, 02.08.2021 08:00
- Prüfungsklausur: Montag, 09.08.2021 09:30
- OPAL Seite:
  - https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/29581049860/CourseNode/162510836055861400 3

© Prof. U. Aßmann

Einschreibung in das Software-Projekt-Praktikum WS 2021/22

- Einschreibung erfolgt für ALLE Studenten (auch für ISTler) ab sofort über OPAL
  - https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/29581049860/CourseNode/16251956122640570003

- Einschreibung **befristet bis 04. August 2021**
  - Praktikum Softwaretechnologie - **Intern** (Standardfall, Tutor ist Kunde)
  - Praktikum Softwaretechnologie – **Extern** (Firma oder andere Institution als realer Kunde)

- 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 04. August 2021 nicht eingeschrieben hat, kann wahrscheinlich nicht teilnehmen, sich jedoch auf der Nachrückliste einschreiben!

© Prof. U. Aßmann

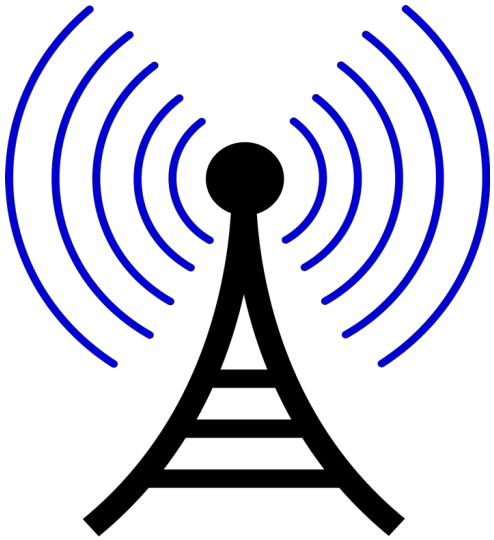
42. Die Softwarearchitektur der Anwendungslogik im Detail

Weitere Tipps zur Gestaltung der Anwendungslogik- und Datenhaltungs-Schicht mit Tool-Tool-, TAM- und Plattform-Kollaborationen

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

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) **Tool- und TAM-Kollaborationen**
    2) **Plattform-Kollaborationen: Verfeinerung mit querschneidender Objektanreicherung**
4) 43: Architektur interaktiver Systeme
5) 44: Punktweise Verfeinerung von Lebenszyklen
    - Verfeinerung von verschiedenen Steuerungsmaschinen



© Prof. U. Aßmann

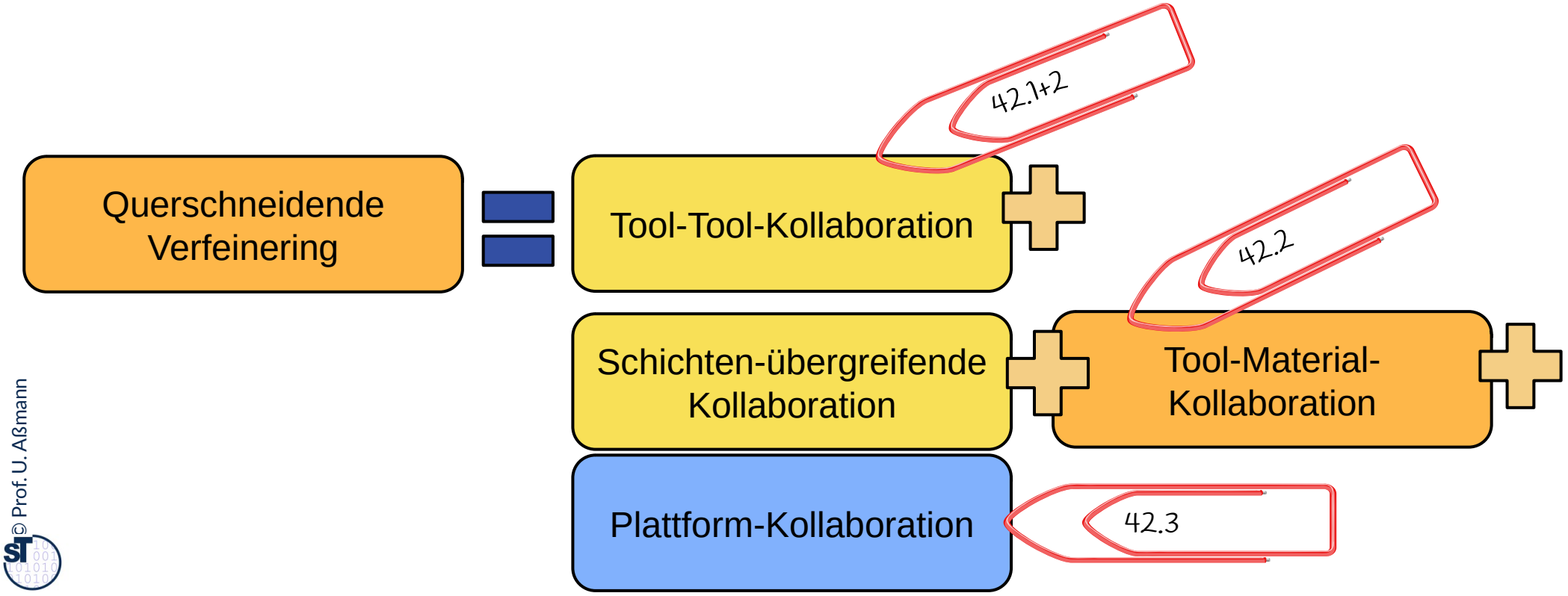
Literatur

- 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.

© Prof. U. Aßmann

Entwurf vom Feldherrnhügel aus gesehen

- Wer Architektur und querschneidende Verfeinerung beherrscht, beherrscht den Entwurf.



© Prof. U. Aßmann

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 oder verzögert synchron) aufgerufen?

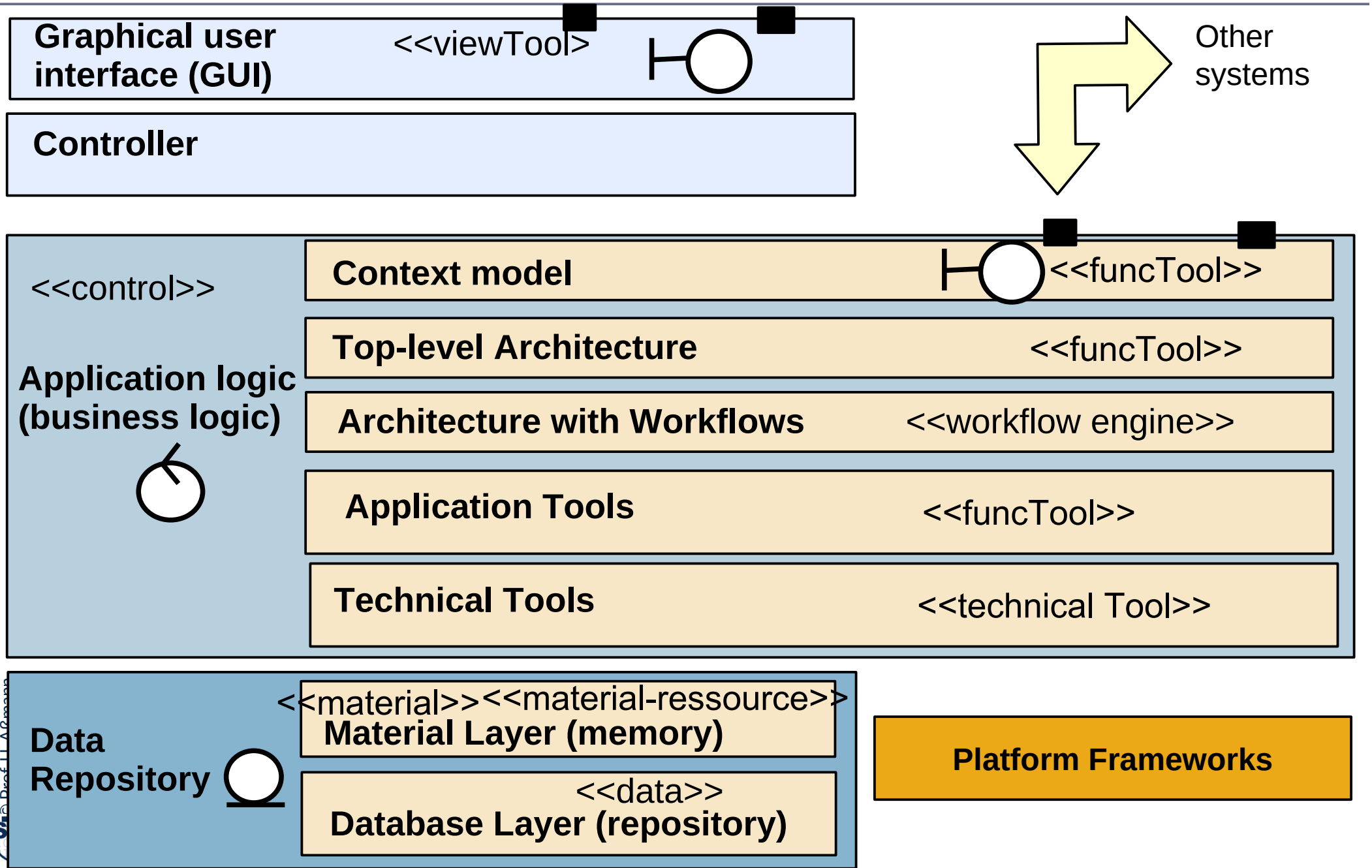
Was ist aktiv, was ist passiv?

Was muss belegt werden (was kann nicht geteilt benutzt werden)?

Welche Klasse wird in welche Schicht eingeordnet?

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Vorausschau:
Q8: Verfeinerte BCED-Schichtung eines Systems mit TAM



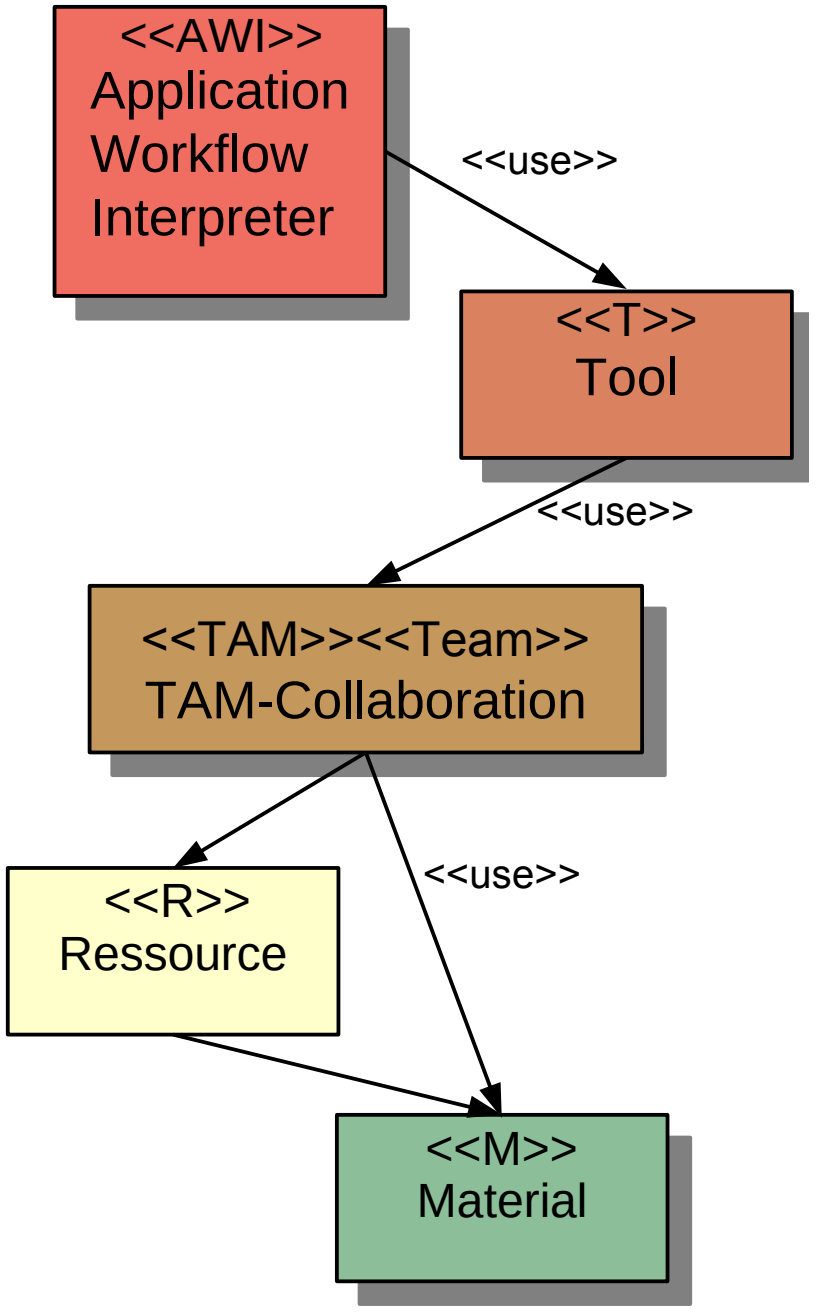
© Prof. U. Aßmann

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, Kommandoobjekte)
2) Ressources (belegbar)
3) Materials (passive Daten, Schicht E und D)
4) TAM-Collaboration
5) Interpreter (für Workflows) koordinieren Tools

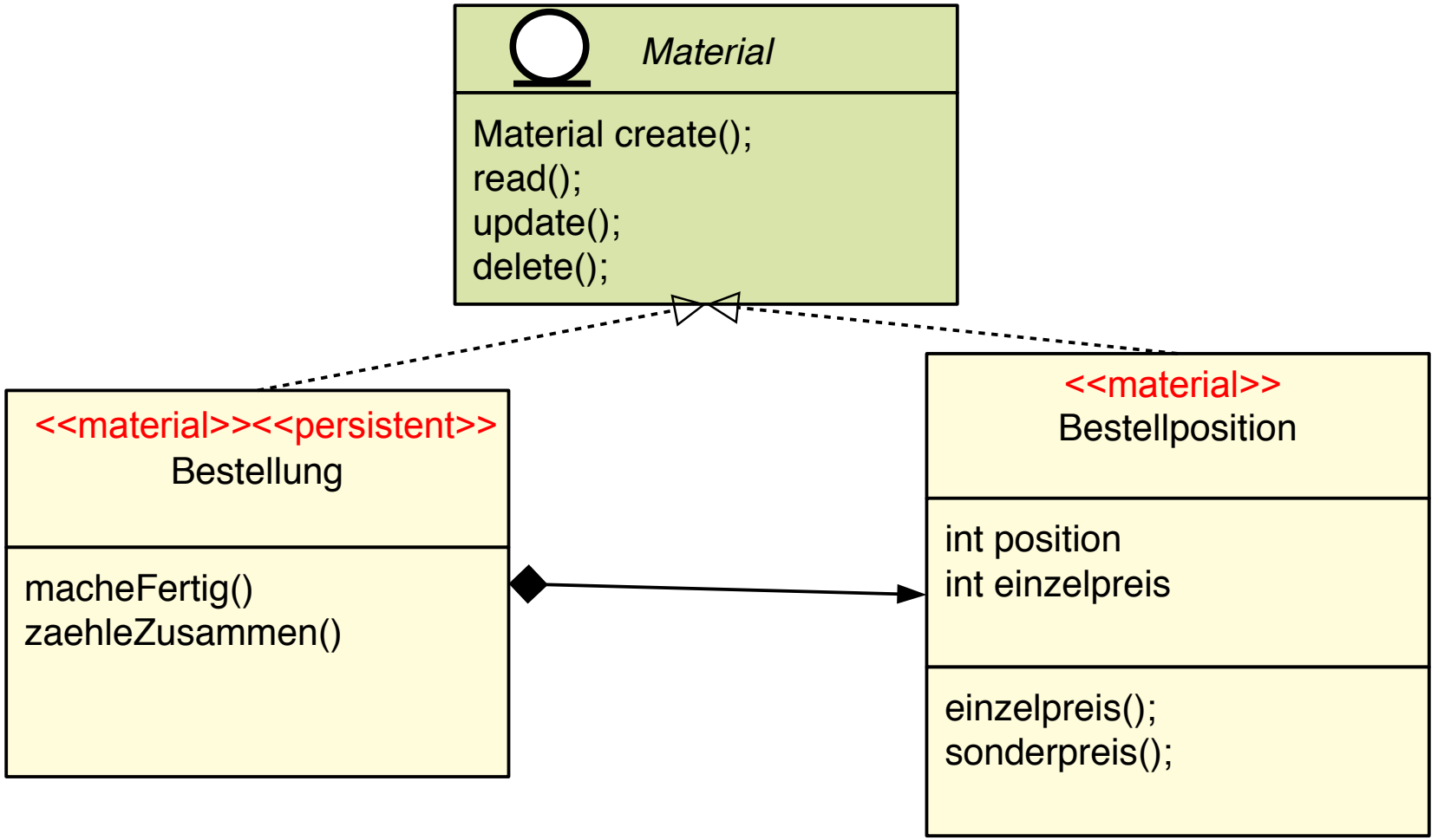
- Klassen, Module, Komponenten, Pakete Kollaborationen, Teams sollten mit diesen Aspekten qualifiziert werden



© Prof. U. Aßmann

Material-Klassen und -Schnittstellen

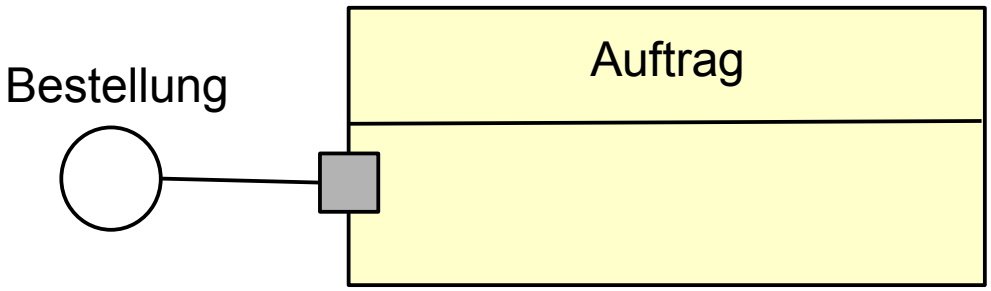
- **Materialobjekte** sind passiv, d.h. werden von außen aufgerufen und geben den Steuerfluss nach außen hin zurück
  - Liegen in der Material- (E) oder auch persistenter Datenablage-Schicht (D)
- Materialobjekte können komposit sein (Muster Composite)
- Materialien folgen der CRUD-Schnittstelle (create,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



© Prof. U. Aßmann

Material-Klassen und -Schnittstellen

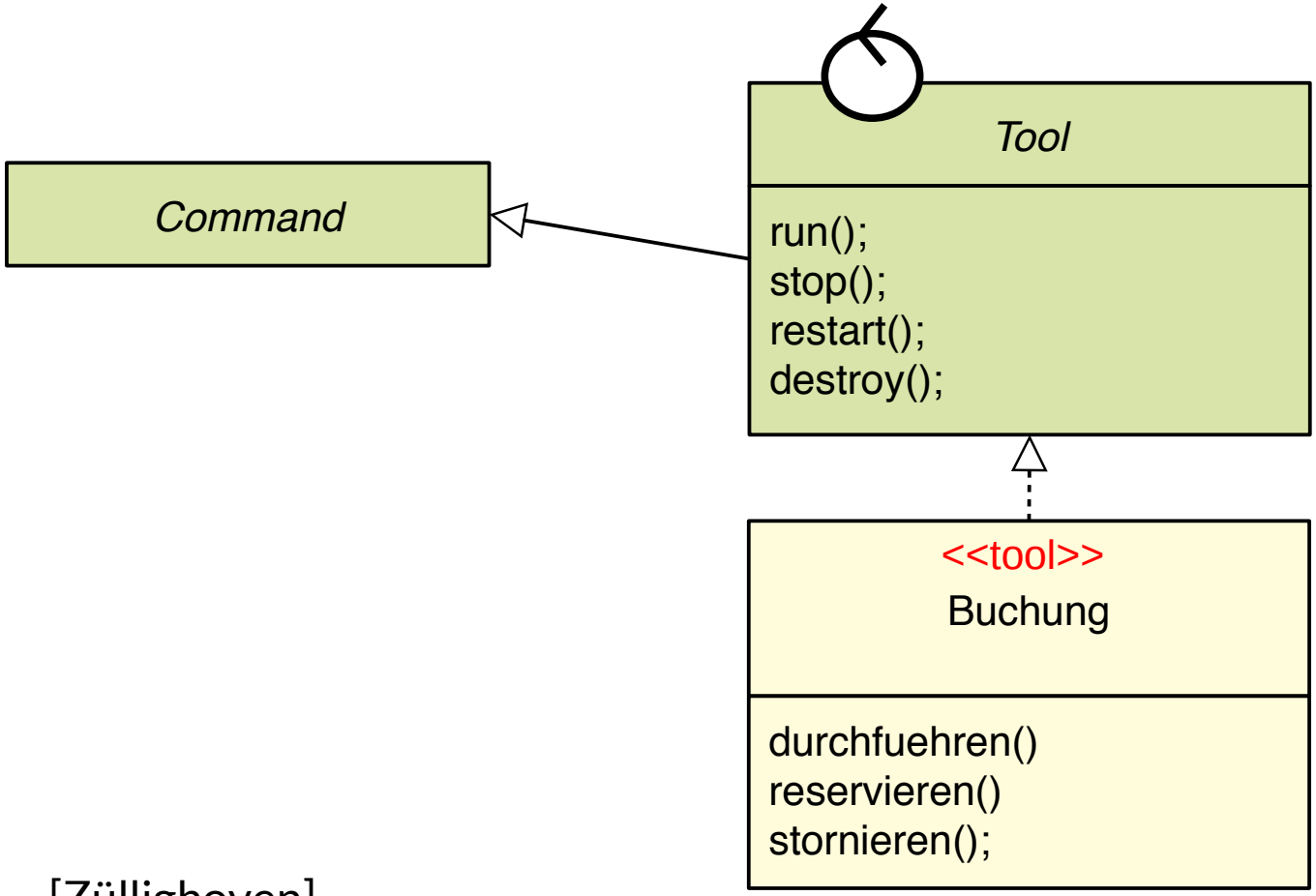
- Materialien können in Port-Schnittstellen von Komponenten auftauchen, z.B. im Kontextmodell



© Prof. U. Aßmann

Tool-Klassen und -Schnittstellen

- Toolobjekte sind I.d.R. *aktiv*, besitzen eigenen Steuerfluss (thread, process)
  - Sie erben vom Entwurfsmuster Command
- Tools liegen in der Regel nicht in der Datenschicht sondern in der GUI (B-Schicht) und der Anwendungslogik (C-Schicht)

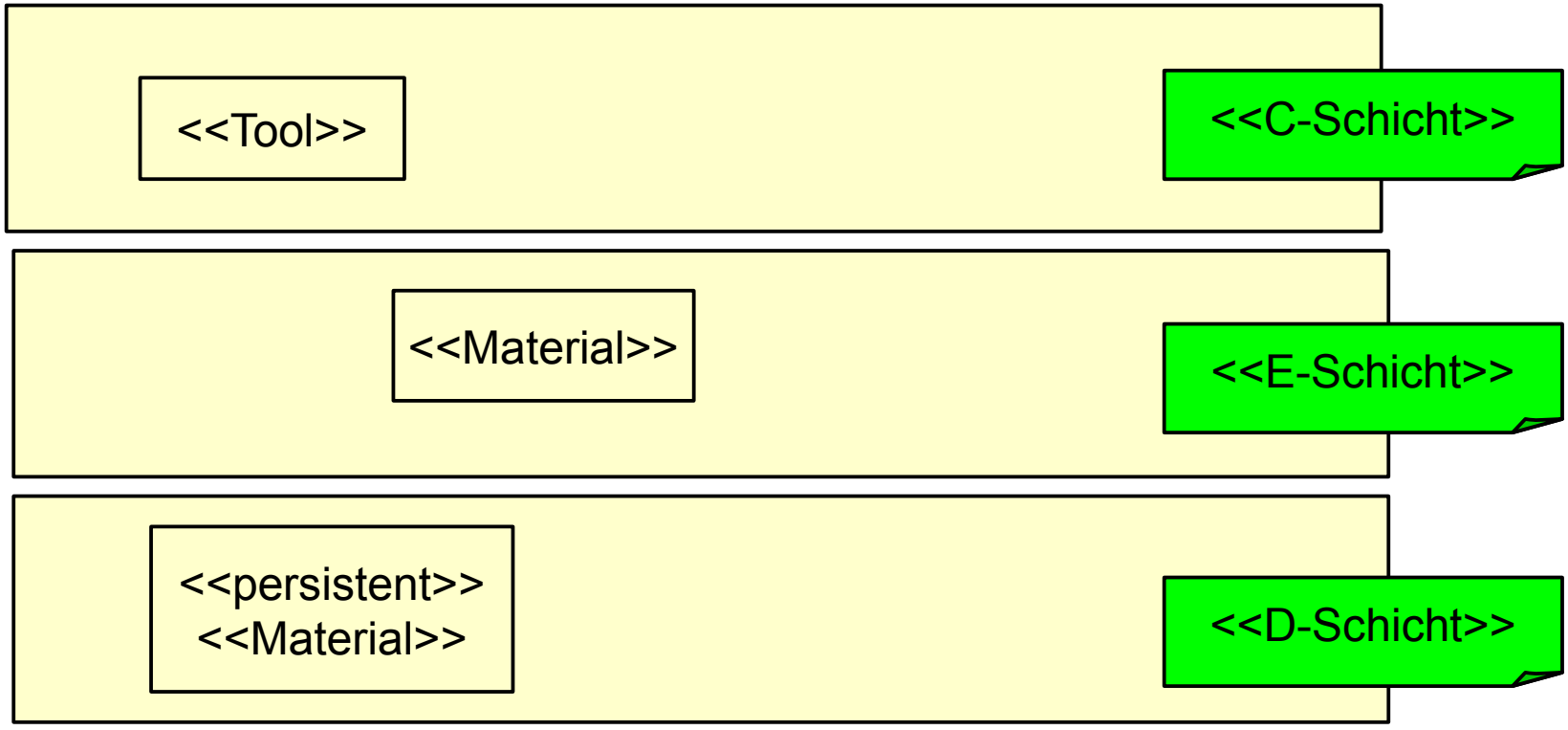


[Züllighoven]

© Prof. U. Aßmann

Das TAM-Separierungsgesetz (Tools-Material-Separation Gesetz)

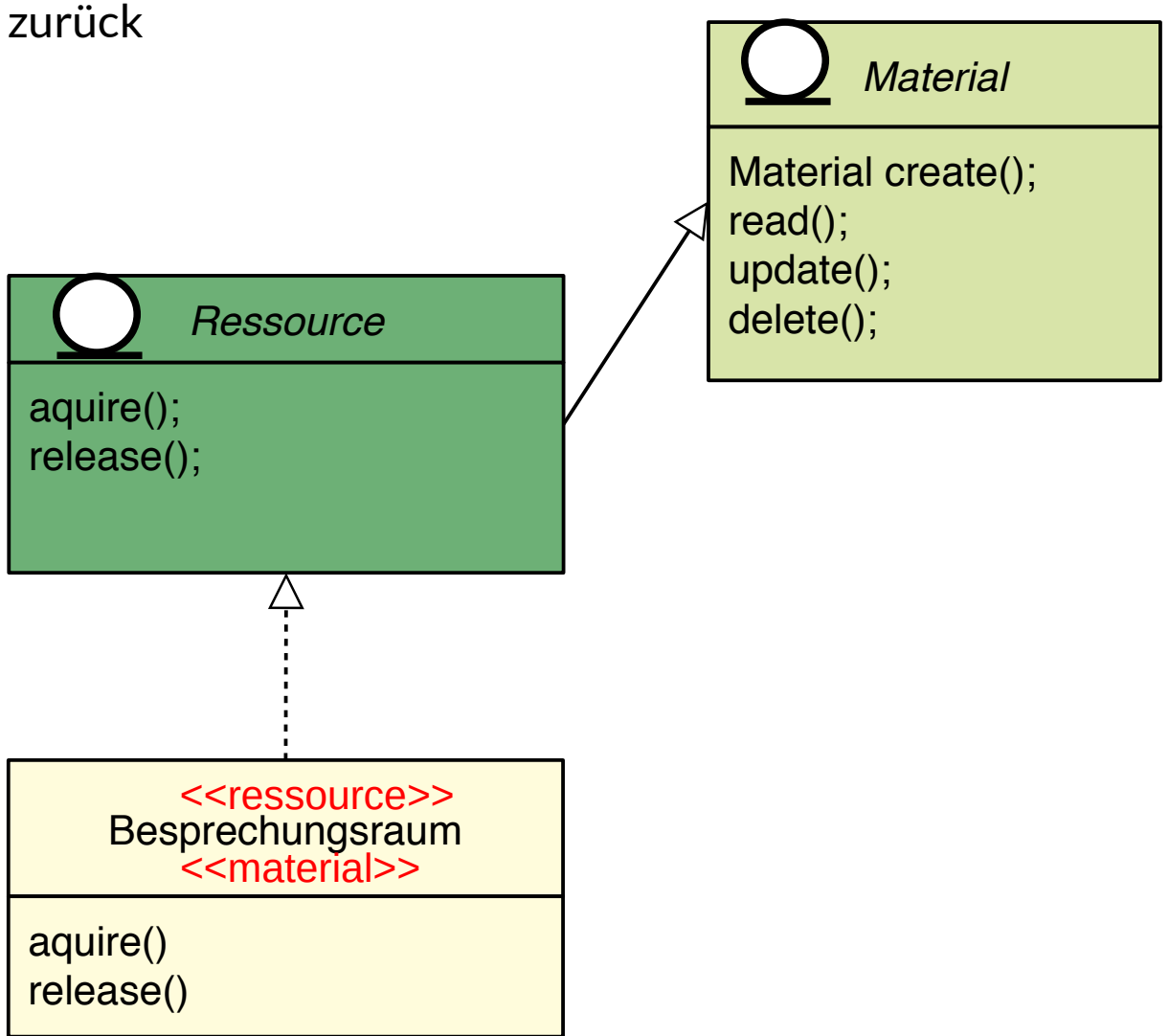
Trenne Tool- von Material-Klassen,
denn sie gehören auf verschiedene Schichten des Systems.



© Prof. U. Aßmann

Ressourcen-Klassen und -Schnittstellen

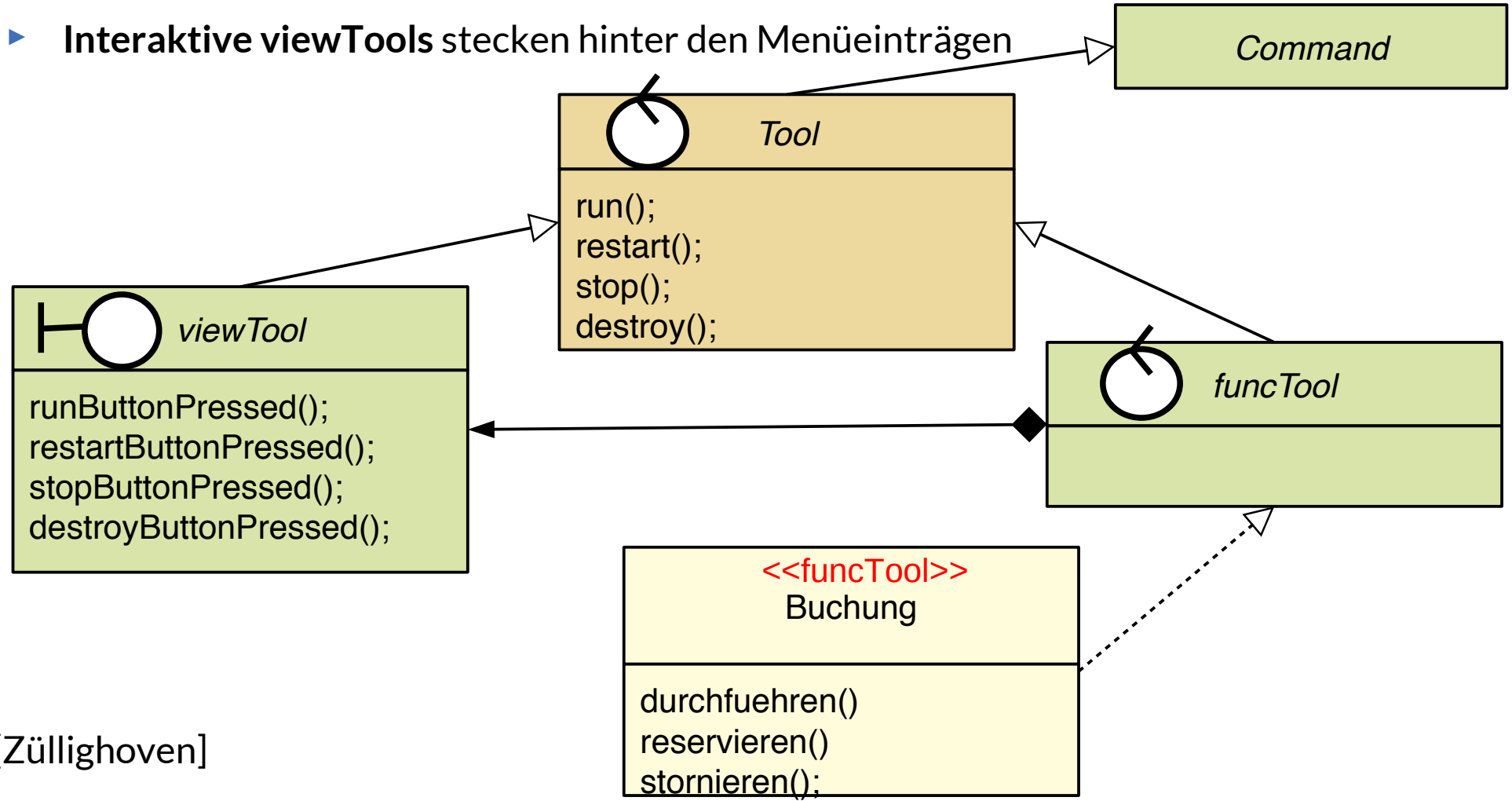
- **Resourcenobjekte** sind *spezielle Materialien*, die vor Nutzung zu *belegen* sind, d.h. sie müssen vor Nutzung alloziert und nach Nutzung freigegeben werden
- **Material-Ressourcen** sind passiv, werden von außen aufgerufen und geben den Steuerfluss nach außen hin zurück



© Prof. U. Aßmann

Tool-Klassen und -Schnittstellen

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

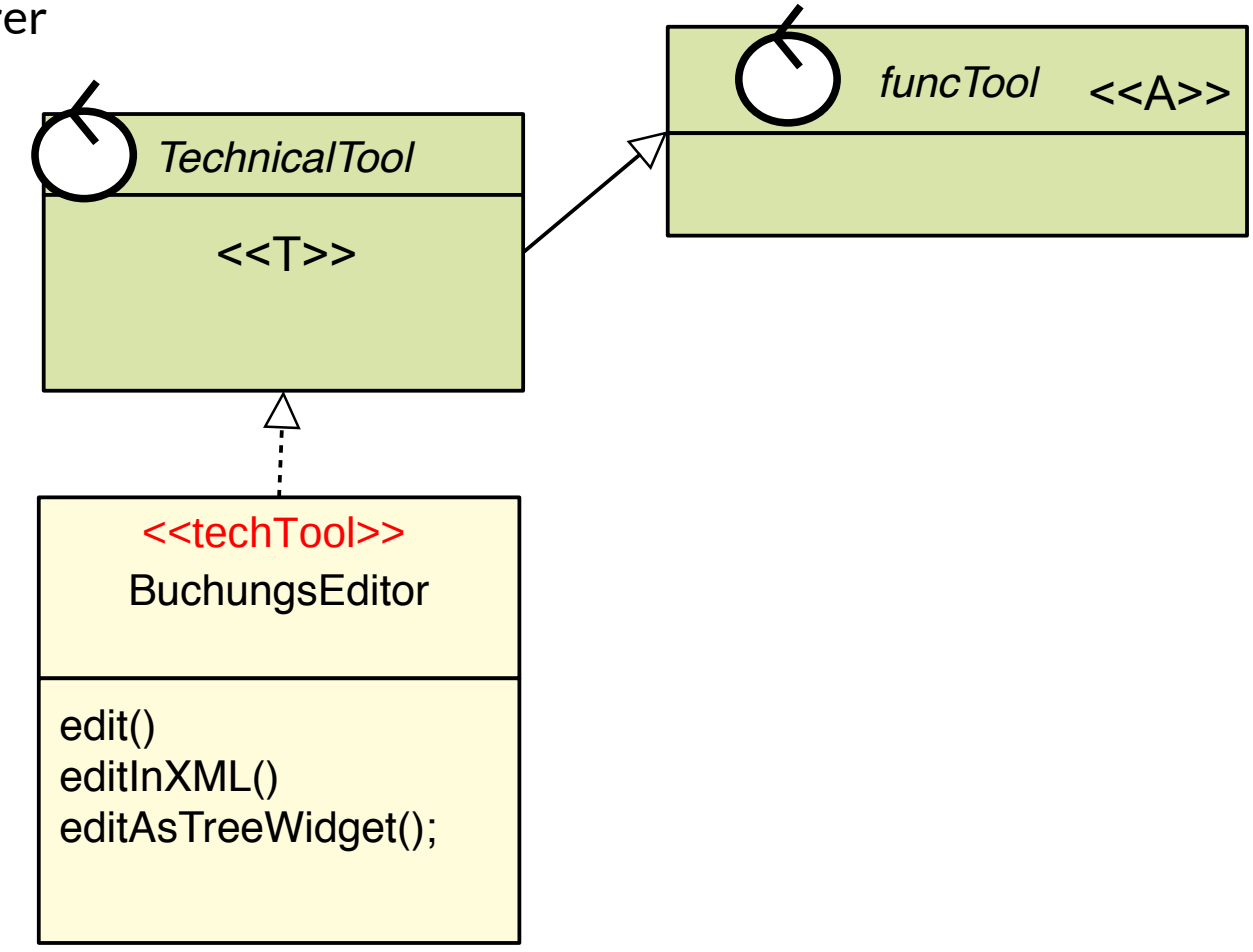


[Züllighoven]

© Prof. U. Aßmann

Technische Tool-Klassen und -Schnittstellen

- **Technische Tools** sind funktionale Tools, die eine technische Funktionalität tragen, die *anwendungsunspezifisch* ist
  - Bsp.: Editor, Lister, Inspektor, Browser, Verschlüsseler, Komprimierer, Optimierer [Züllighoven]

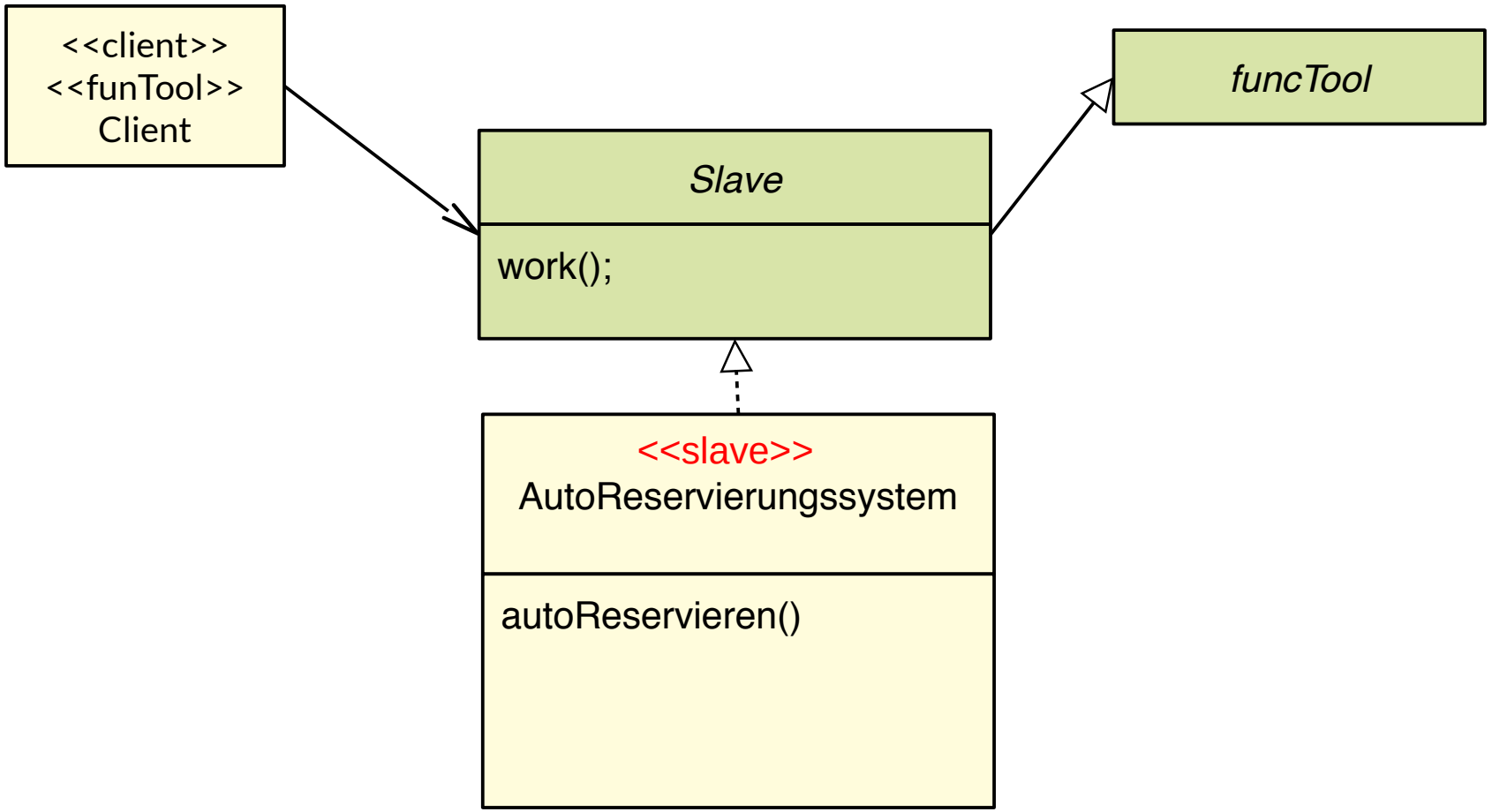


- Technische Tools verwalten das Material und bilden eine C-Teilschicht, T, direkt über der Materialschicht
- Damit trennt sich die C-Schicht in A- und T-Schicht auf

© Prof. U. Aßmann

Slave-Klassen und -Schnittstellen

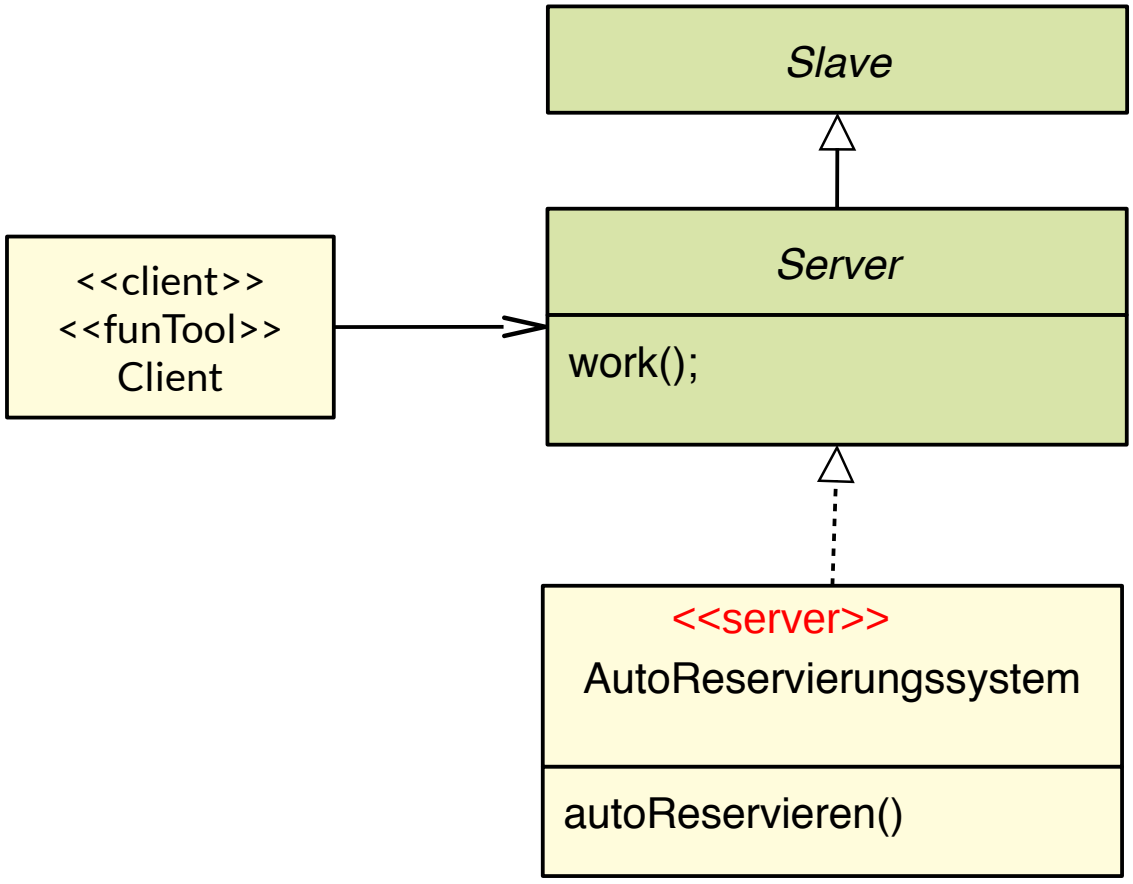
- ***Slave-Objekte*** sind passiv funktionale Tools. Sie werden beauftragt, laufen im batch ab (Design pattern "$Master–Slave$")
- Slave-Objekte bilden also spezielle beauftragbare funcTools (Kommandoobjekte)



© Prof. U. Aßmann

Server-Klassen und -Schnittstellen

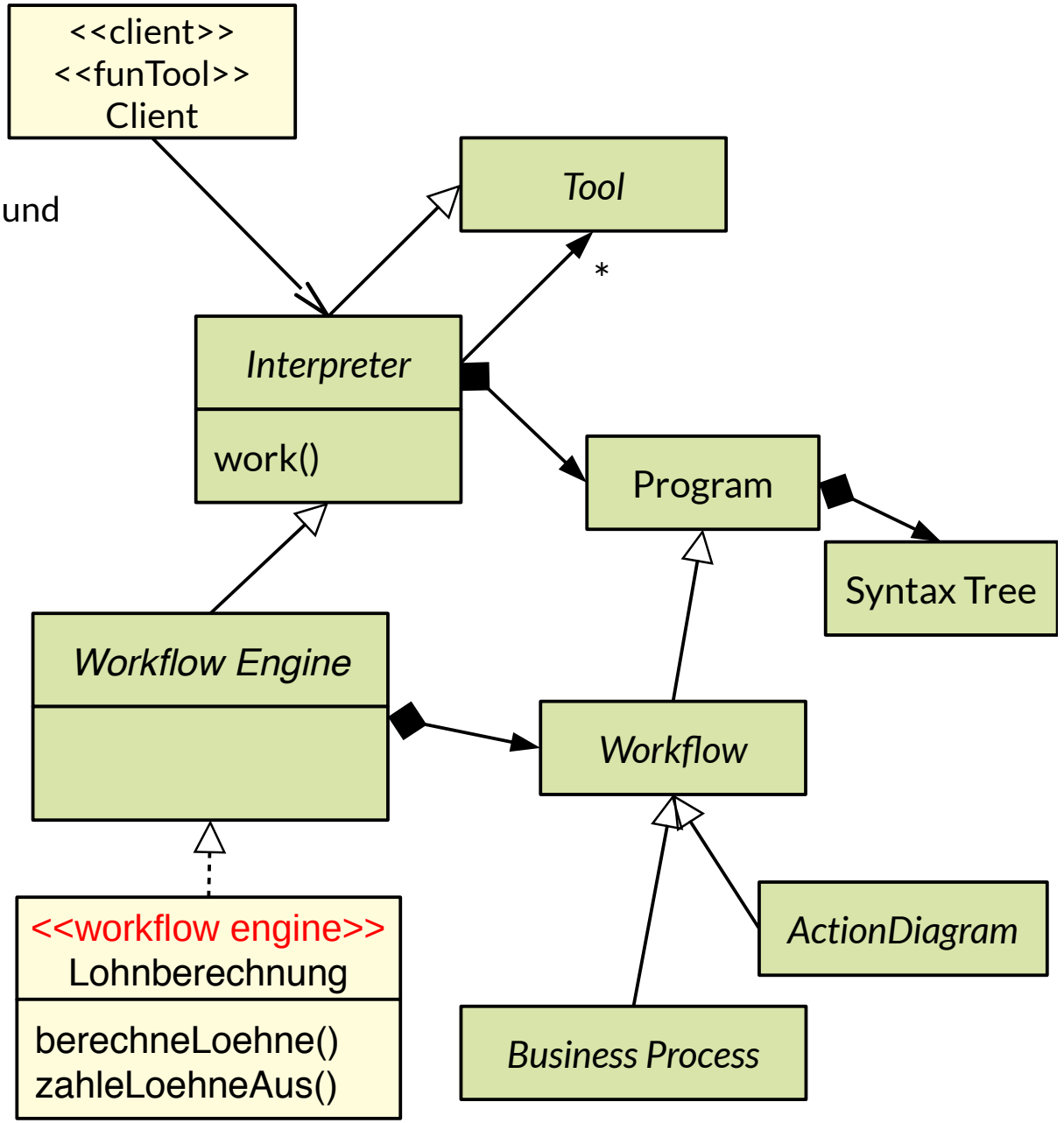
- Ein ***Server***-Objekt ist ein spezielles Slave-Tool, das von einem "Client" mit *verzögert synchronem Auftrag* beauftragt wird (Design pattern Client–Server)
  - Sie können einen eigenen Steuerfluss besitzen (thread, process) und damit mehrere Anfragen gleichzeitig bearbeiten
- Serverobjekte bilden also spezielle SlaveTools



© Prof. U. Aßmann

Design Pattern Interpreter
Workflow-Engine-Klassen und -Schnittstellen

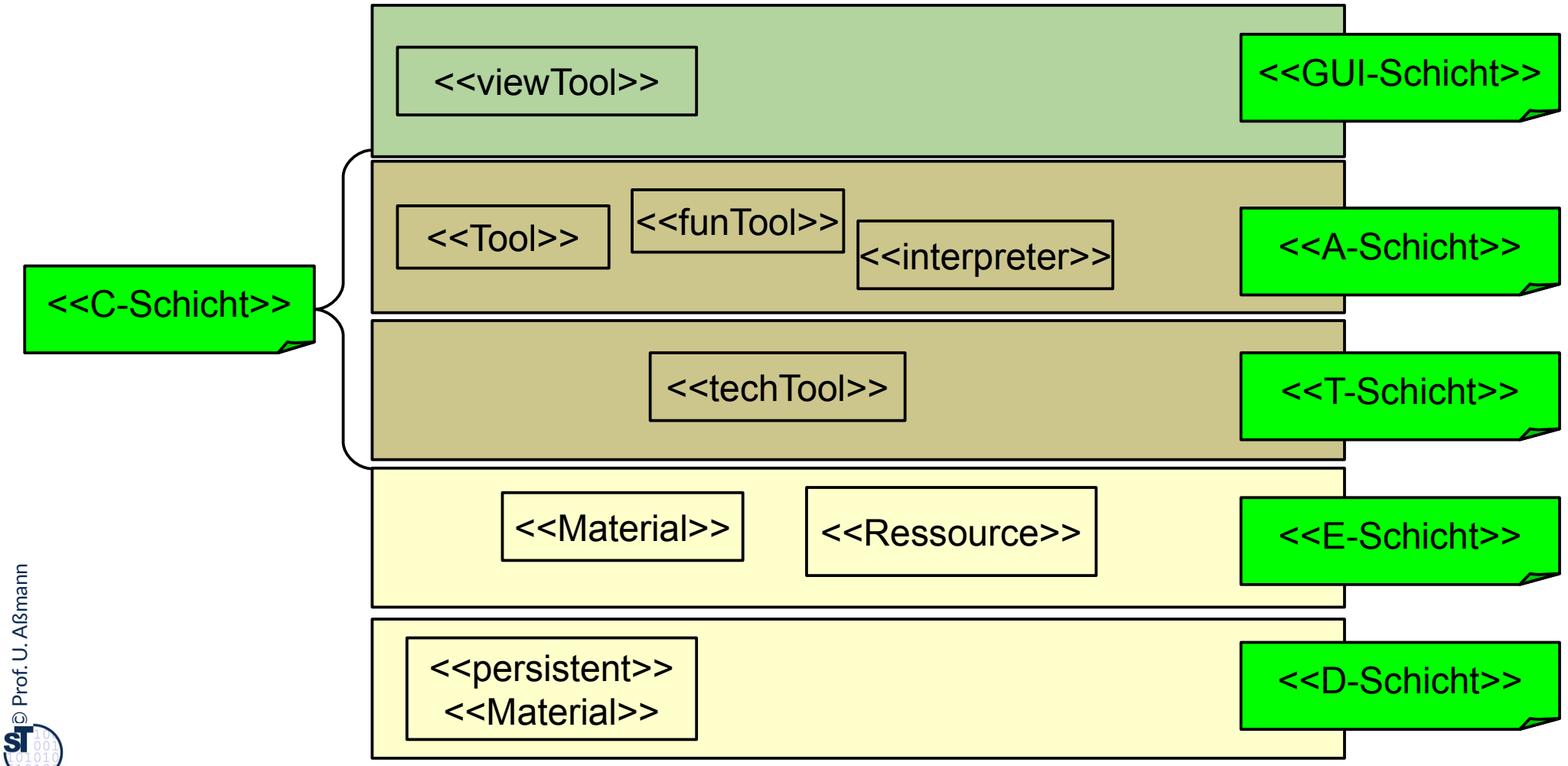
- Def.: Ein *Interpreter* (**Workflow-Engine,Workflow-Interpreter)** ist ein funktionales Tool, das einen komplexen Arbeitsablauf in Form eines Workflows oder Programms abarbeitet (interakiv oder batch) und andere Tools ansteuert
  - Das Programm ist als Syntaxbaum (*Composite*, *Visitor*) vorhanden
  - Interpreter rufen andere Tools auf und steuern sie an
- Workflows können beschrieben werden durch
  - Aktionsdiagramme (Aktivitätendiagramm, Statecharts),
  - Sprachen für Business Processes (Business Process Modeling Notation, BPMN)
- Workflow-Engines gehören zu C-Schicht



© Prof. U. Aßmann

Das TAM-Separierungsgesetz (Tools-Material-Separation Gesetz)

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



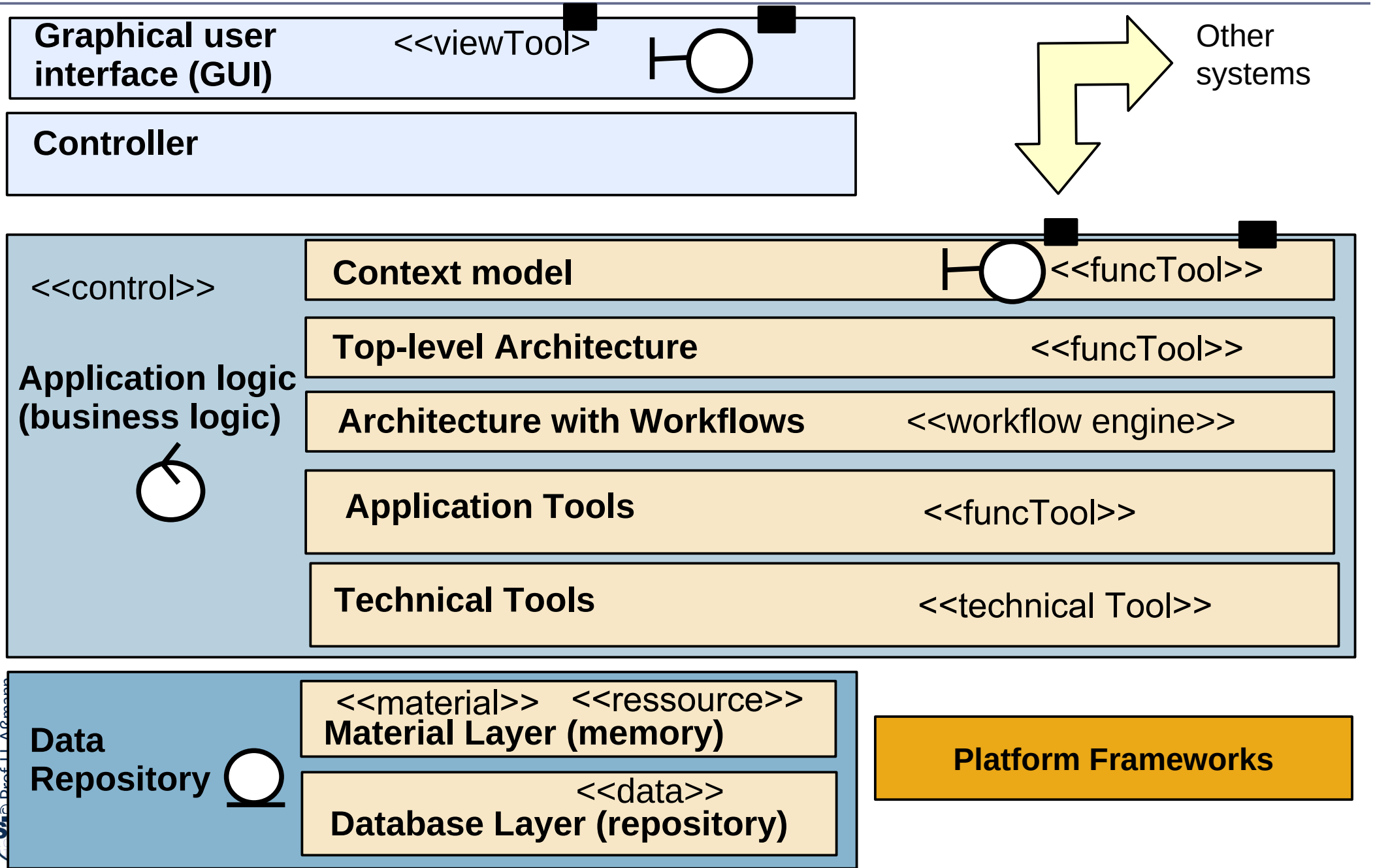
© Prof. U. Aßmann

Frage: Wie ordnet man TAM-klassifzierte Objekte den BCED-Schichten zu?

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

© Prof. U. Aßmann

Q8: Verfeinerte BCED-Schichtung eines Systems mit TAM

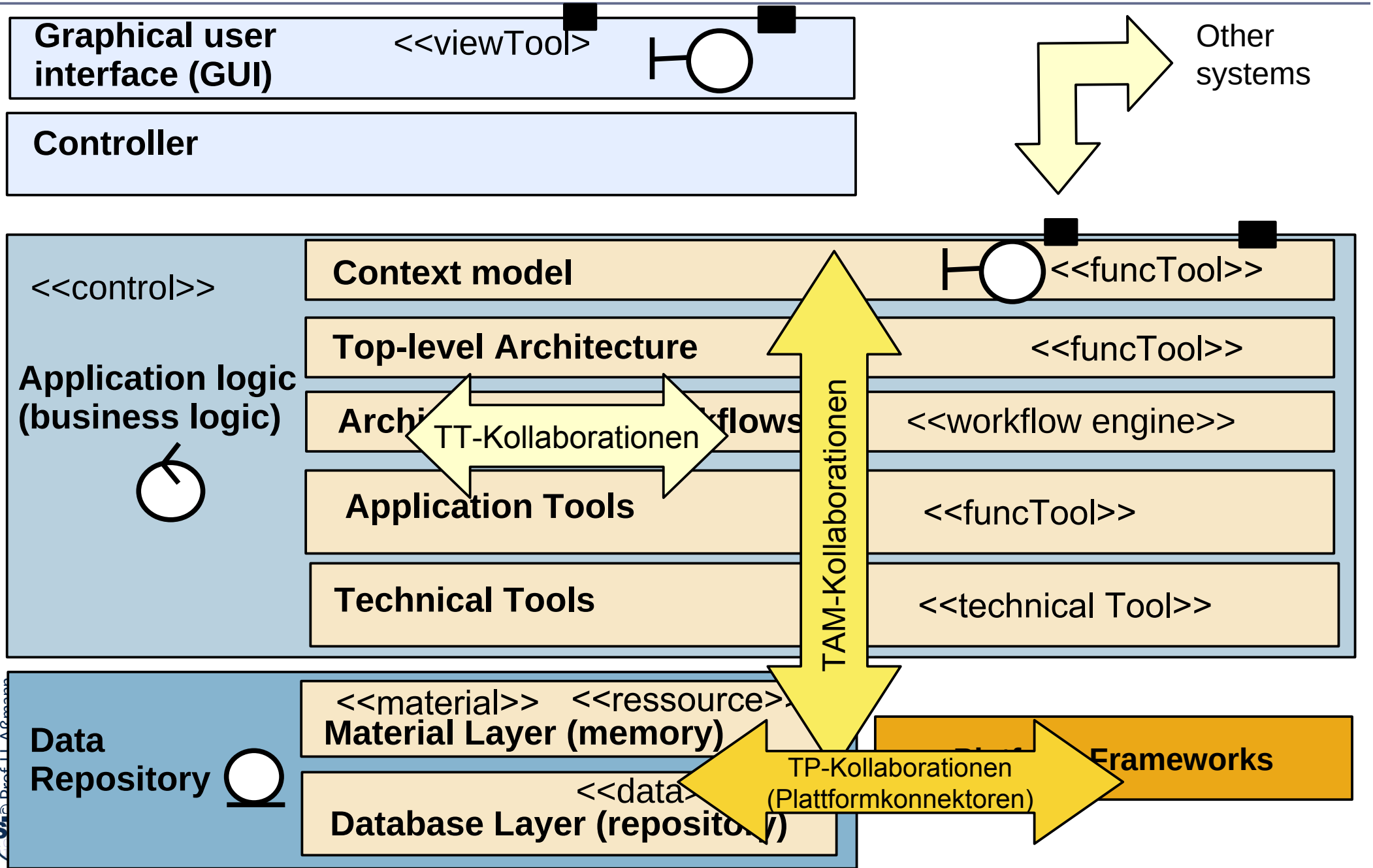


© Prof. U. Aßmann

42.2 Querscheidende Verfeinerung in der BCED-Schichtenarchitektur mit TAM-Kollaborationen

- Einordnung in die Schichten durch TAM
- Tools und Interpreter gehören zur Anwendungslogik
- Materialien in die Datenhaltung
- Verfeinerung durch TAM-Kollaborationen

Q8: Verfeinerte BCED-Schichtung eines Systems mit TAM



TAM-Kollaborationen

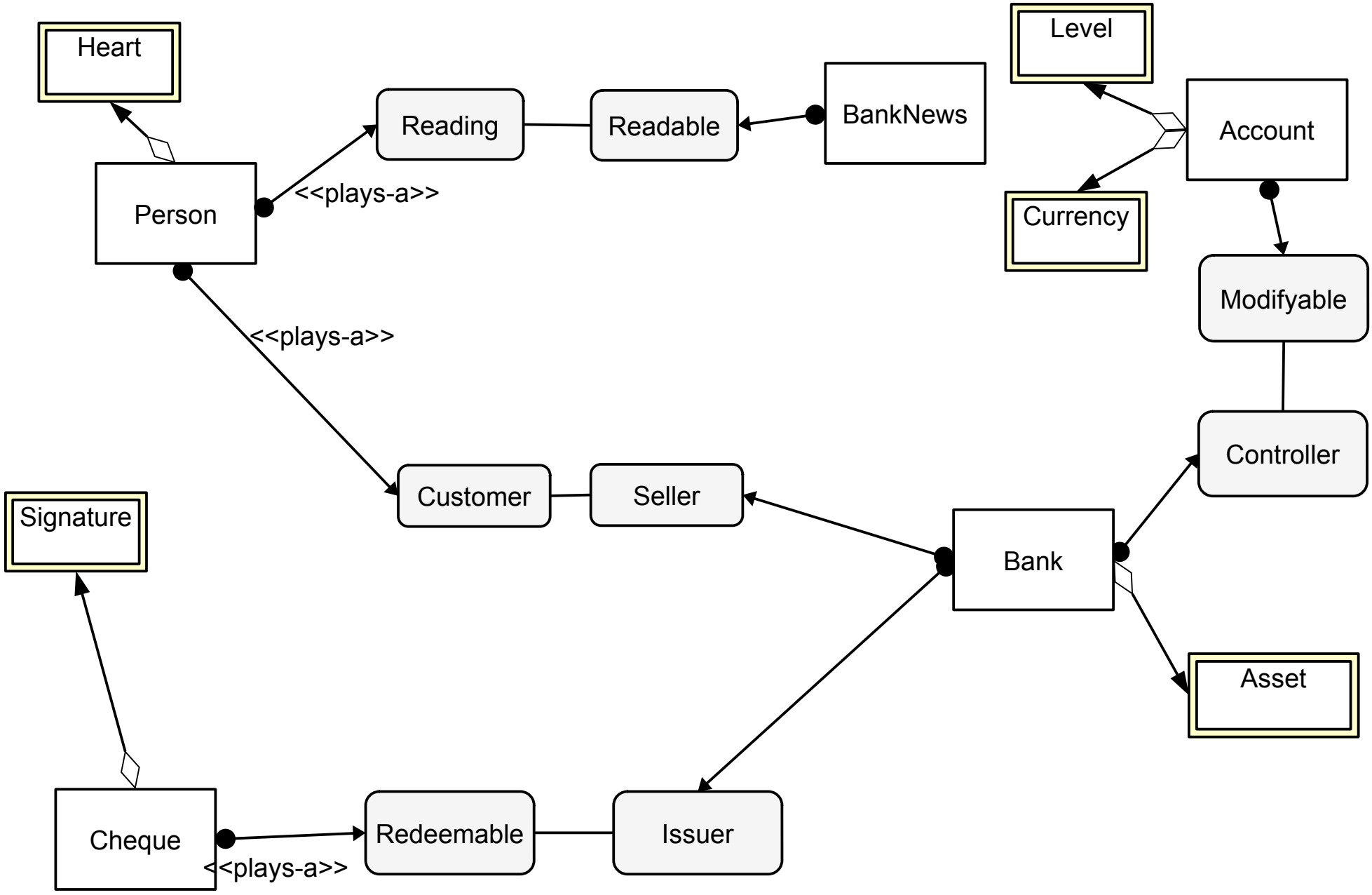
- Eine ***TAM-Kollaboration*** ist eine *schichtenübergreifende Kollaboration* zwischen einer Gruppe von Tool- und Material-Objekten.
- Eine ***TAM-Kollaborationsklasse*** fasst das in einer Klasse.

- Eine ***TT-Kollaboration*** ist eine Kollaboration zwischen einer Gruppe von Tool-Objekten.
- Eine ***TP-Kollaboration*** ist eine Kollaboration zwischen einer Gruppe von Tool-Objekten und Platform-Objekten.

- Alle Arten dieser Kollaborationen können auch reifiziert vorliegen:
  - (TT-, TAM-, TP-)**Teams** (Kollaboration mit refiziertem Hauptobjekt)
  - (TT-, TAM-, TP-)**Konnektoren** (Technische Teams)

© Prof. U. Aßmann

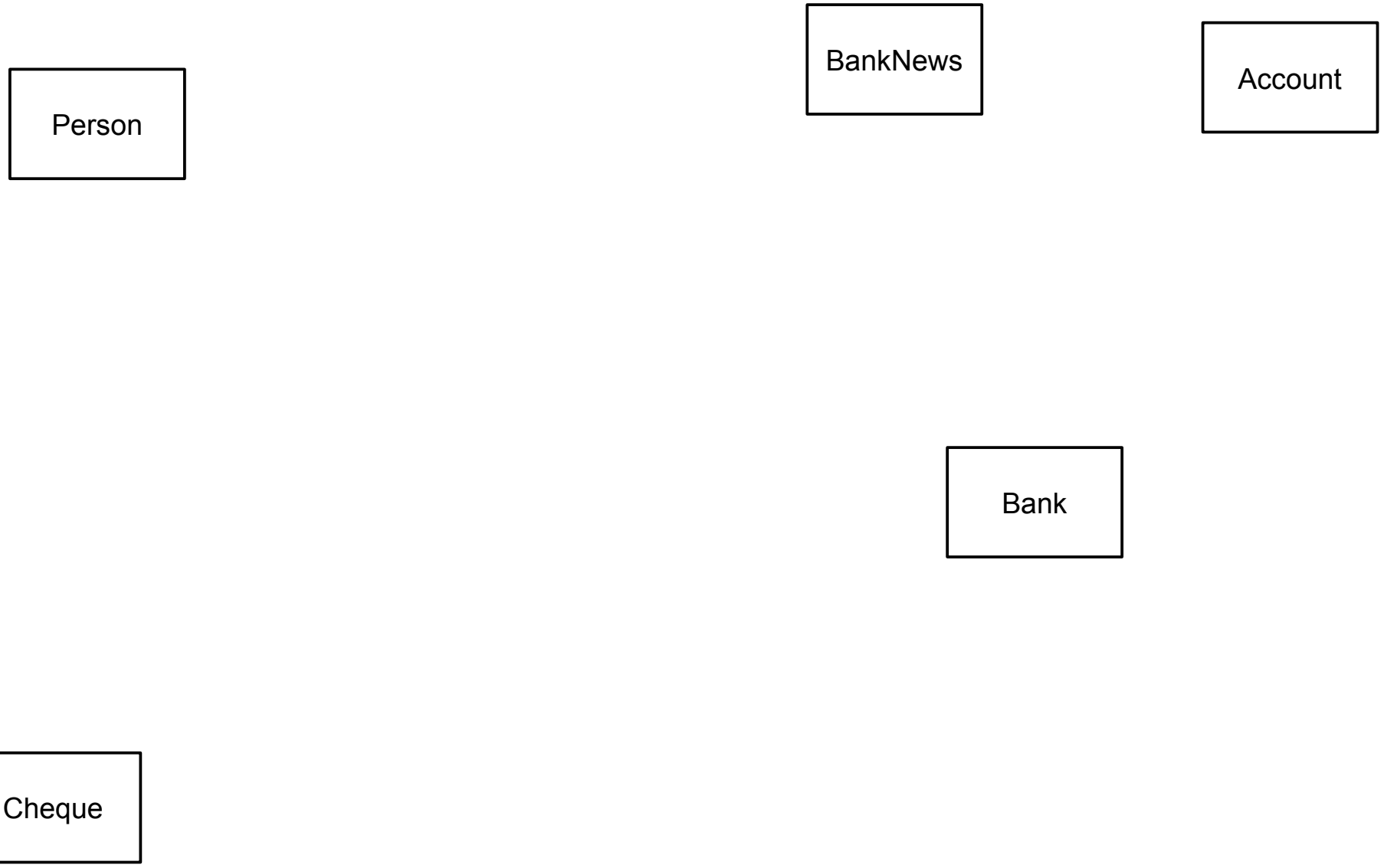
Beispiel: Personen-Analysemodell mit Rollenobjekten und Teilen – Wie teile ich das auf die Schichten auf?



© Prof. U. Aßmann

Mit Verfeinerung durch Integration von Unterobjekten (Mixin-Anreicherung, Object Fattening)

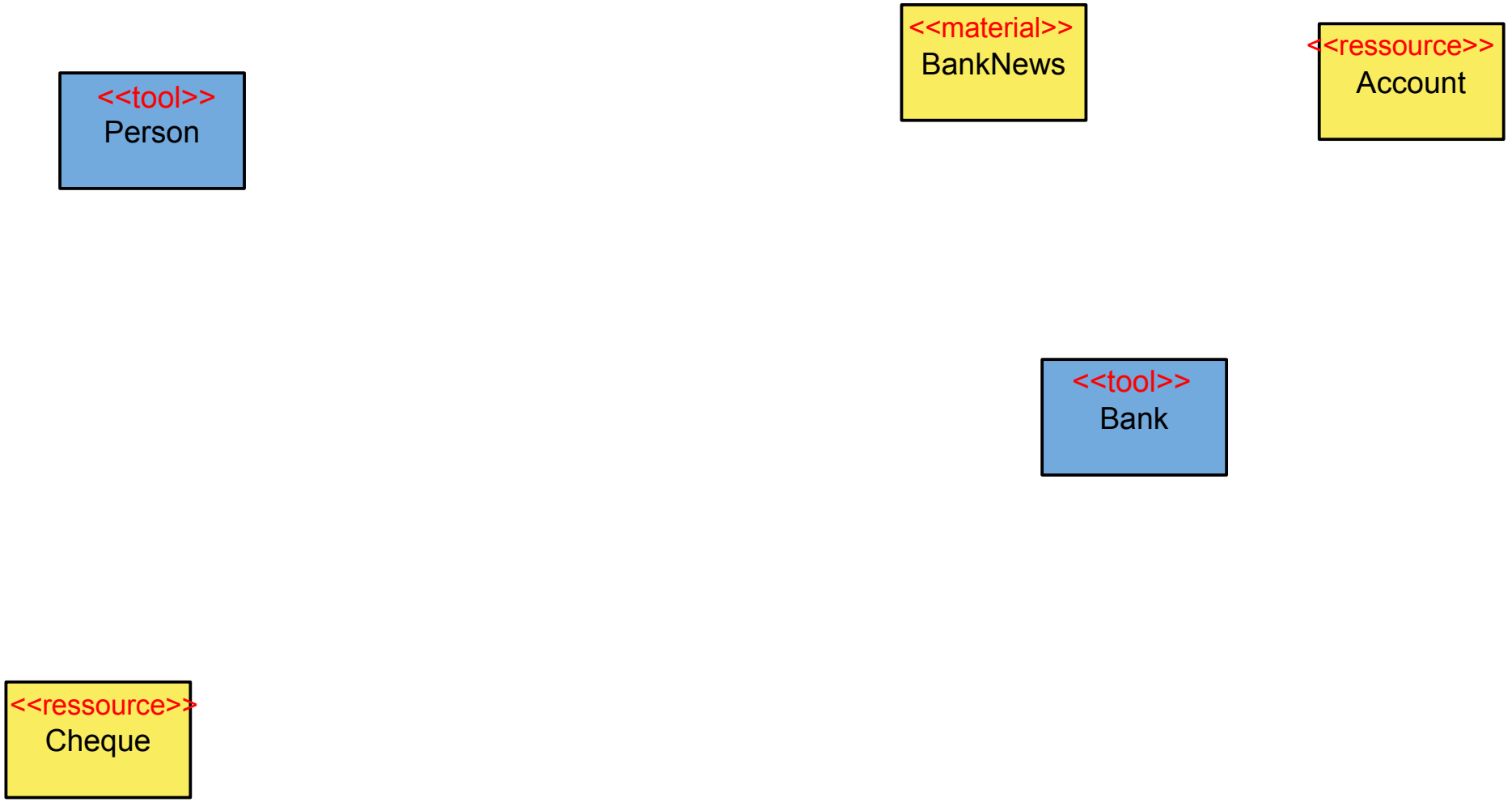
- Rohzustand: Identifikation der natürlichen Typen (in dem Domänenmodell)



© Prof. U. Aßmann

Mit Klassifikation von Tools and Materials

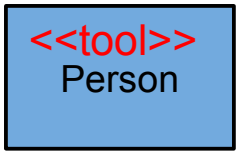
- Bestimme Tools, Servers, Interpreters in der C-Schicht, mit Unter-Schichten A (Application Logic) und T (Technical Logic)
- Bestimme Materials, Ressourcen (E- und D-Schicht, falls persistent)

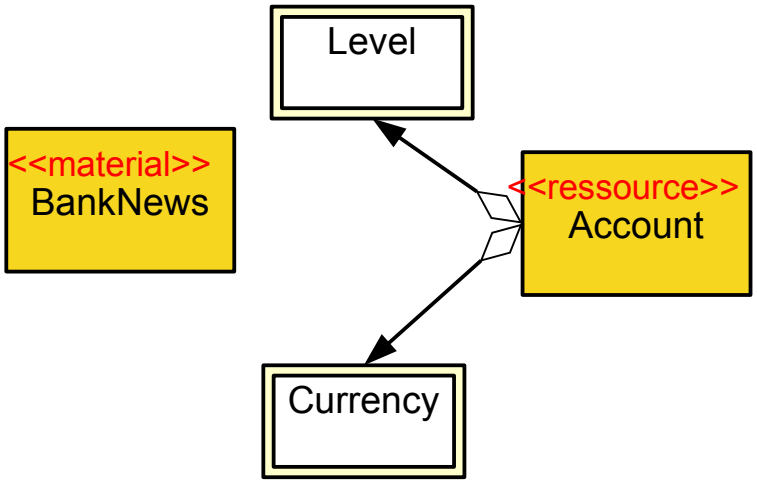


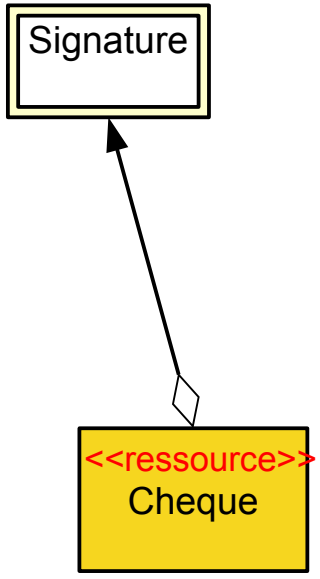
© Prof. U. Aßmann

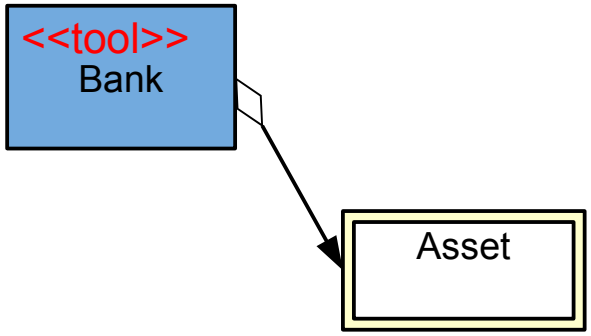
Mit Verfeinerung durch Integration von Unterobjekten (Object Fattening)

- Schritt 1: Teile-Verfeinerung





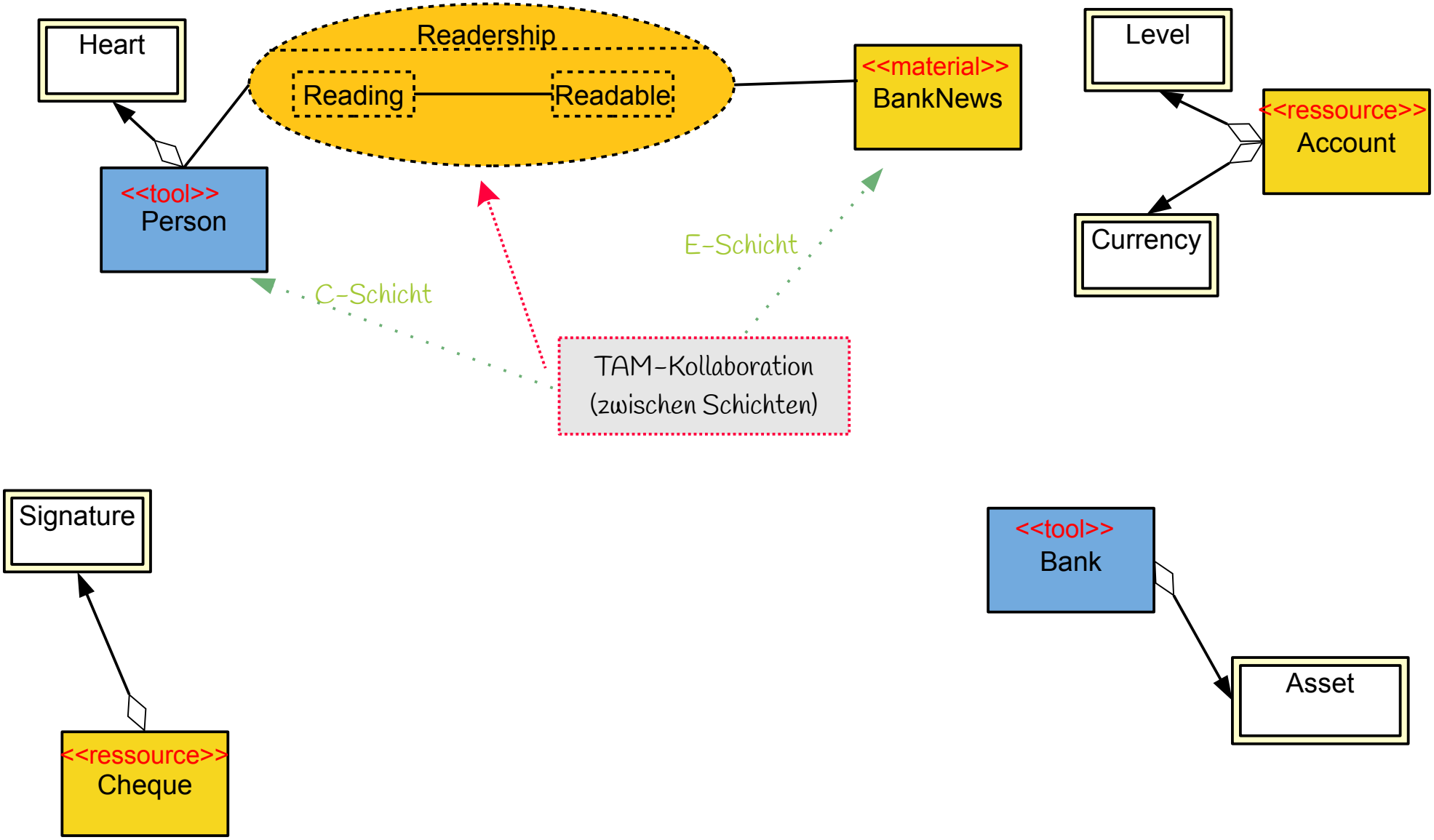




© Prof. U. Aßmann

Mit Querschneidender Verfeinerung durch TAM-Kollaborationen zwischen Tool- und Materialschichten

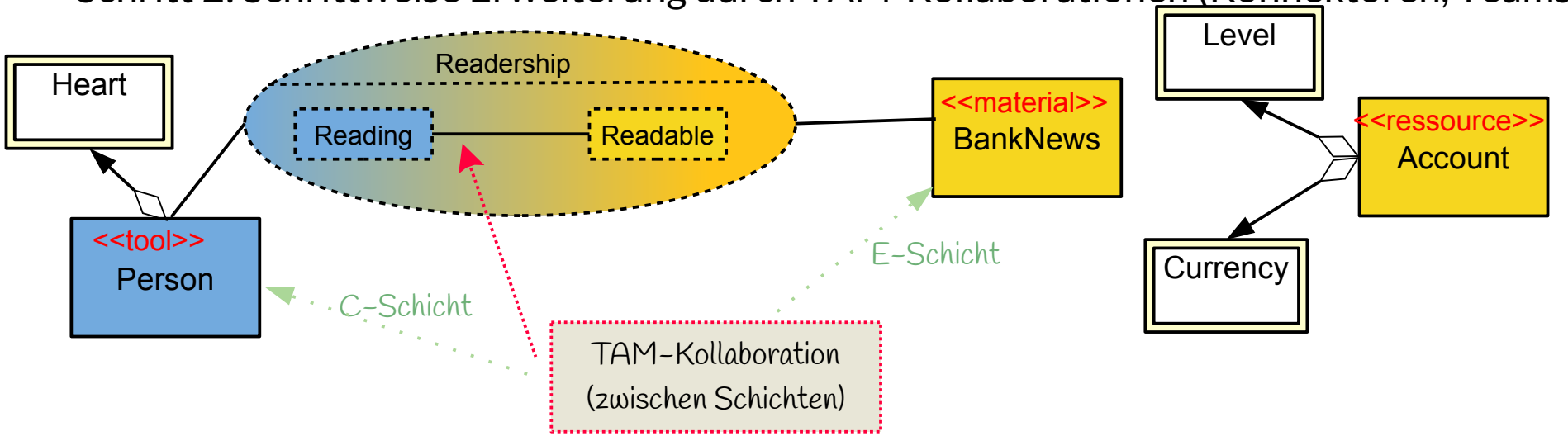
- Schritt 2: Schrittweise Erweiterung durch **TAM-Kollaborationen**

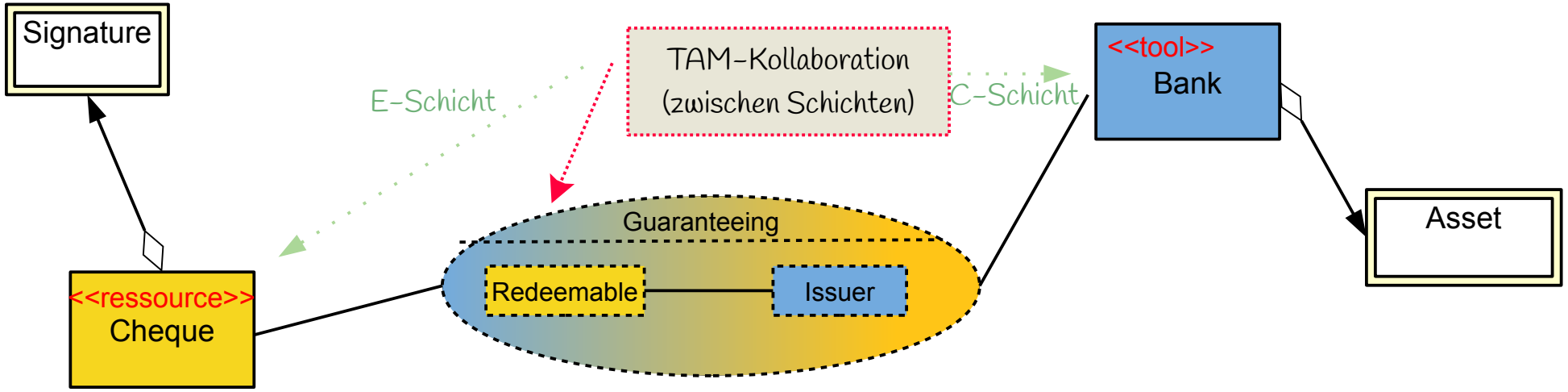


© Prof. U. Aßmann

Mit Querschneidender Verfeinerung durch TAM-Kollaborationen

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

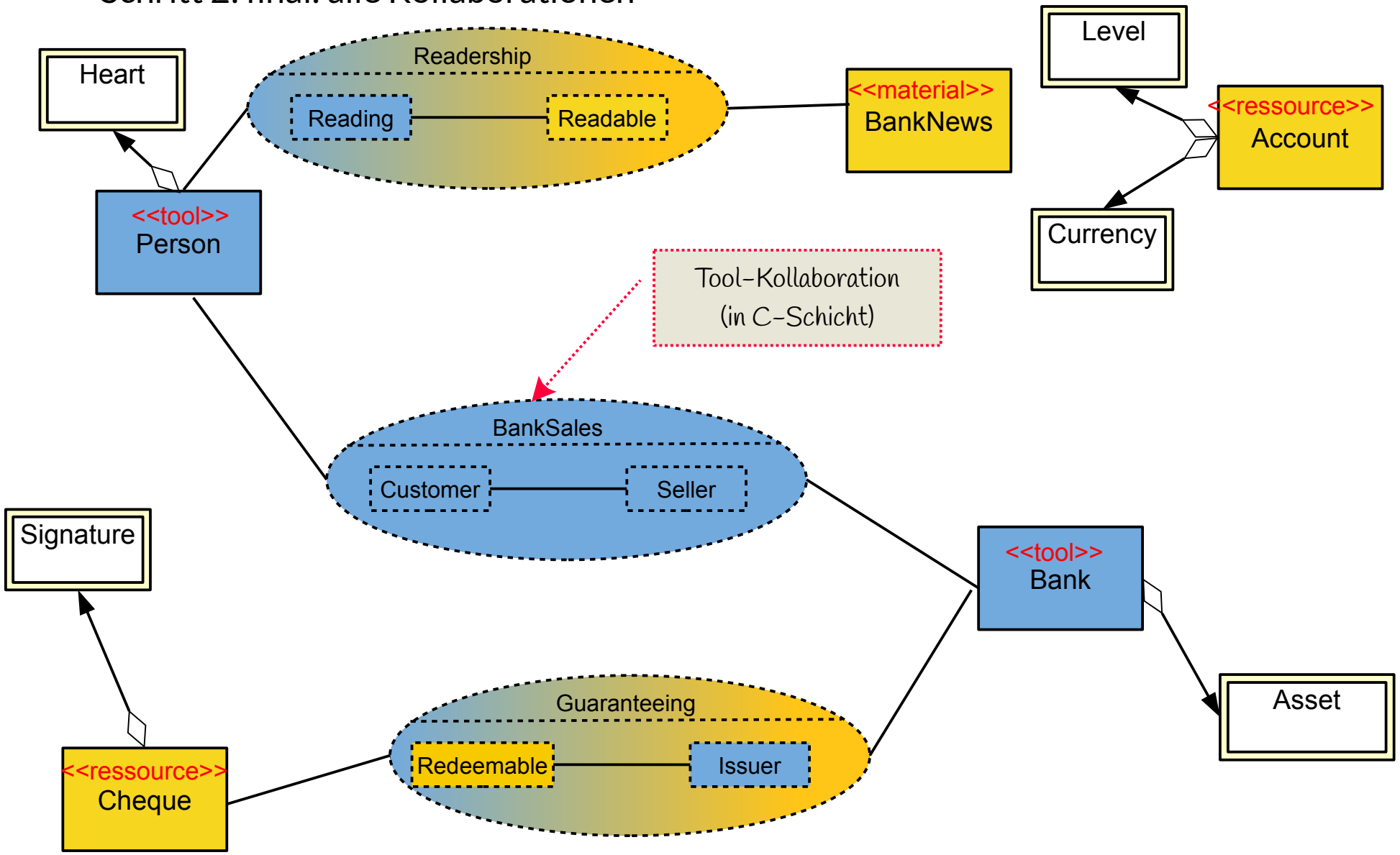




© Prof. U. Aßmann

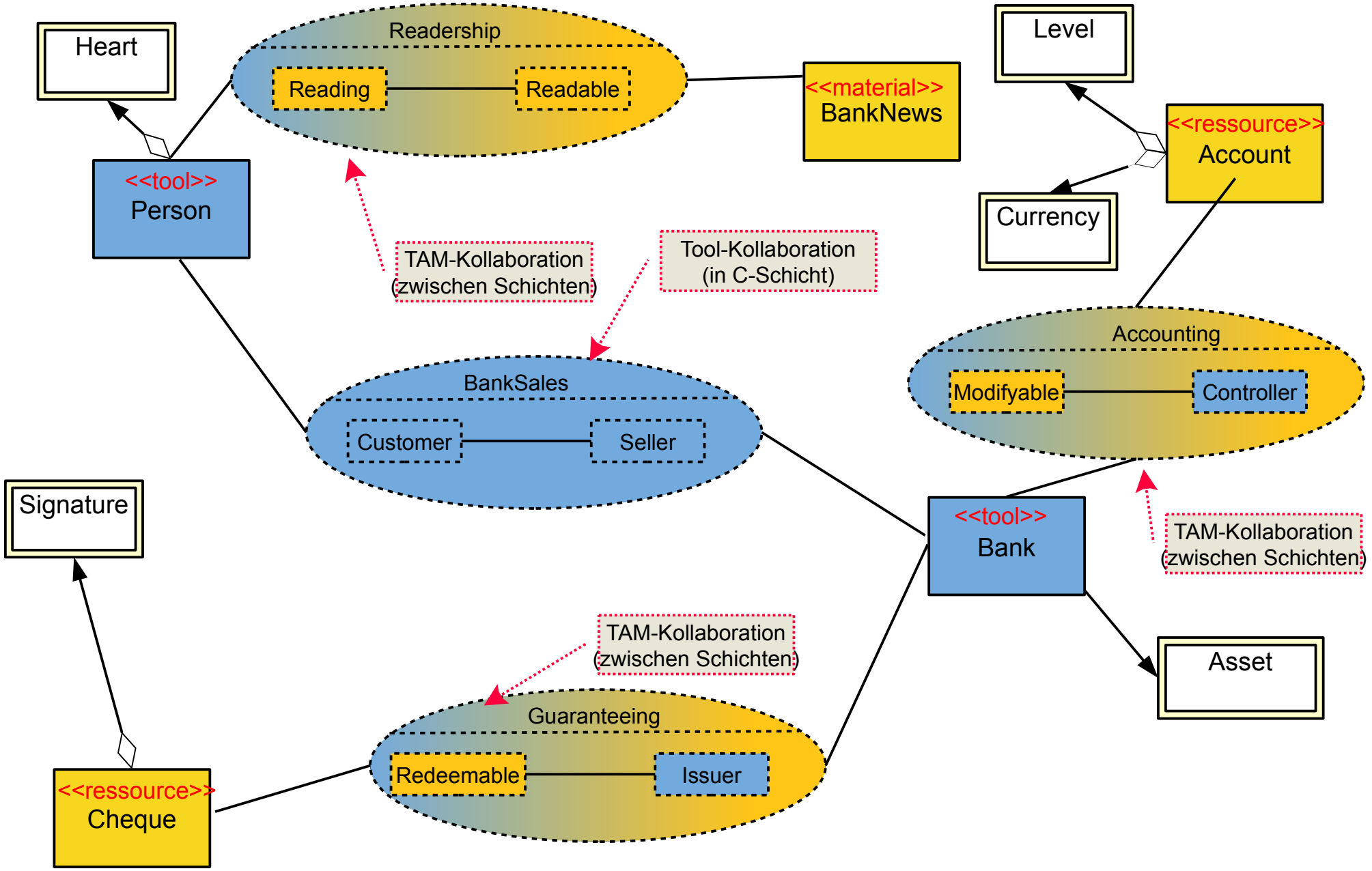
Mit Querschneidender Verfeinerung durch Tool-Tool-Kollaborationen

- Schritt 2: final: alle Kollaborationen



© Prof. U. Aßmann

Analysemodell – Angereichert durch Einziehen von querschneidenden Kollaborationen



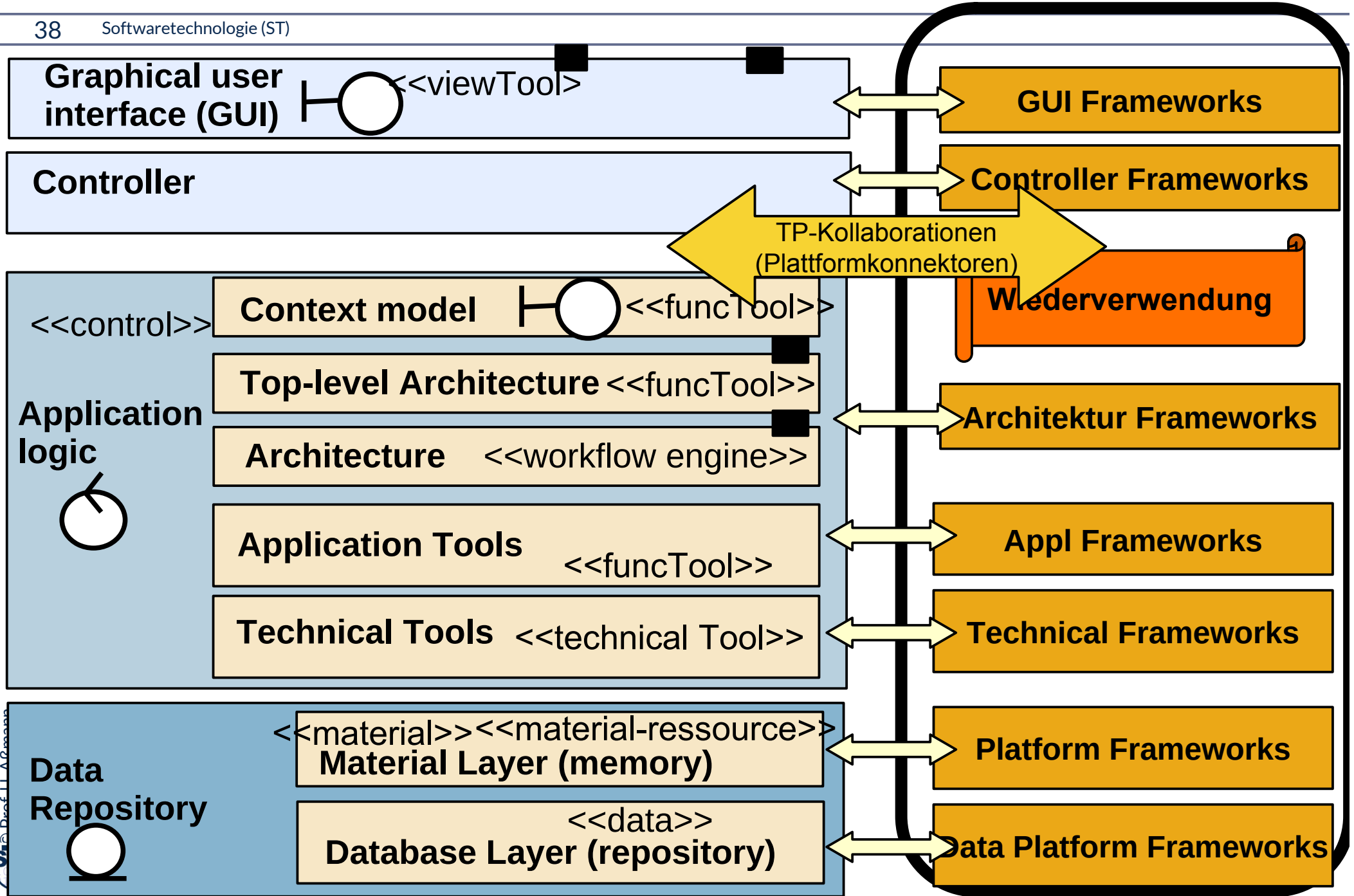
© Prof. U. Aßmann

42.3 Feinentwurf: Plattformanpassung mit Plattformkollaborationen

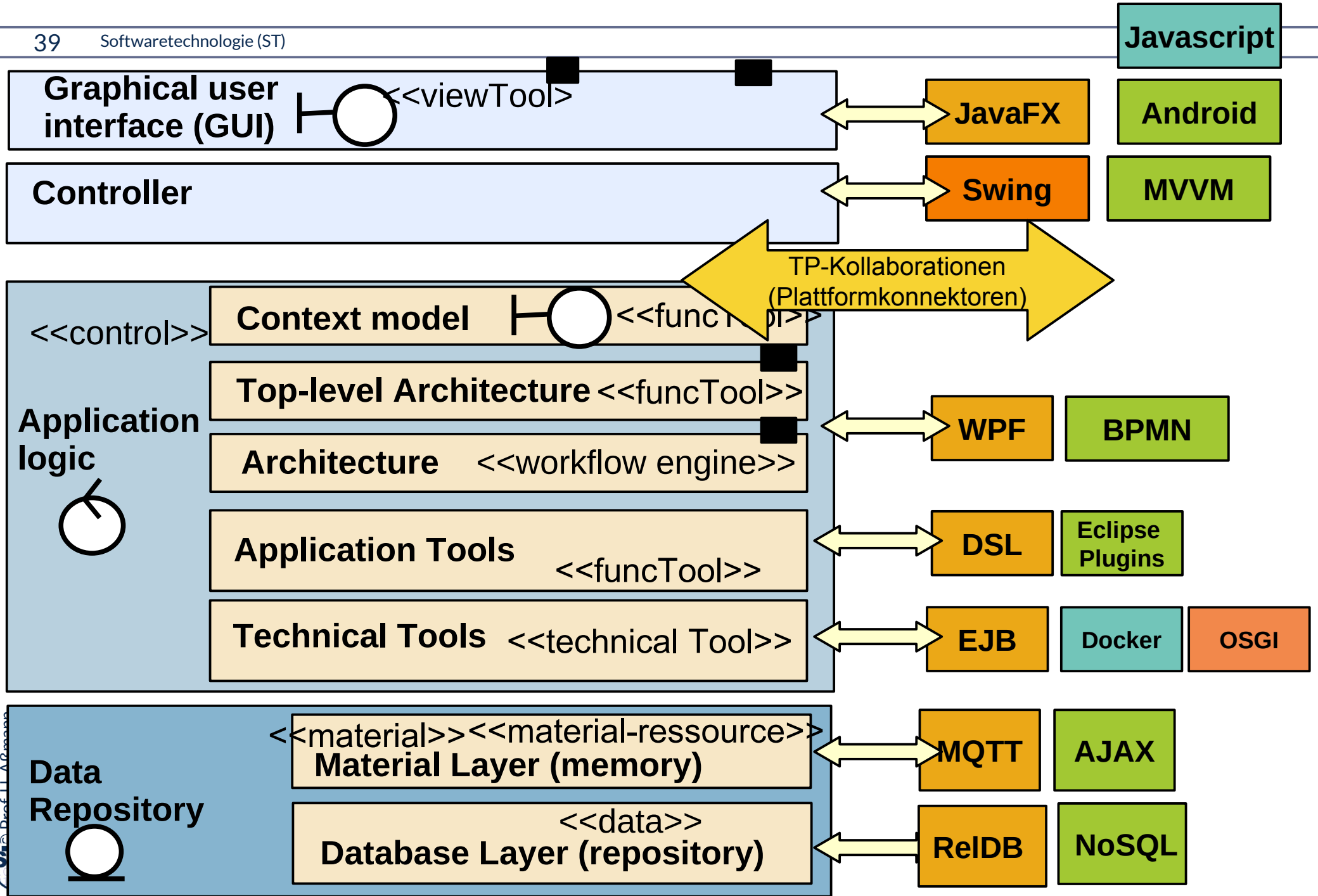
Verfeinerungsbeispiel für Anpassung auf Plattformen

.. Verfeinerung durch Integration von Unterobjekten..
Teile und Rollen zu Plattformobjekten hin

Q7': Verfeinerte BCED-Schichtung eines Systems mit TAM und Technik-Plattform-Frameworks



Plattform-Wechsel immer und überall – wie beherrschen?



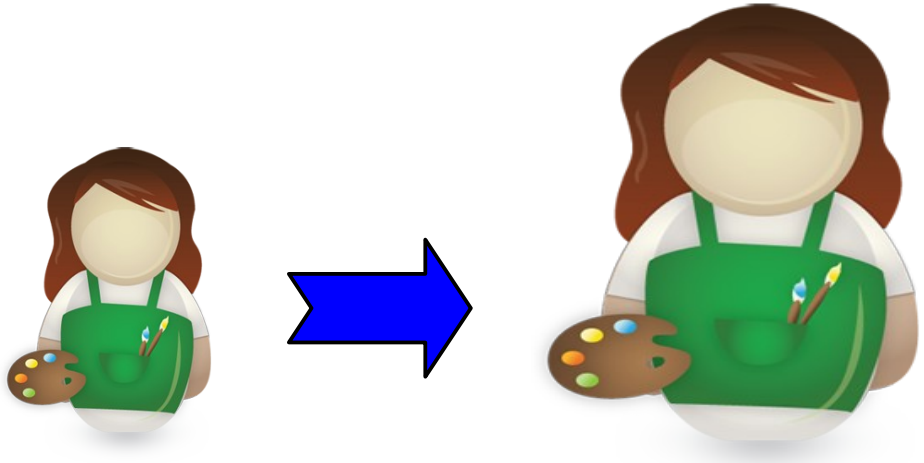
42.3.1 Anpassung von Material-Objekten an Plattformen
Mixin-Anreicherung mit Plattforminformation (Querschneidende Verfeinerung für Plattformen)

.. Verfeinerung durch Integration von Unterobjekten..

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Plattformen als Komponenten der Infrastruktur

- Eine ***Plattform*** ist eine Komponente der Infrastruktur (Schicht, Bibliothek oder Framework), auf denen die Anwendung aufgebaut wird
  - Plattformen kommen immer von Drittanbietern, sind also Components-off-the-shelf (COTS)
  - Beispiele: Betriebssystem, Datenbanken, Middleware, SAP, Android, etc.
- Def.: ***Plattform-Verfeinerung*** ist ein Mixin-Anreicherungs-Prozess zur Entwurfszeit, der die Anwendung mit Kollaborationen zu Plattformen, mit plattform-spezischem Verhalten, ergänzt (Plattform-Verfeinerung)
- Die hinzugefügten ***Plattform-Kollaborationen*** und ***Plattform-Konnektoren*** mit ihren Rollen klären Beziehungen zu Plattformen



© Prof. U. Aßmann

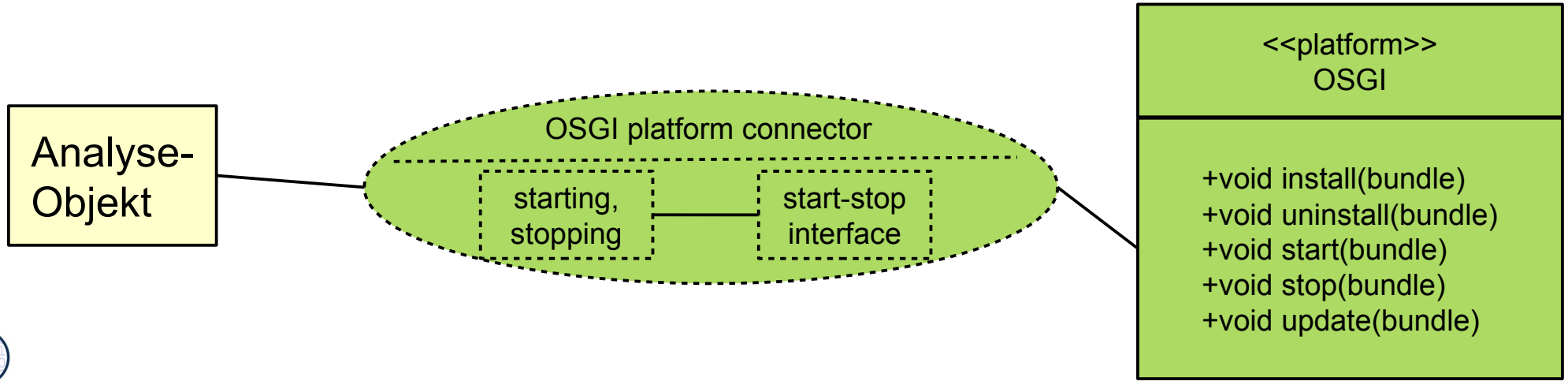
Plattformverfeinerung – Schritte im Feinentwurf

- Plattformverfeinerung findet **Plattform-Konnektoren,** mit Rollen-Unterobjekten, 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
  - Plattformkomponenten und -objekten (Frameworks, Middleware, Sprachen, Komponenten, Services)
  - Komponentenmodellen (durch Adaptergenerierung)
  - Realisierung der Integrationsrelation
- **Realisierung** der Konnektoren und der Integrationsrelation
  - Einfache Implementierung durch Konnektoren oder Entwurfsmuster

© Prof. U. Aßmann

Plattformobjekte und Plattform-Konnektoren

- Def.: 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
- Def.: Eine **Plattformkollaboration** kapselt die Kollaboration eines Anwendungsobjekts mit der Plattform durch einen Konnektor zum Plattformobjekt
- OSGI: Komponentenplattform www.osgi.org kann in einem OSGI-Objekt gekapselt werden
  - im Handy, 5er BMW, in Eclipse 3.0, Shell home automation HomeGenie
  - Ein *bundle (Komponente)* paketiert verschiedene Klassen

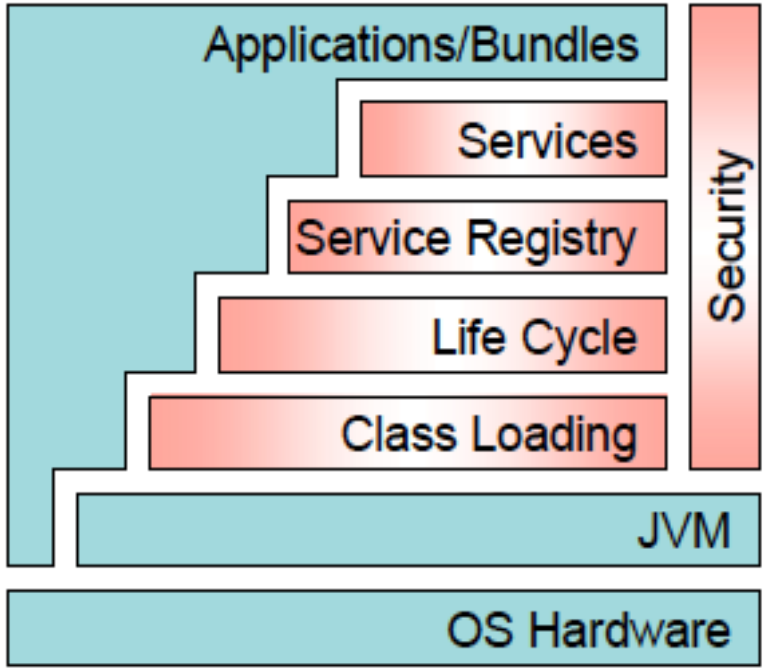


© Prof. U. Aßmann

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

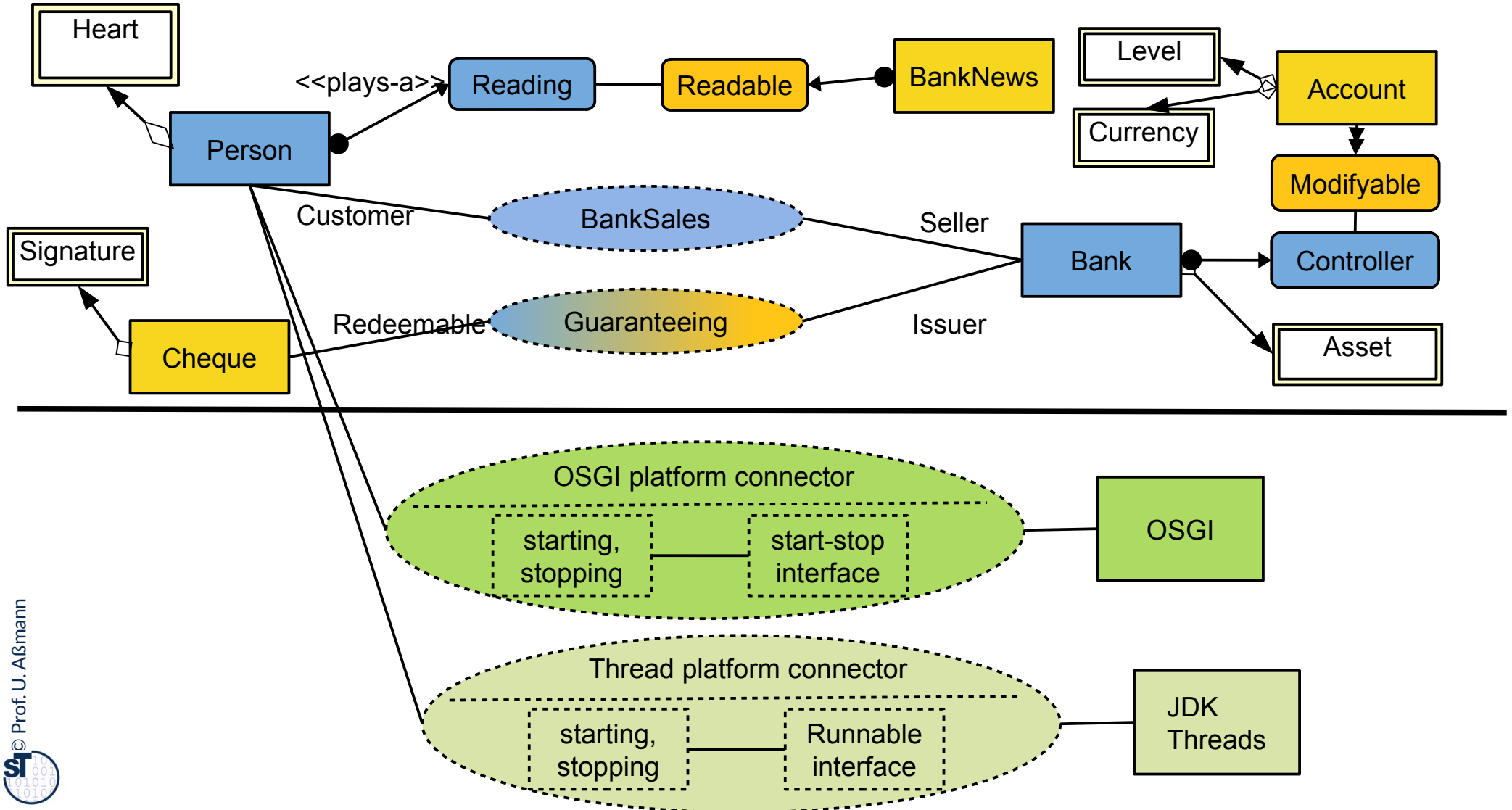
- [OSGI Technical White Paper]



© Prof. U. Aßmann

Mit Verfeinerung durch Plattform-Konnektoren (platform fattening)

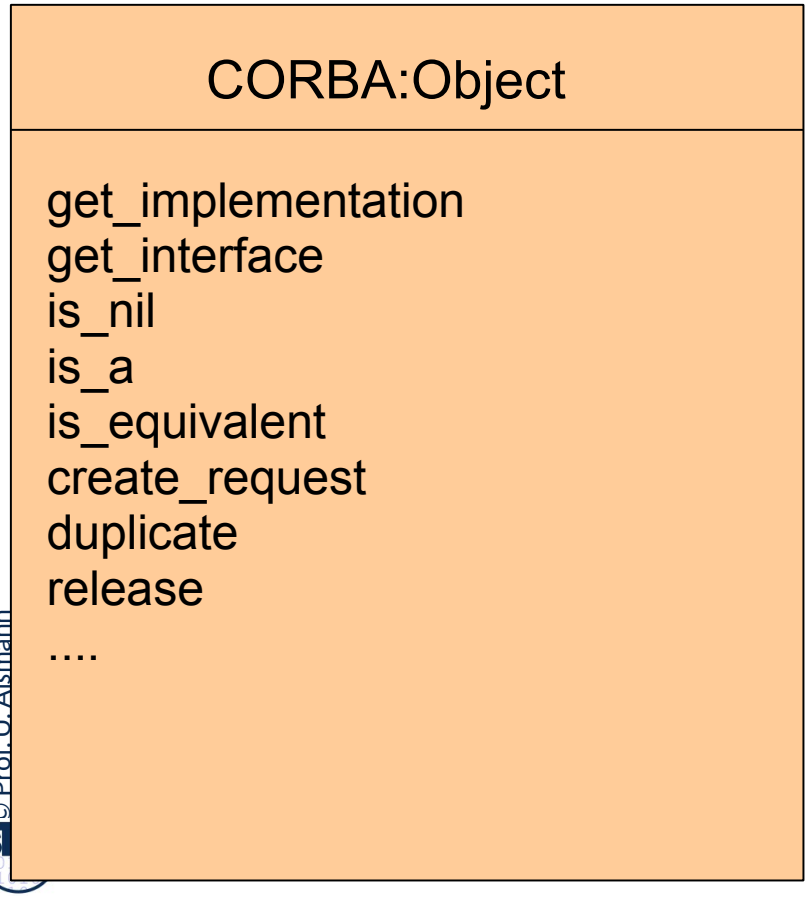
- Plattform-Konnektoren beschreiben die Beziehungen zu Plattformobjekten sowie die Interaktion der Anwendungsobjekte mit ihnen (orange; Analyse-Konnektoren: lila)
- Plattformobjekte können als Alternativen existieren (hier OSGI, JDK threads) für die Plattform "Lebenszyklus"



© Prof. U. Aßmann

Plattform CORBA: CORBA:Object

- CORBA bildet eine Komponentenplattform für heterogen programmierte Systeme

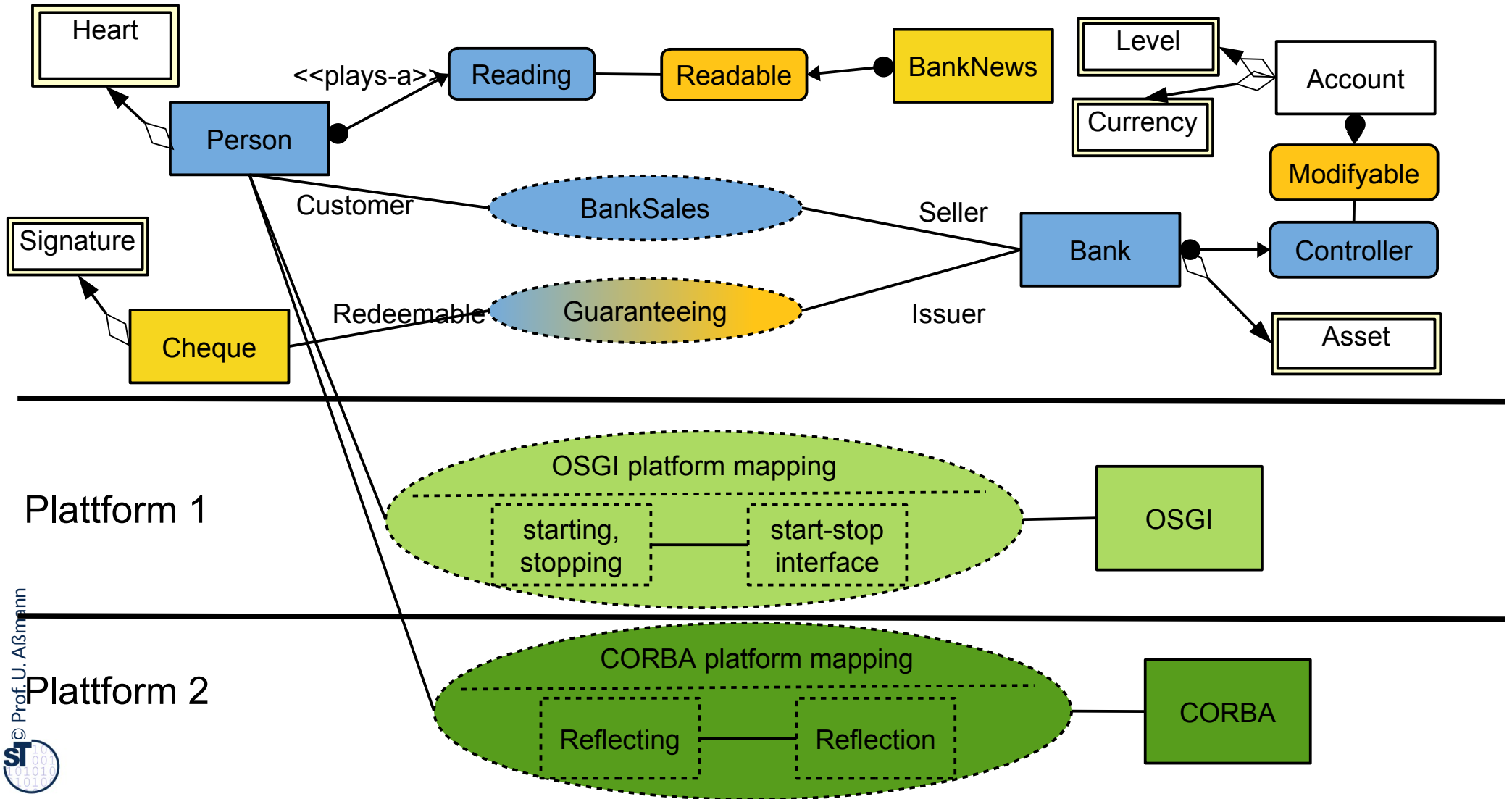


- In der Klasse CORBA:Object wird elementare Funktionalität einer CORBA Komponente definiert
  - heterogen benutzbar über viele Sprachen hinweg
- CORBA untertützt Reflektion:
  - get_interface liefert eine Referenz auf ein "Schnittstellenobjekt"
  - get_implementation eine Referenz auf eine "Implementierung"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

© Prof. U. Aßmann

Mit Verfeinerung durch mehrere Plattform-Konnektoren verschiedener Plattformen

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



© Prof. U. Aßmann

Das Portabilitätsgesetz

Kapselt man Plattformabhängigkeiten in einen Plattformkollaboration, 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 Plattformkollaborationen entwickelt werden.

© Prof. U. Aßmann

Verfeinerungsgesetze im Entwurf

Querscheidende Verfeinerung im Entwurf nutzt **Tool-Tool-Kollaborationen**, **Tool-Materal-(TAM-)Kollaborationen, Plattform-Kollaborationen.**

Man kapselt **Tool-Tool-Kollaborationen**, um sie auf der C-Schicht (Anwendungslogik) später variieren und erweitern zu können.

Man kapselt **Tool-Material-Kollaborationen**, um sie zwischen der C-Schicht (Anwendungslogik) und der D-Schicht (Data) später variieren und erweitern zu können.

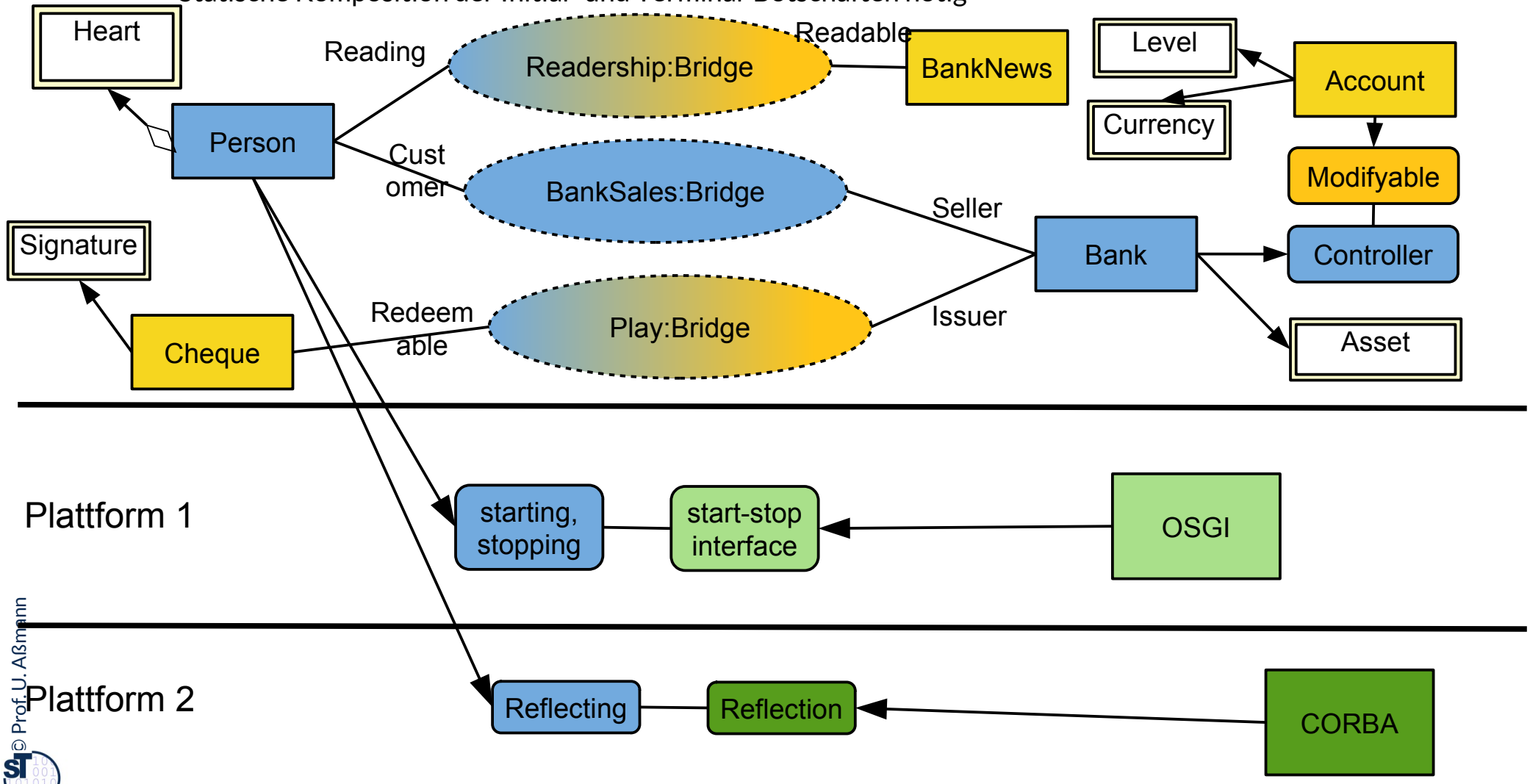
© Prof. U. Aßmann

42.4 Feinentwurf: Abbildung der Kollaborationen auf klassische Programmiersprachen

.. in der Implementierung ..

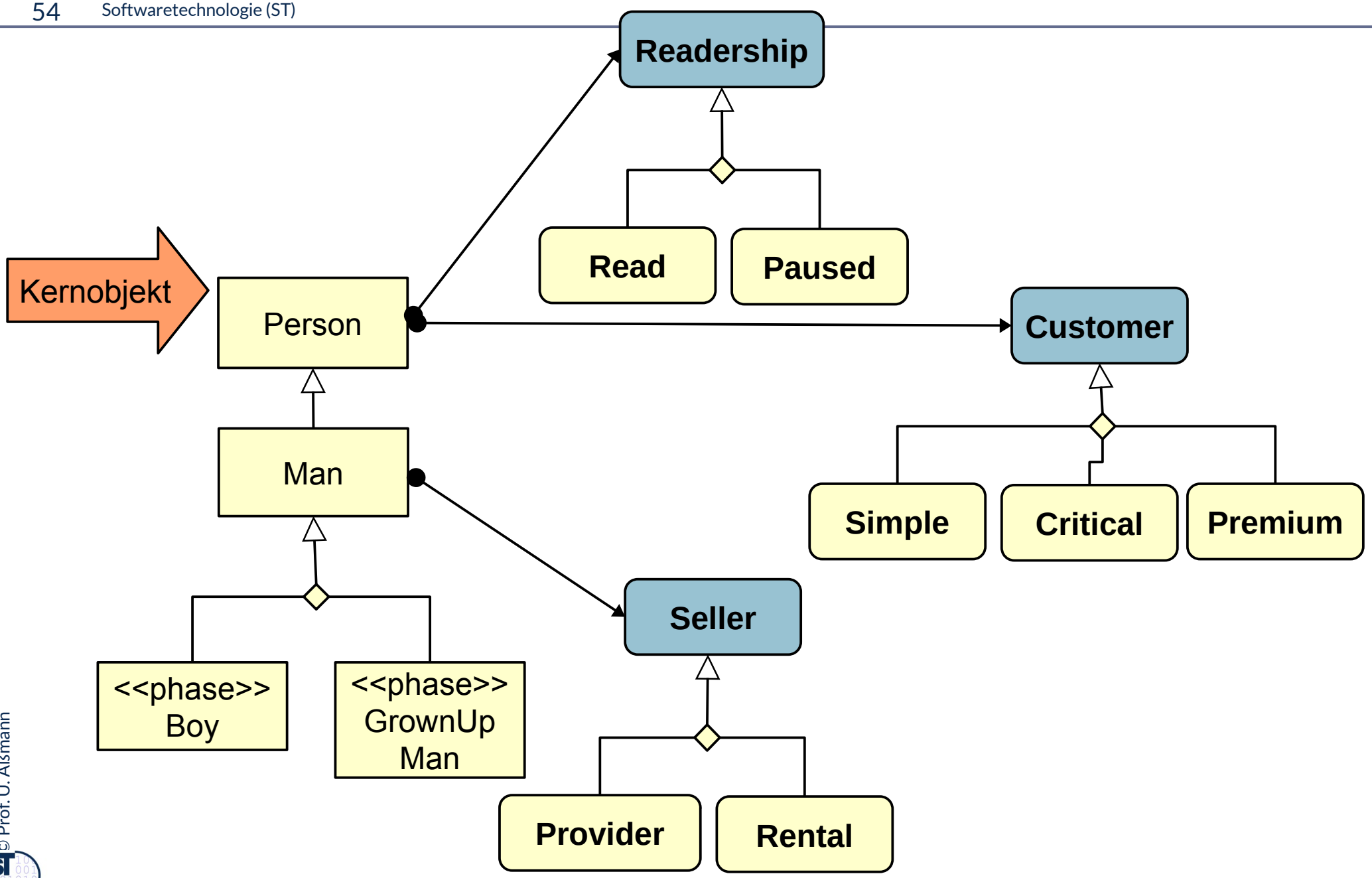
a) Wie bilde ich “plays-a” durch Multi-Bridge (Delegation) ab?

- 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



© Prof. U. Aßmann

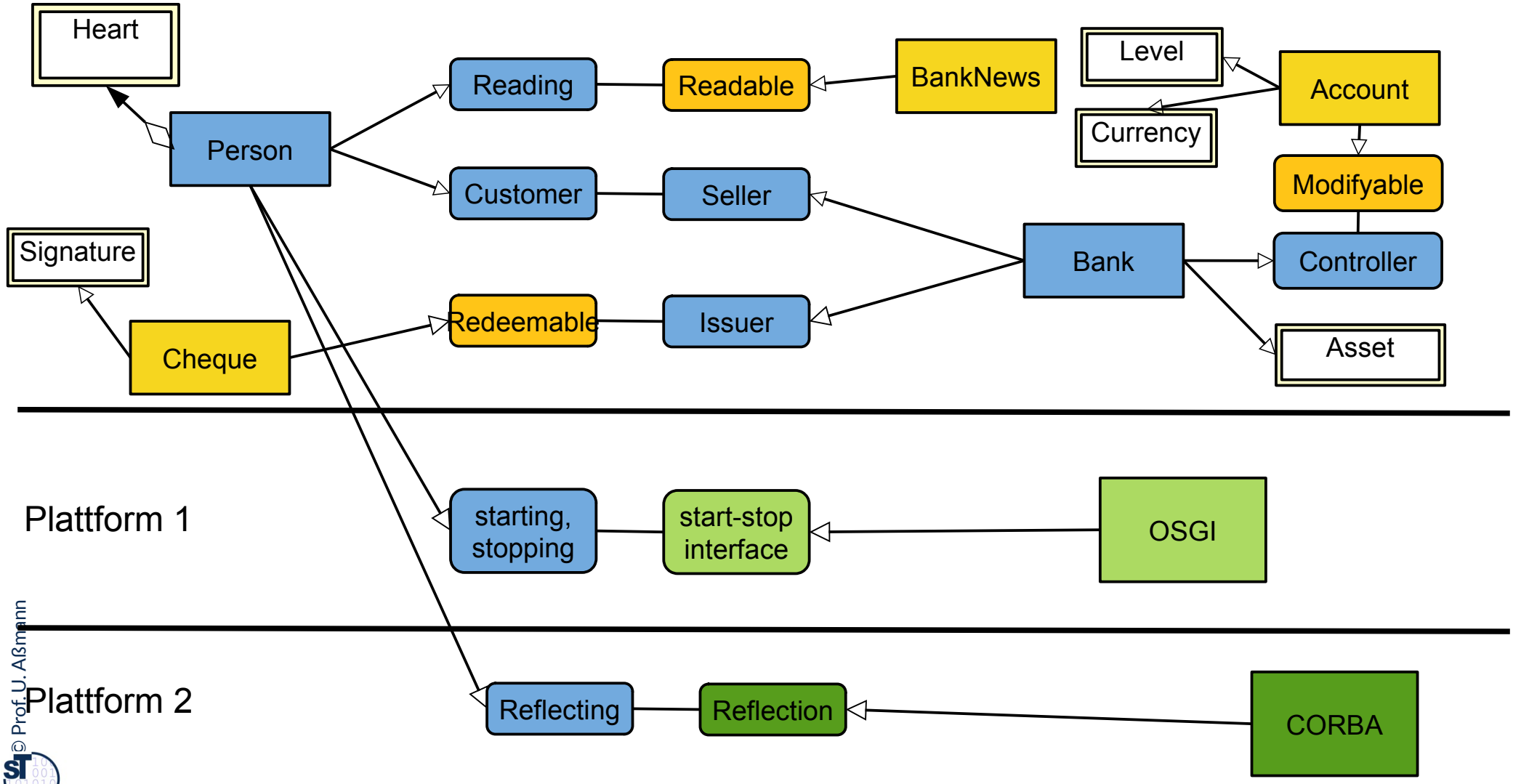
Erinnerung: Realisierung von Rollen mit Multi-Bridge



© Prof. U. Aßmann

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

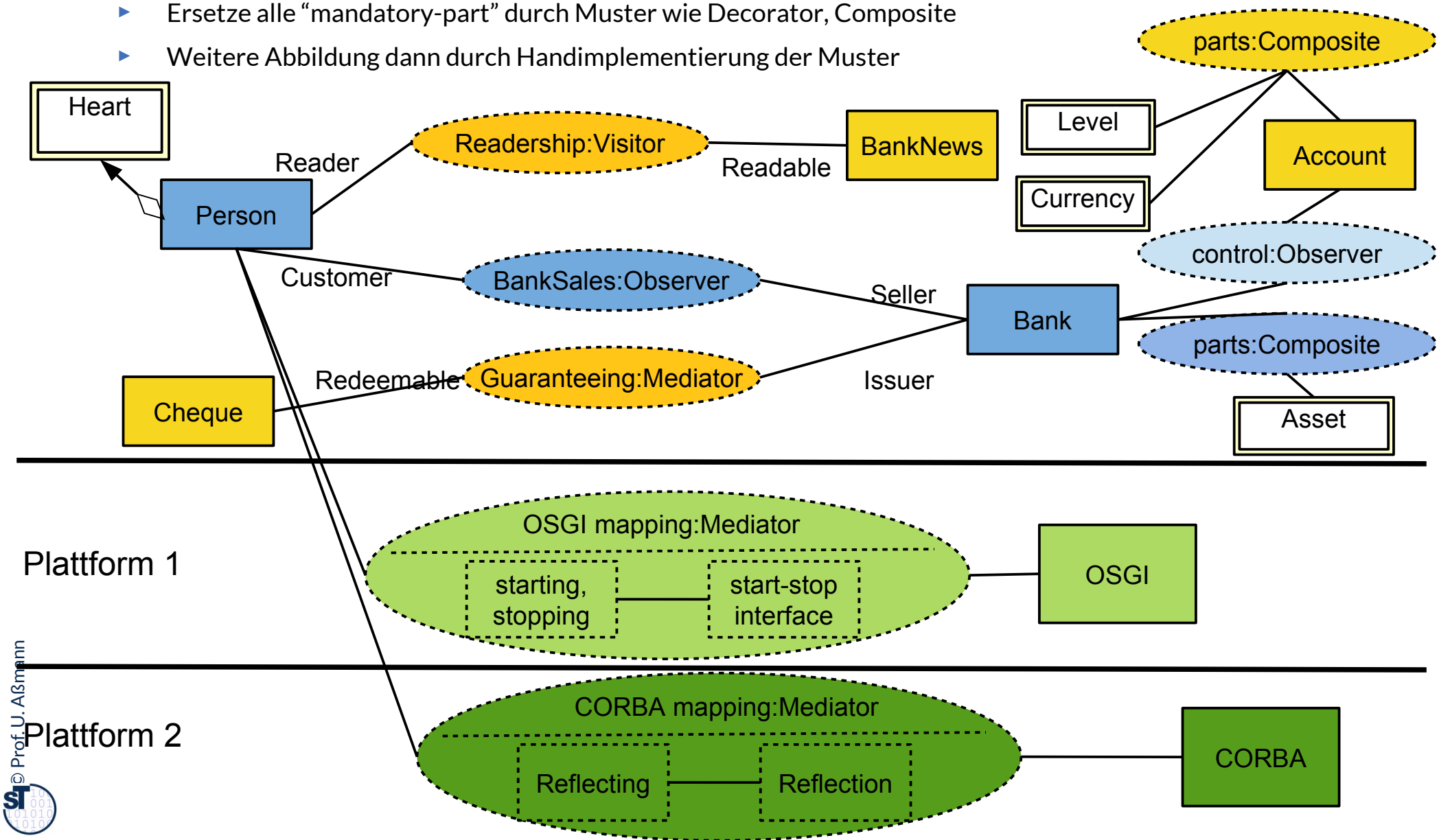
- Ersetze alle “plays”, “mandatory-part”, etc. durch Vererbung (Mehrfachvererbung oder “mixin inheritance”)



© Prof. U. Aßmann

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

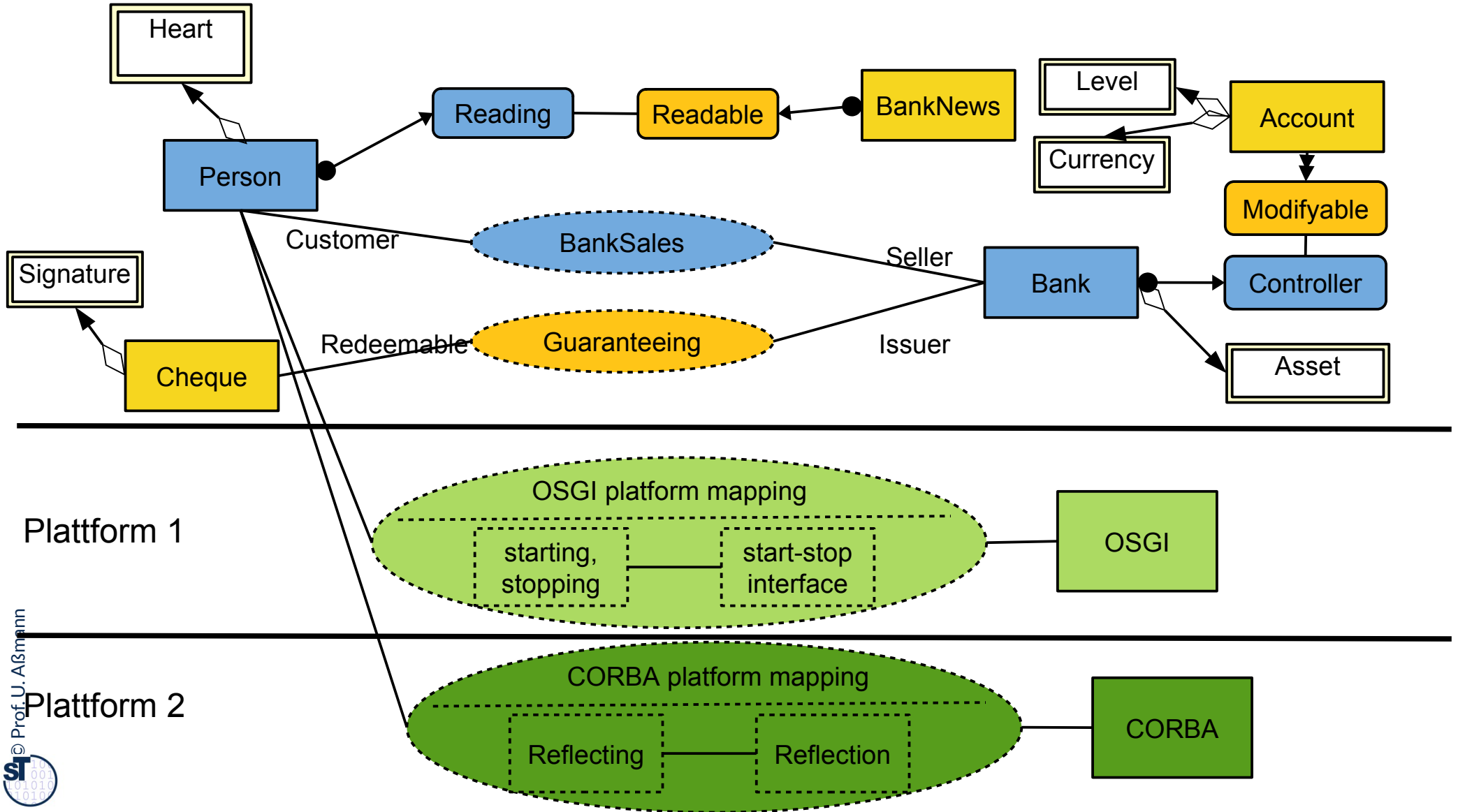
- Ersetze alle “plays”, etc. durch Muster wie Observer, Visitor
- Ersetze alle “mandatory-part” durch Muster wie Decorator, Composite
- Weitere Abbildung dann durch Handimplementierung der Muster



© Prof. U. Aßmann

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

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



© Prof. U. Aßmann

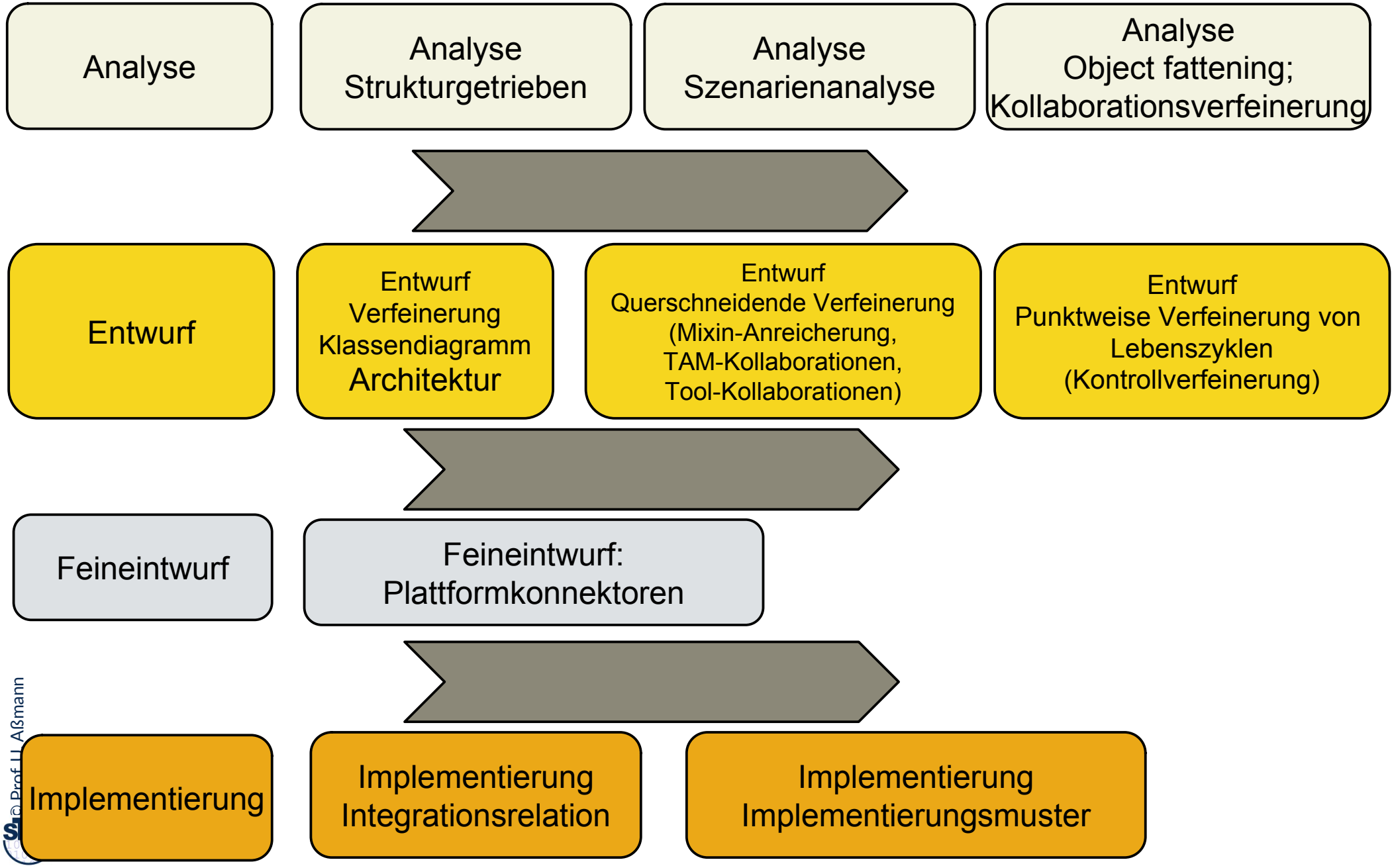
e) Wie bilde ich “integrates” durch Transformation ab?

- Ersetze alle “integrates”, “plays”, etc. durch *Transformationsregeln*
- Führt auf *Modellgetriebene Architektur (model-driven architecture, MDA)*
- Weiter in den Kursen Softwaretechnologie-II und Model-Driven Software Technology (MOST)

© Prof. U. Aßmann

42.5 Gesamtbild der Verfeinerung

Gesamtbild der Verfeinerung



© Prof. U. Aßmann

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?
- Was unterscheidet eine TAM-Kollaboration von einer Tool-Kollaboration?
- Was unterscheidet eine TAM-Kollaboration von einem Tool-Konnektor?
- Was unterscheidet eine TAM-Kollaboration von einer Plattform-Kollaboration?
- Was unterscheidet eine Plattform-Kollaboration von einem Plattform-Konnektor?
- 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 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

© Prof. U. Aßmann

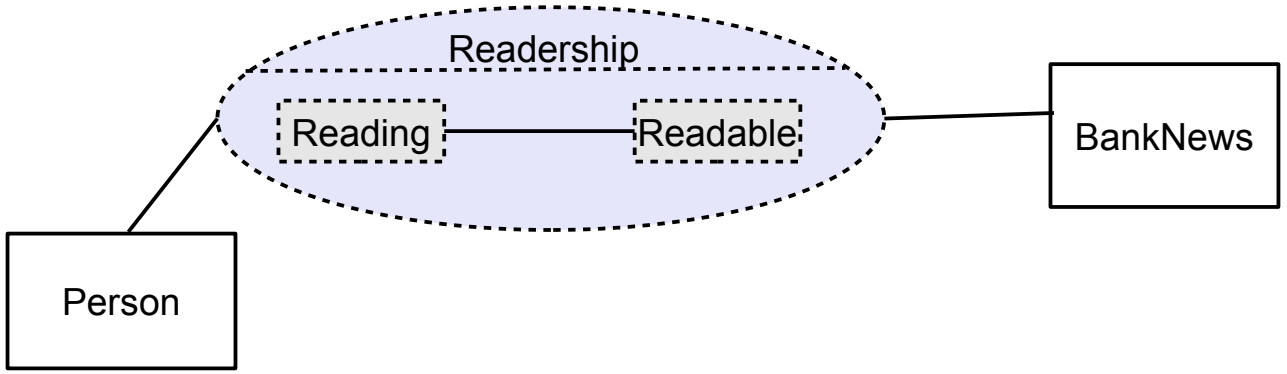
Anhang A: Nebenbemerkung

- 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

© Prof. U. Aßmann

Wdh: Mixin-Anreicherung durch Kollaborationen und Konnektoren

- **Mixin-Anreicherung (Object fattening) durch Unterobjekte, die von Kollaborationen und 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
    - anderen Objekten
    - Plattformen (middleware, Sprachen, Komponenten-services)



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

© Prof. U. Aßmann

Wdh: Querschneidende Verfeinerung durch Mixin-Anreicherung

Querschneidende **Mixin-Anreicherung**
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

Def.: Kann eine Kollaboration durch eine Klasse gekapselt werden, spricht man von einer **Teamklasse (Konnektor).**

© Prof. U. Aßmann