35 Szenarienanalyse mit Anwendungsfalldiagrammen und Kollaborationen (Querschneidende dyn. Modellierung)

Prof. Dr. rer. nat. Uwe Aßmann

Institut für Software- und Multimediatechnik

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

TU Dresden

Version 17-0.1, 20.05.17

1) Anwendungsfalldiagramme
2) Szenarienanalyse mit Interaktionsdiagrammen
3) Szenarienanalyse mit Kommunikationsdiagrammen
4) Szenarienanalyse mit Aktionsdiagrammen
5) Konnektoren
6) Wozu braucht man querschneidende Verfeinerung?

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Obligatorische Literatur

- ST für Einsteiner, Kap. Anwendungsfalldiagramme, Sequenzdiagramme, Aktivitätsdiagramme, Zustandsdiagramme
- Zuser, Kap. 7-9, insbes. 7.3+7.5
- Störrle Kap 9, Kap 12, Störrle 5.3, 5.4

Weitere Literatur

- Die Beispiele zur Servicestation finden sich in
  - S. Pfleeger, Software Engineering. Theory and Practice. Prentice-Hall.
- L. Maciaszek. Requirements Analysis and System Design – Developing Information Systems with UML. Addison-Wesley.
- Giancarlo W. Guizzardi. Ontological foundations for structure conceptual models. PhD thesis, Twente University, Enschede, Netherlands, 2005.
- Nicola Guarino, Chris Welty. Supporting ontological analysis of taxonomic relationships. Data and Knowledge Engineering, 39:51-74, 2001.

Überblick Teil III: Objektorientierte Analyse (OOA)

1. Überblick Objektorientierte Analyse
   1. Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodellgetriebene Modellierung mit UML für das Domänenmodell
   1. Strukturelle metamodellgetriebene Modellierung
      1. Modellierung von komplexen Objekten
   2. Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen (Verhaltensmodell)
   1. Funktionale Verfeinerung: Dynamische Modellierung von Lebenszyklen mit Aktionsdiagrammen
   2. Funktionale querschneidende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen (Kap 34, **35**)
4. Beispiel Fallstudie EU-Rent

Wdh. Punktweise und querschneidende dynamische Verfeinerung

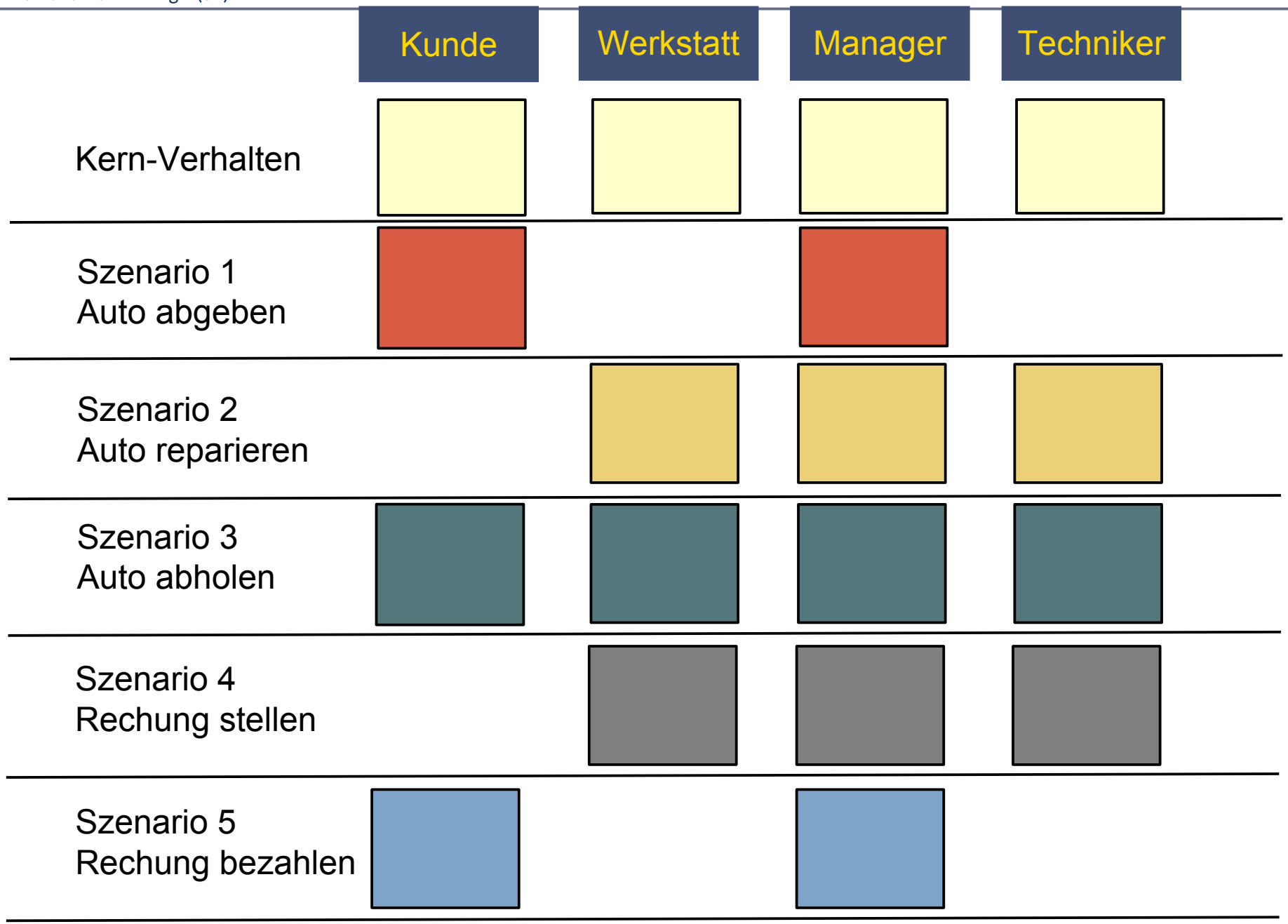
Punktweise funktionale Verfeinerung ist eine funktionale Verfeinerung eines Modellfragmentes (meist Objekt oder Methode), die *punktweise* geschieht, d.h. pro Modellfragment separat durchgeführt wird.

- Ergebnis:
  - **Lebenszyklus** des Objekts
  - **Implementierung** einer Methode

Querschneidende funktionale Verfeinerung ist eine funktionale Verfeinerung *mehrerer* Modellfragmente gleichzeitig, die *querschneidend* geschieht.

- Damit kann man das Zusammenspiel mehrerer Objekte oder Methoden untersuchen, eine *Szenarienanalyse*, die quasi die Draufsicht auf ein Szenario ermittelt
- Die querschneidende Verfeinerung geschieht dadurch, dass die beteiligten Objekte durch Kollaborationen erweitert werden, die Rollen-Satelliten anlagern
- Querscheidende Verfeinerung baut auf das Entwurfsmuster Bridge auf

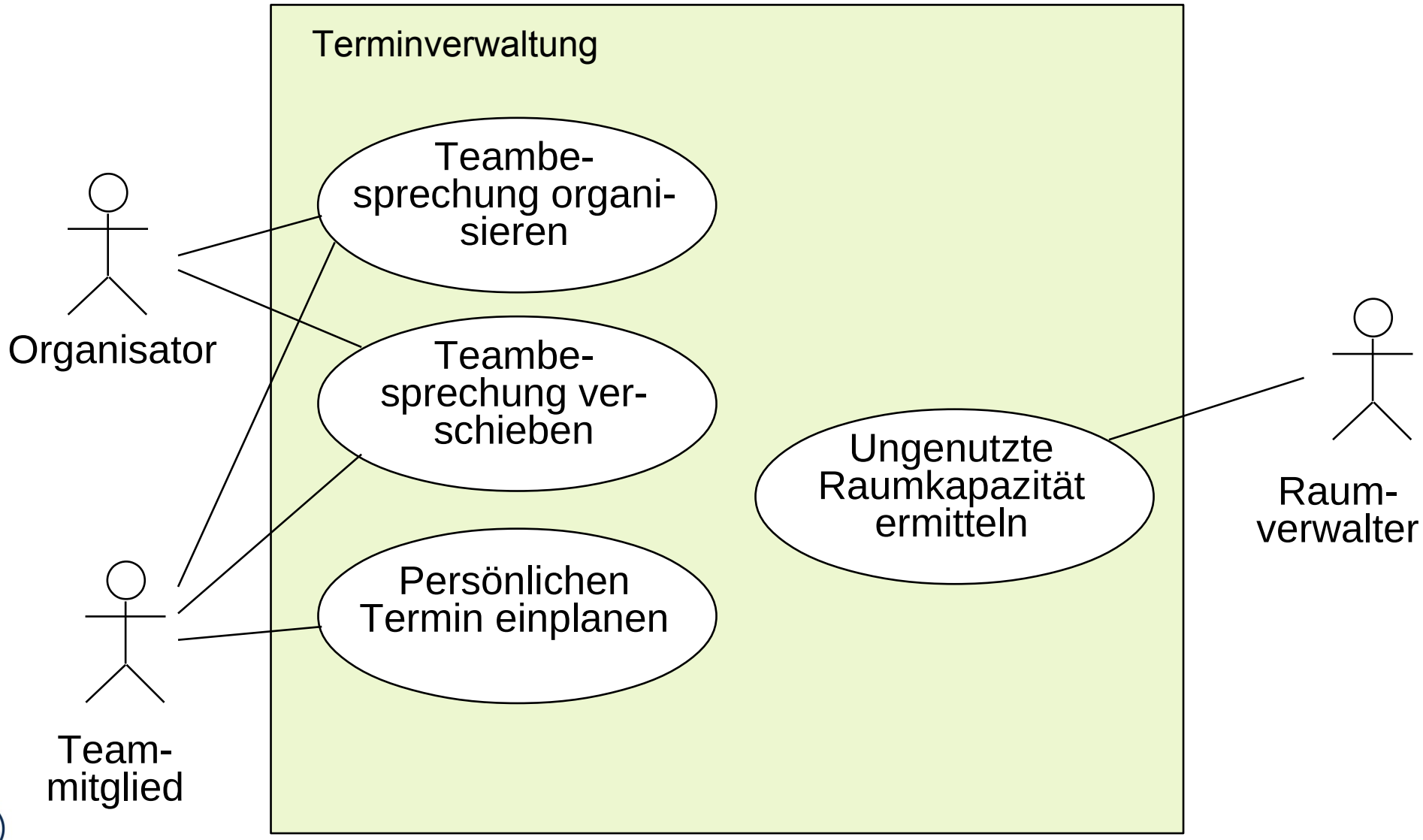
Querschneidende Verfeinerung (Objekt-Szenario-Matrix)



35.1 Nutzfalldiagramme (Anwendungsfalldiagramme)

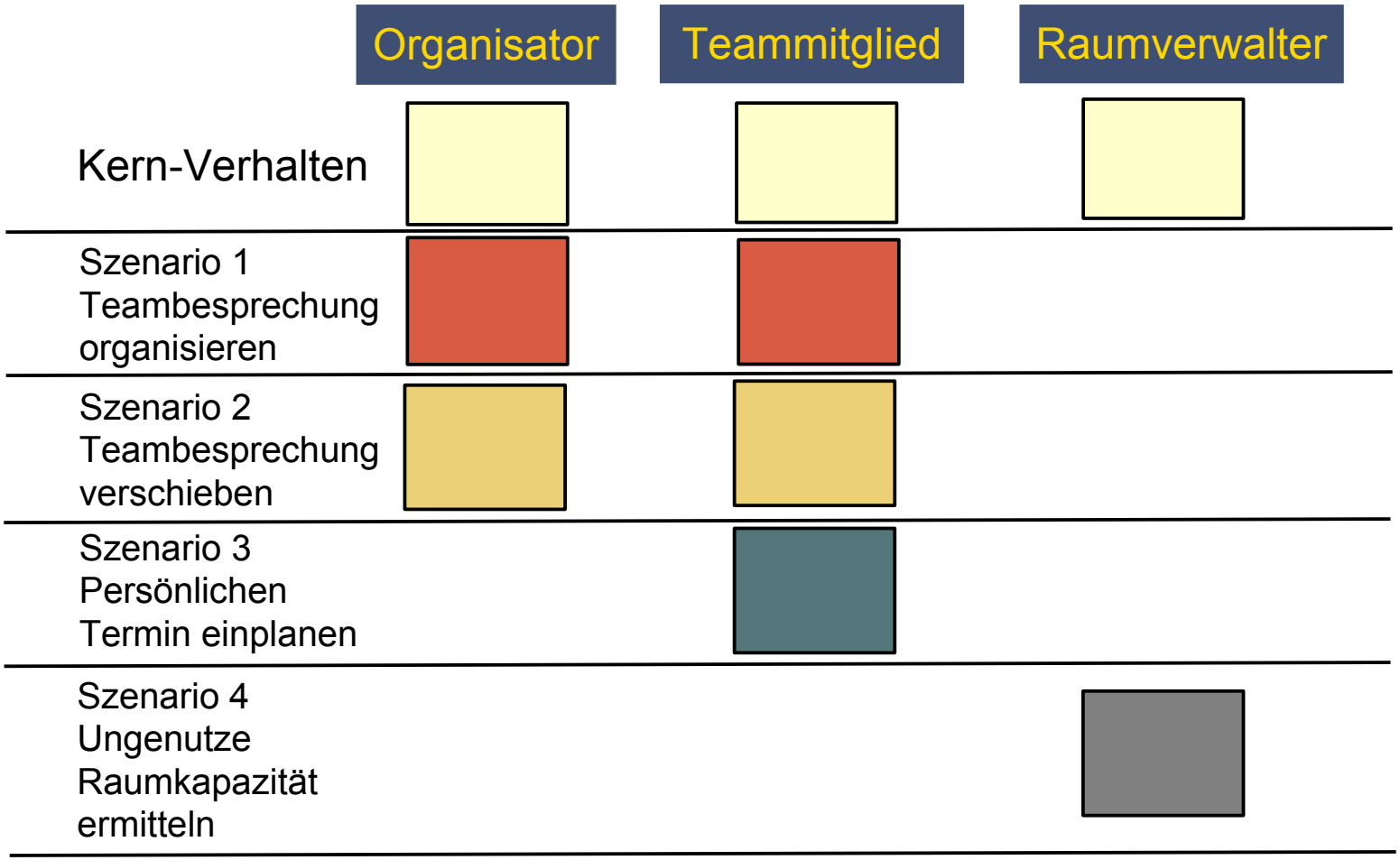
UML-Anwendungsfall-Diagramm (Nutzfall-, Use-Case-Diagramm)

- Ein Anwendungsfall beschreibt die Interaktion (Kollaboration) der Akteure mit dem System (*querschneidend* durch das System)



Übung

- Erstellen Sie von allen Anwendungsfalldiagrammen dieses Kapitels eine Objekt-Szenario-Matrix, die die Beteiligung der Objekte an den Szenarien festhält.

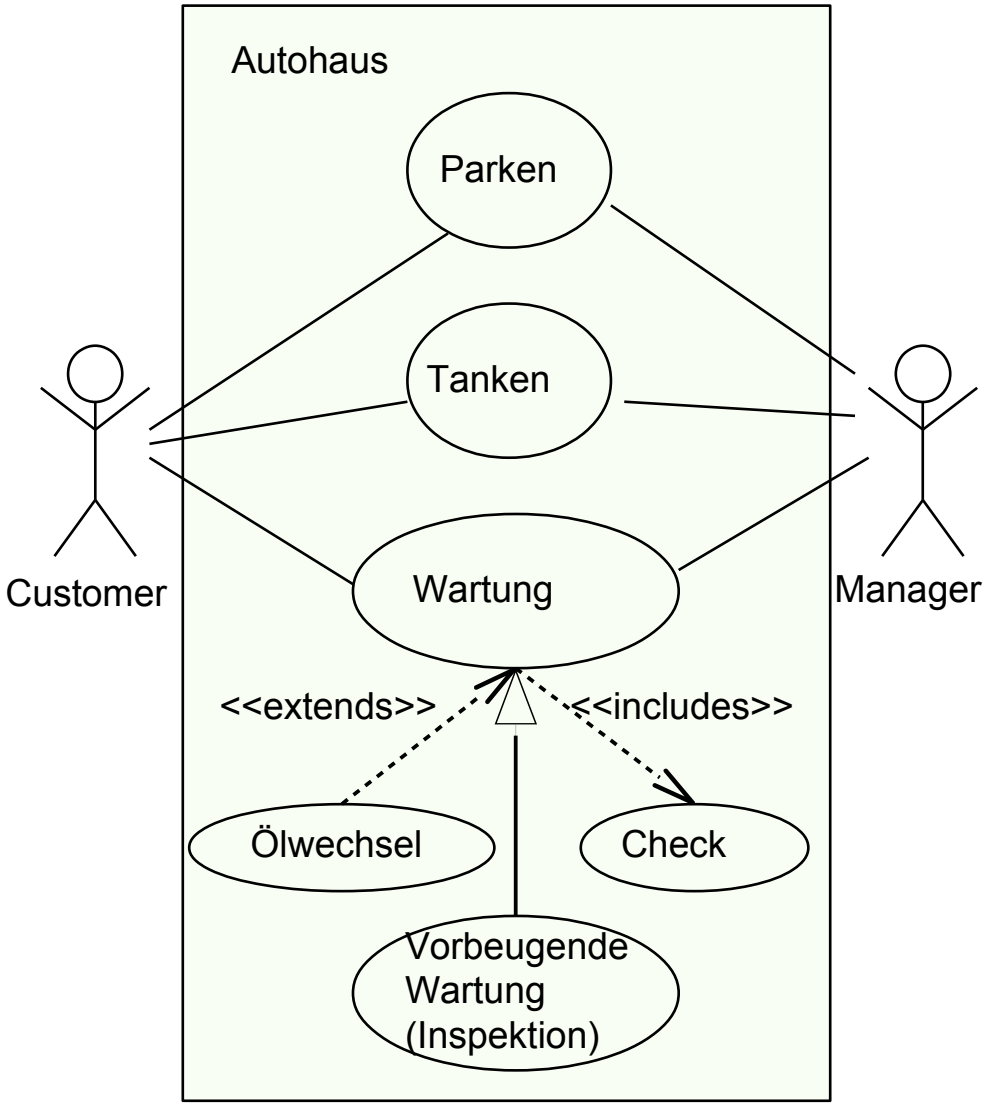


Anwendungsfälle (Nutzfälle)

- **Anwendungsfälle** beschreiben funktionale Anforderungen an ein System
  - Anwendungsfälle sind Funktionen des Systems *im Kontext* von Anwendern (Aktoren)
- Aus den Anwendungsfällen setzt man mittels ***Szenarienanalyse*** dann zusammen:
  - das **Kontextmodell** mit der Schnittstelle des Systems, also die sichtbaren Funktionen des Systems
  - die **Top-Level-Architektur**, die die Realisierung der sichtbaren Funktionen des Systems auf oberster Ebene zeigt
  - **Kollaborationen** zwischen Objekten

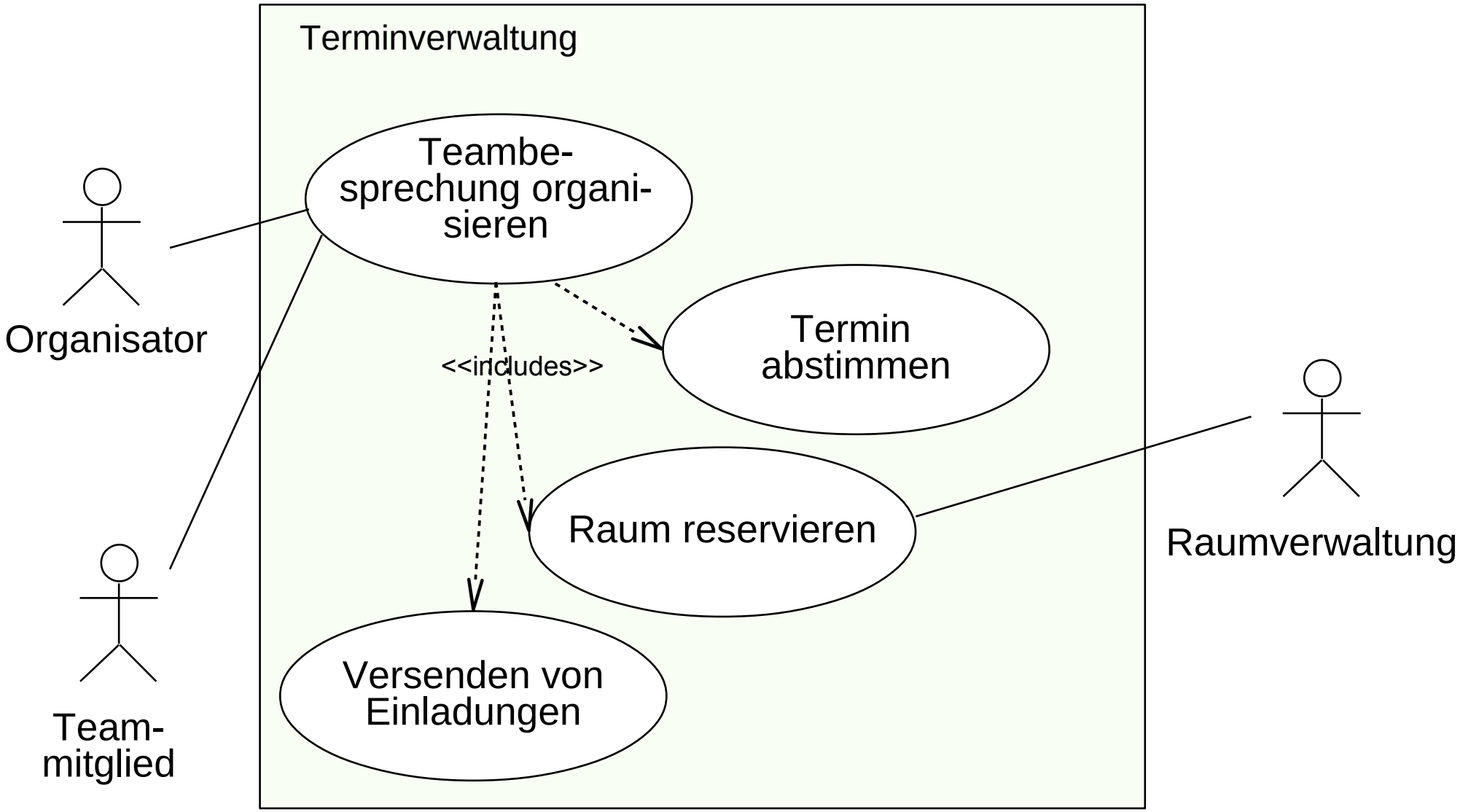
Verallgemeinerung, Erweiterung und Aufruf von Anwendungsfällen

- Die Vererbungsrelation zwischen Anwendungsfällen beschreibt Generalisierung bzw. Spezialisierung
  - Hier: *Wartung* ist allgemeiner als *Vorbeugende Wartung*
- Die ***Includes***-Relation beschreibt Bestandteile der Aktionen (Aufrufbeziehung zwischen Aktionen)
  - Hier: *Wartung* beinhaltet *Check*
- Die ***Extends***-Relation beschreibt *optionale* Erweiterungen
  - Hier: *Ölwechsel* kann Teil von *Wartung* sein



[nach Pfleeger]

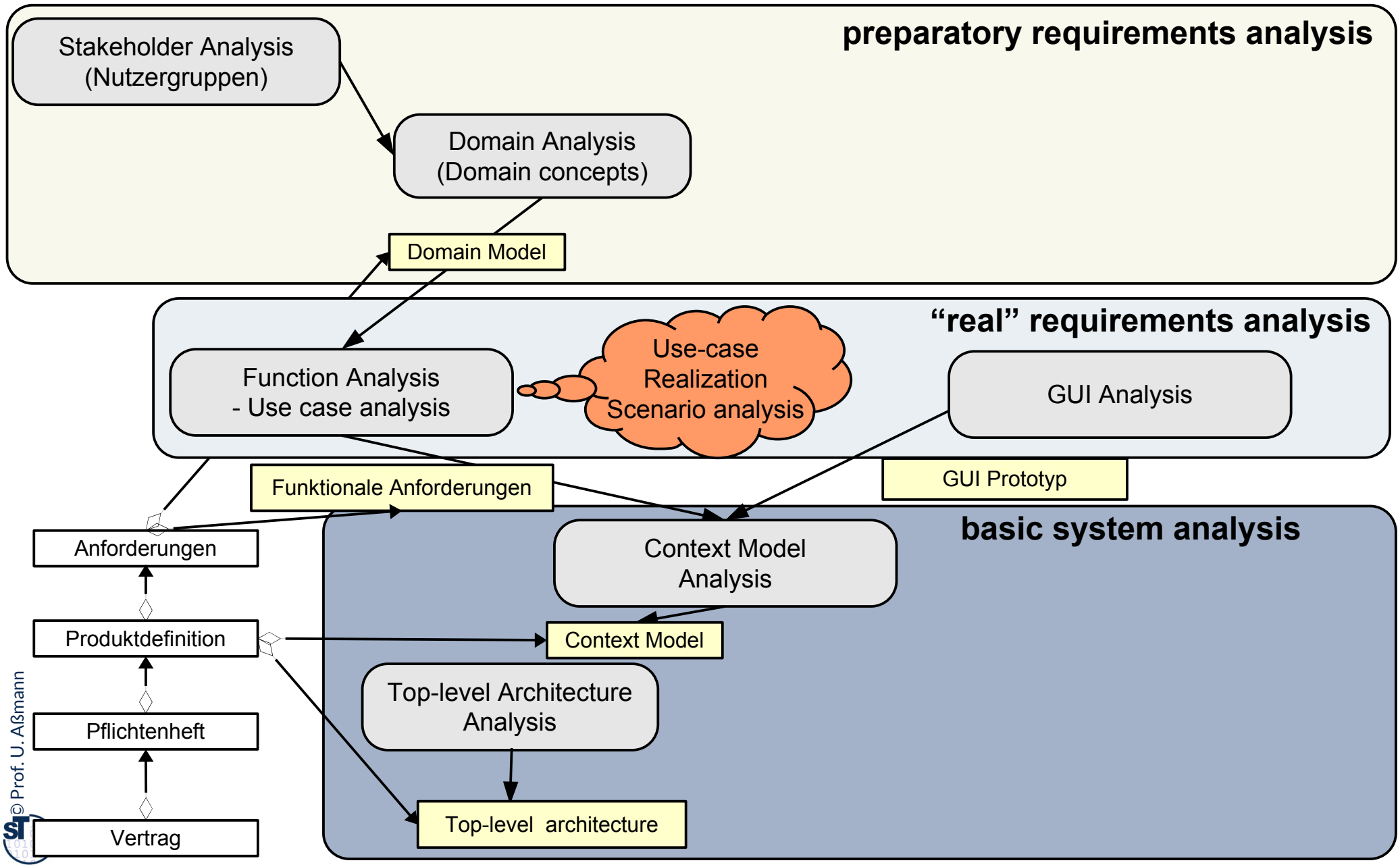
Verfeinerung des Anwendungsfalls „Teambesprechung organisieren”



35.2 Szenarienanalyse - Ableitung von Kollaborationen aus Anwendungsfällen

Anwendungsfallrealisierung, use case realization

Erinnerung: Schematischer Ablauf der Analyse



Wege der Szenarienanalyse (use case realization analysis)

- Die Methode der ***Anwendungsfallrealisierung*** *(use case realization, Szenarienanalyse, scenario analysis)* wird verwendet, um:
  - **Systemanalyse** des Kontextmodells und der Top-Level-Architektur
  - **Querschneidende Verfeinerung** durch mehrere Klassen/Objekte durchzuführen, in dem Kollaborationen und Konnektoren für die Objektverfettung abgeleitet werden
- Anwendungsfallrealisierung nutzt zur Verfeinerung verschiedene Interaktionsdiagramme **mit Schwimmbahnen**:
  - Verfeinere Anwendungsfalldiagramm mit Interaktionsdiagrammen
    - mit Sequenzdiagramm (sequence diagram, sequence chart)
    - mit Kommunikationsdiagramm (communication diagram)
  - Verfeinere Anwendungsfalldiagramm mit Aktionsdiagrammen
    - mit Schwimmbahnen im Aktivitätsdiagramm
    - mit einem Netz von kommunizierenden Verhaltens-(Zustands-)maschinen
- Wie arbeitet man mit dem Kunden?
  - Verfeinerung geschieht zusammen in Abstimmung

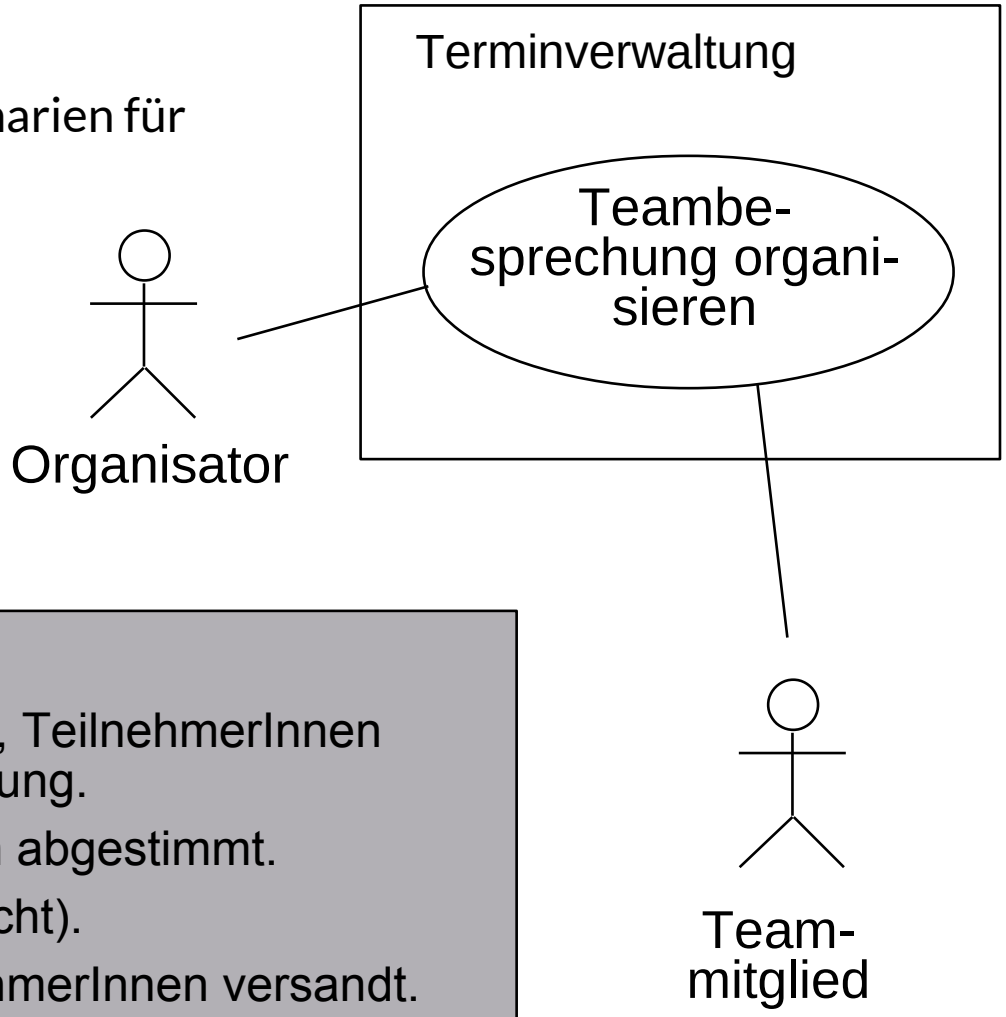
Szenarien

- **Definition:** Ein ***Szenario*** ist eine Beschreibung einer beispielhaften Folge von *Interaktionen* von Akteuren mit dem System zur Beschreibung eines Anwendungsfalls (use case realization).
  - Es gibt Szenarien für Normalfälle ('gut-Fälle'), Ausnahmefälle ('exception case') und Fehlerfälle ('negativ'-Fall).
- Szenarien spielen Anwendungsfälle durch
  - ermittle zeitliches Zusammenspiel, verfeinere über der Zeit
  - ermittle feinere Aktionen und binde sie mit Vererbung ein
  - ermittle Unteraktionen und binde sie mit <<includes>> ein
  - ermittle optionale Erweiterungen von Aktionen und binde sie mit <<extends>> ein
- Szenarien können durch CRC-Rollenspiel ermittelt werden
- Wähle als Szenariobeschreibung durch Interaktionsdiagramme oder Aktionsdiagramme
  - Leite daraus eine Kollaboration ab (Konnektor, Team)

Szenarienanalyse

- Die Szenarienanalyse beginnt mit Anwendungsfällen und analysiert das Zusammenspiel der Akteure
- **Beispiel:**

  Durchspielen eines der Normalfall-Szenarien für 'Teambesprechung organisieren'

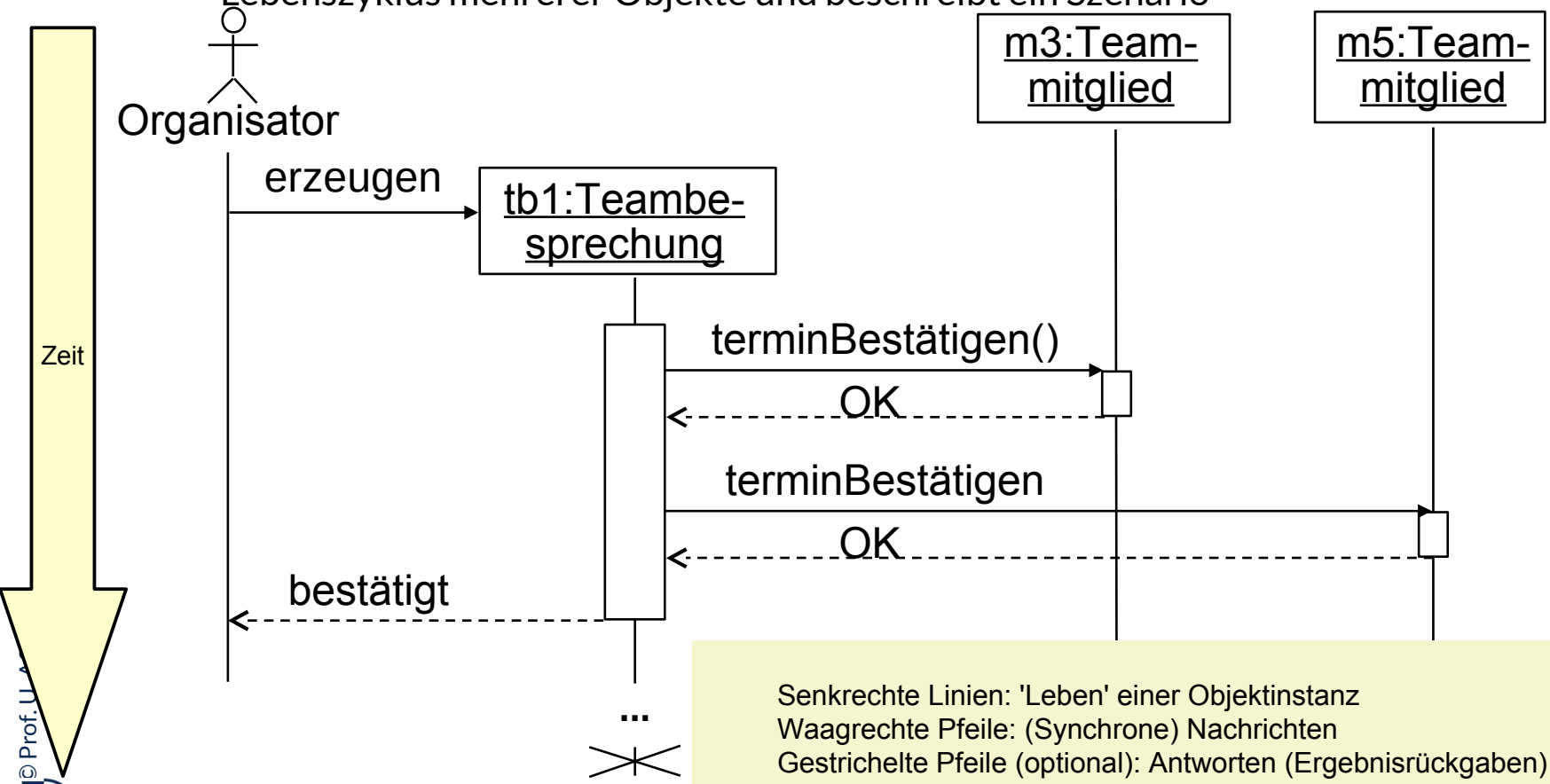


Durchspielen:

- Organisator erfährt Thema, Termin, TeilnehmerInnen einer neu geplanten Teambesprechung.
- Zeitpunkt wird mit TeilnehmerInnen abgestimmt.
- Raum wird reserviert (falls gewünscht).
- Einladungen werden an die TeilnehmerInnen versandt.

35.2.1 Szenarienanalyse mit UML-Sequenzdiagrammen

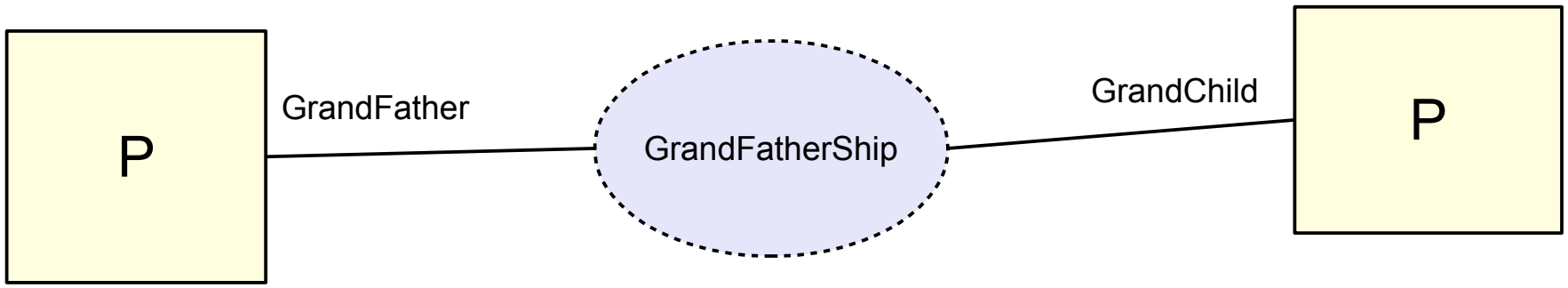
- Ein **Sequenzdiagramm** ist eine **Objekt-Lebenszeit-Matrix**, in der die Objekte von links nach rechts aufgereiht sind und die Zeit von oben nach unten läuft (Objekt-Lebenslinien oder "Schwimmbahnen")
  - Sequenzen von Nachrichten, geordnet durch die Zeit
- Achtung: das Sequenzdiagramm schneidet mit seinen Schwimmbahnen quer durch das Lebenszyklus mehrerer Objekte und beschreibt ein Szenario



Senkrechte Linien: 'Leben' einer Objektinstanz
Waagrechte Pfeile: (Synchrone) Nachrichten
Gestrichelte Pfeile (optional): Antworten (Ergebnisrückgaben)
Blöcke auf den senkrechten Linien: Steuerfokus (Aktivierung)

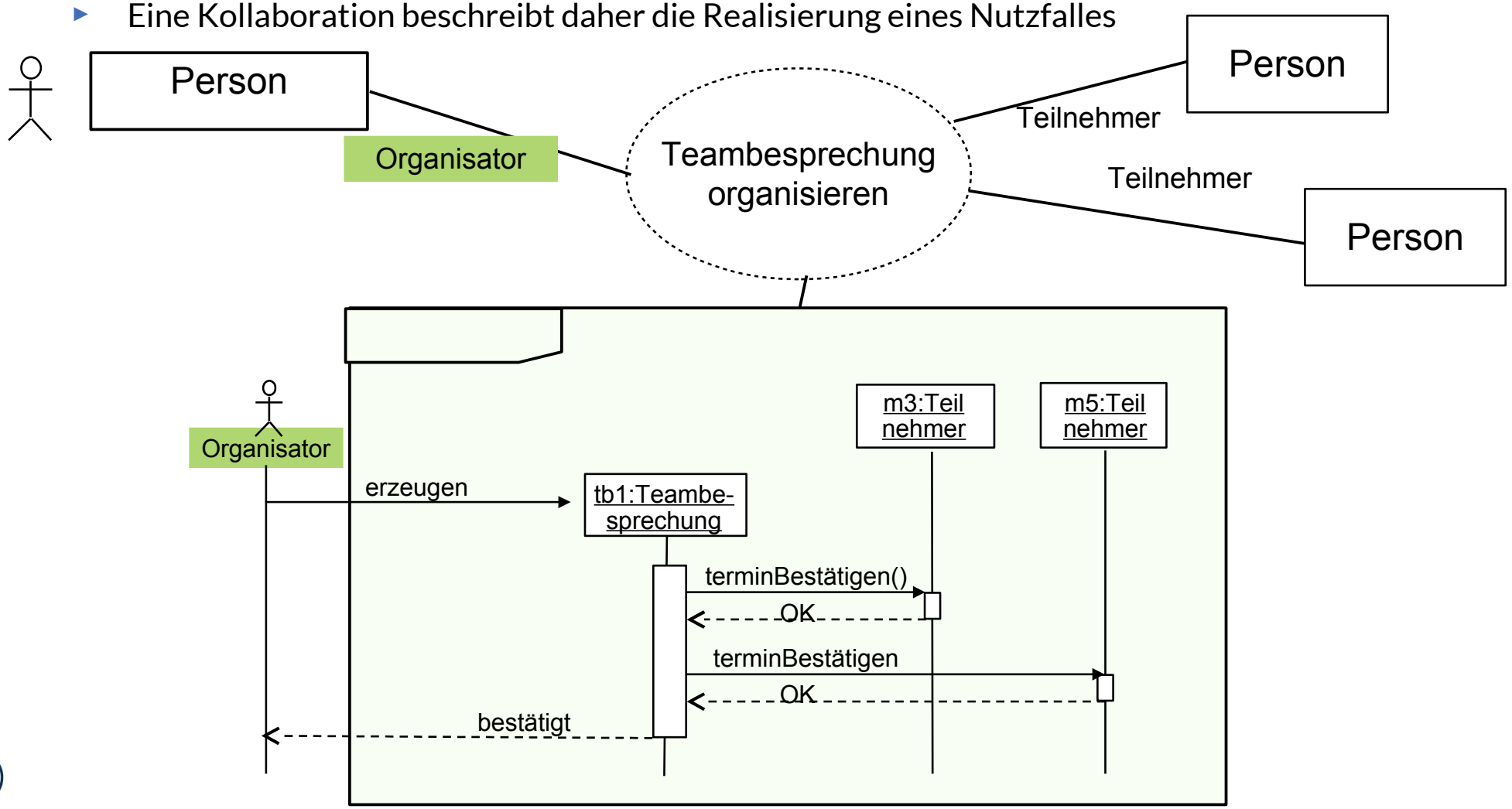
Kollaborationen (collaborations, Teamklassen) in UML

- Eine **Kollaboration** *(team class, collaboration, Rollenmodell) ist ein* Schema für die Zusammenarbeit von Objekten. Sie definiert mehrere Rollen von Spielern (player) im Zusammenspiel
- In UML stellt sich eine Kollaboration dar als
  - generisches Sprachkonstrukt mit Klassen-Parameter P und Rollenname als Bezeichner für Tentakel
  - konkret instantiiert mit Klassen



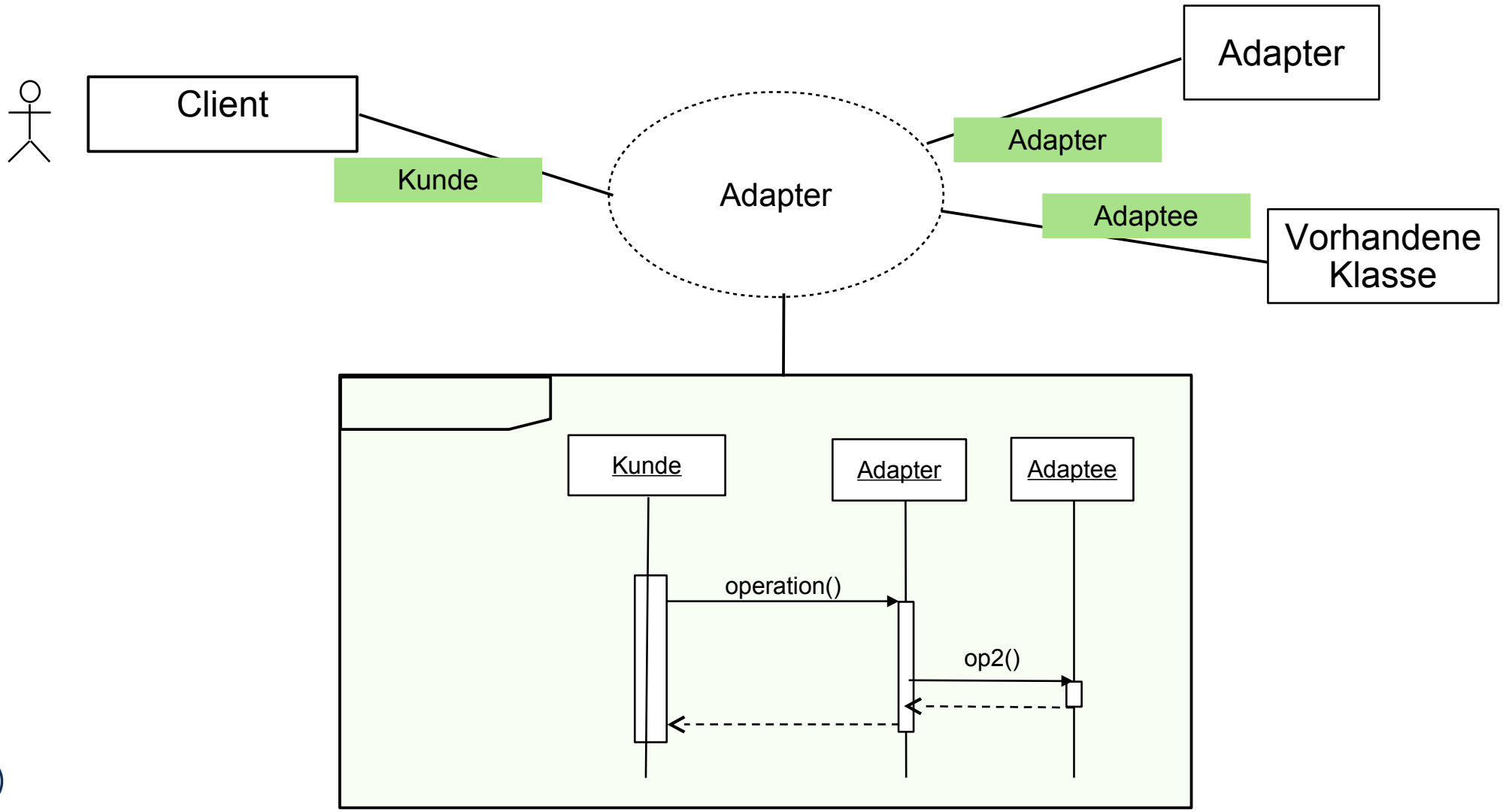
Kapseln eines Szenarios in einer Kollaboration

- Eine **Kollaboration** kann mit einem Sequenzdiagramm als Verhalten unterlegt werden
  - Die einzelnen Lebenslinien geben das Verhalten eines Objekts in der Kollaboration an
- Die Kollaboration beschreibt also ein *Szenario querschneidend durch die Lebenszyklen mehrerer Objekte*
- Eine Kollaboration beschreibt daher die Realisierung eines Nutzfalles



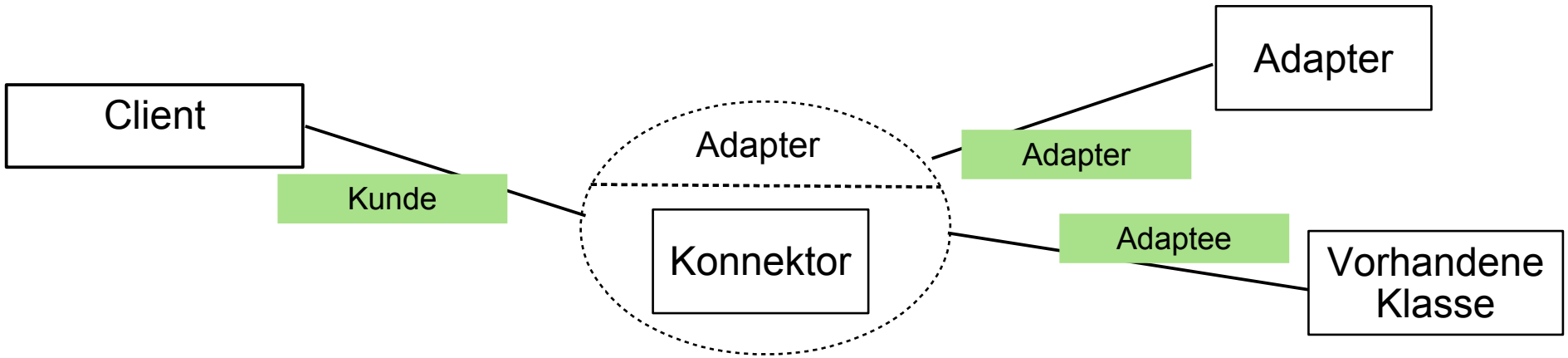
N.B.: Entwurfsmuster werden in UML mit Kollaborationen spezifiziert

- Auch ein Entwurfsmuster wird in UML mit einer **Kollaboration** spezifiziert. Das Gamma-Buch ordnet jedem Entwurfsmuster ein Sequenzdiagramm zu



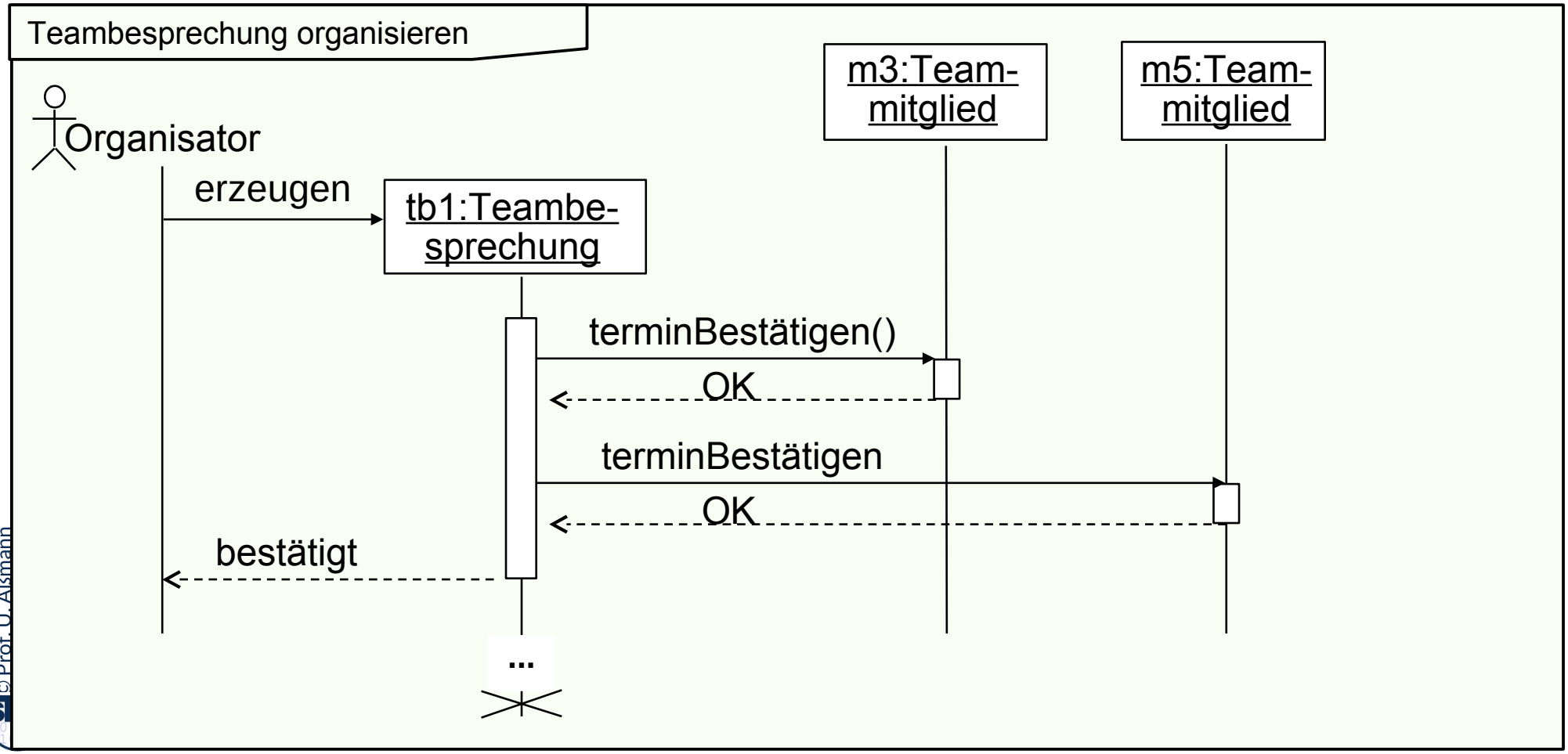
35.2.2 Konnektoren

- Ein **Konnektorobjekt** ist ein Steuerobjekt für eine Kollaboration. Es ist ein Assoziationsobjekt, das aus bisher nicht kooperierenden Objekten ein Netz aufbaut, ihre Kollaboration (Interaktion und Kommunikation) leitet und dann wieder auflöst.
- Eine **Konnektorklasse** ist also eine Kollaborationsklasse, die definiert:
  - **ein Konnektor-Hauptklasse** für das Konnektorobjekt
  - die Spieler als innere Klassen (Rollenklassen)
  - **Netzaufbau-Methoden**, die das Konnektorobjekt mit den Spielern verbinden
  - **Kommunikationsmethoden**, die auf die inneren Objekte delegieren
  - **Kanäle**, die Daten zwischen den Objekten hin- und herschieben
- In Java implementiert man eine Kollaboration immer als **Konnektorklasse**



Einordnung in Kontextmodell und Top-Level-Architektur

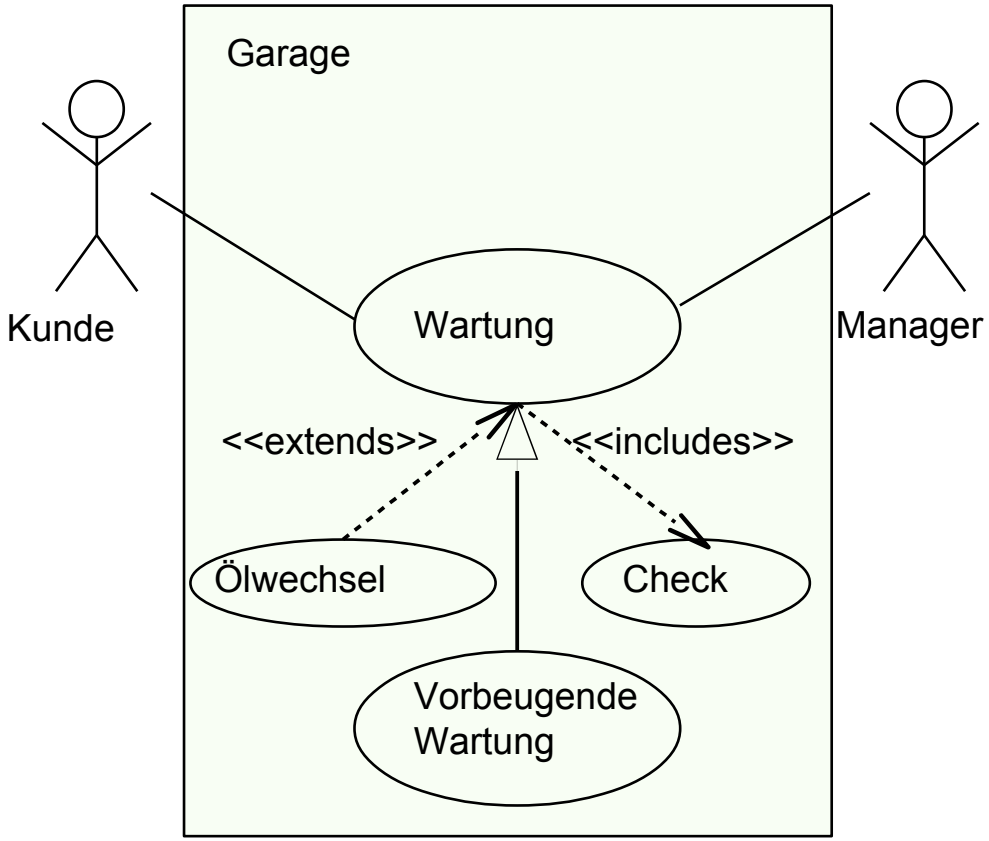
- Nach der Szenarienanalyse muss unterschieden werden, welche Klassen zum Kontextmodell und welche zur Top-Level-Architektur gehören
- Hier: noch alles im Kontextmodell



35.2.2 Beispiel Szenarienanalyse Servicestation

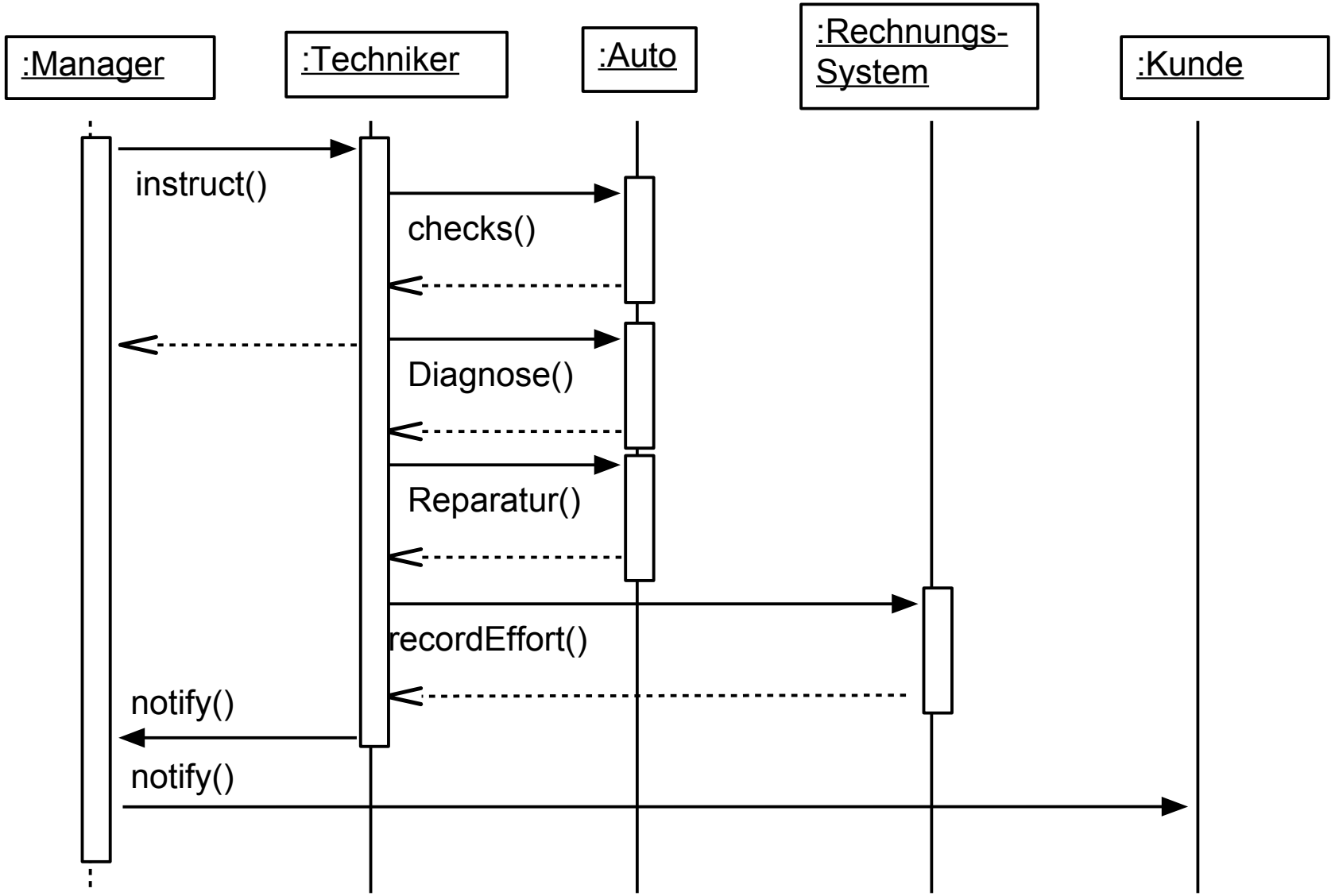
Szenarienanalyse mit Sequenzdiagrammen

- Ausgangspunkt: Anwendungsfall Auto-Wartung (Wartung) [Pfleeger]



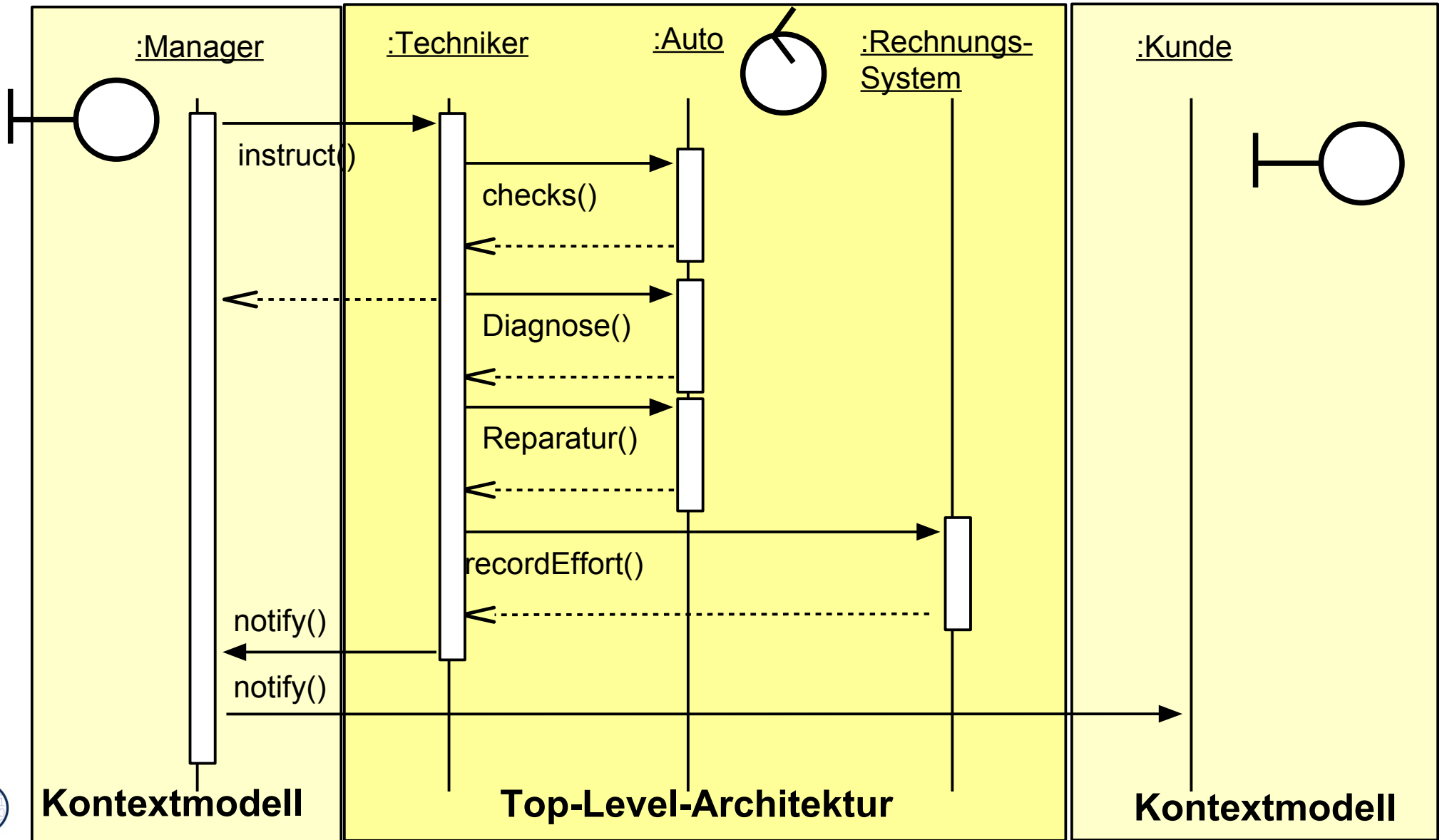
Szenarienanalyse Sequenzdiagram Service-Station

- Sequenzdiagramme werden benutzt zur Analyse von Szenarien mit wenigen Objekten, die viel kommunizieren



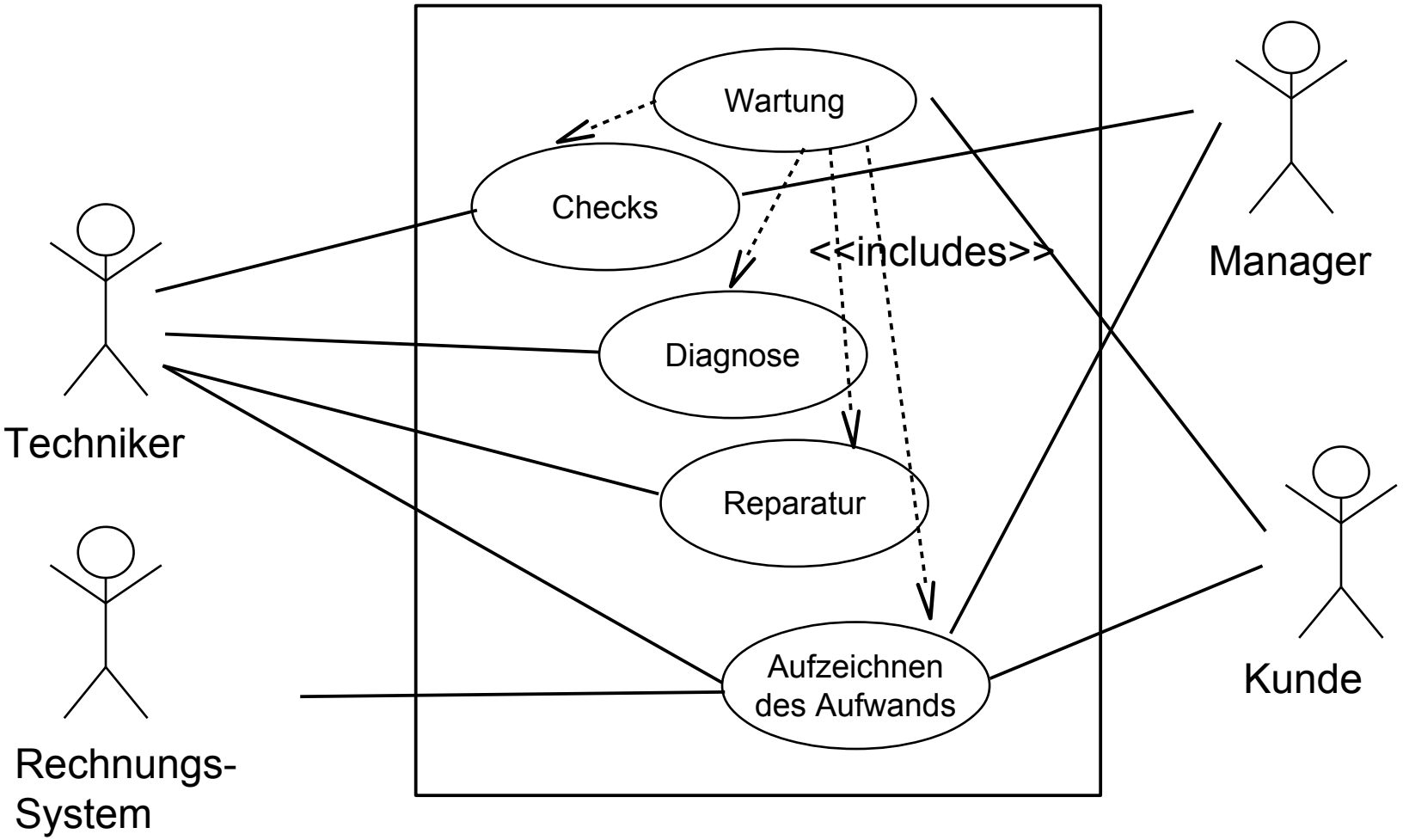
Beziehung zum Kontextmodell und Top-Level-Architektur

- Ein Sequenzdiagramm eines Szenarios muss in die TLA eingeordnet werden: Welche Klassen sind B, C, D?



Verfeinertes Anwendungsfall-Diagramm Service-Station

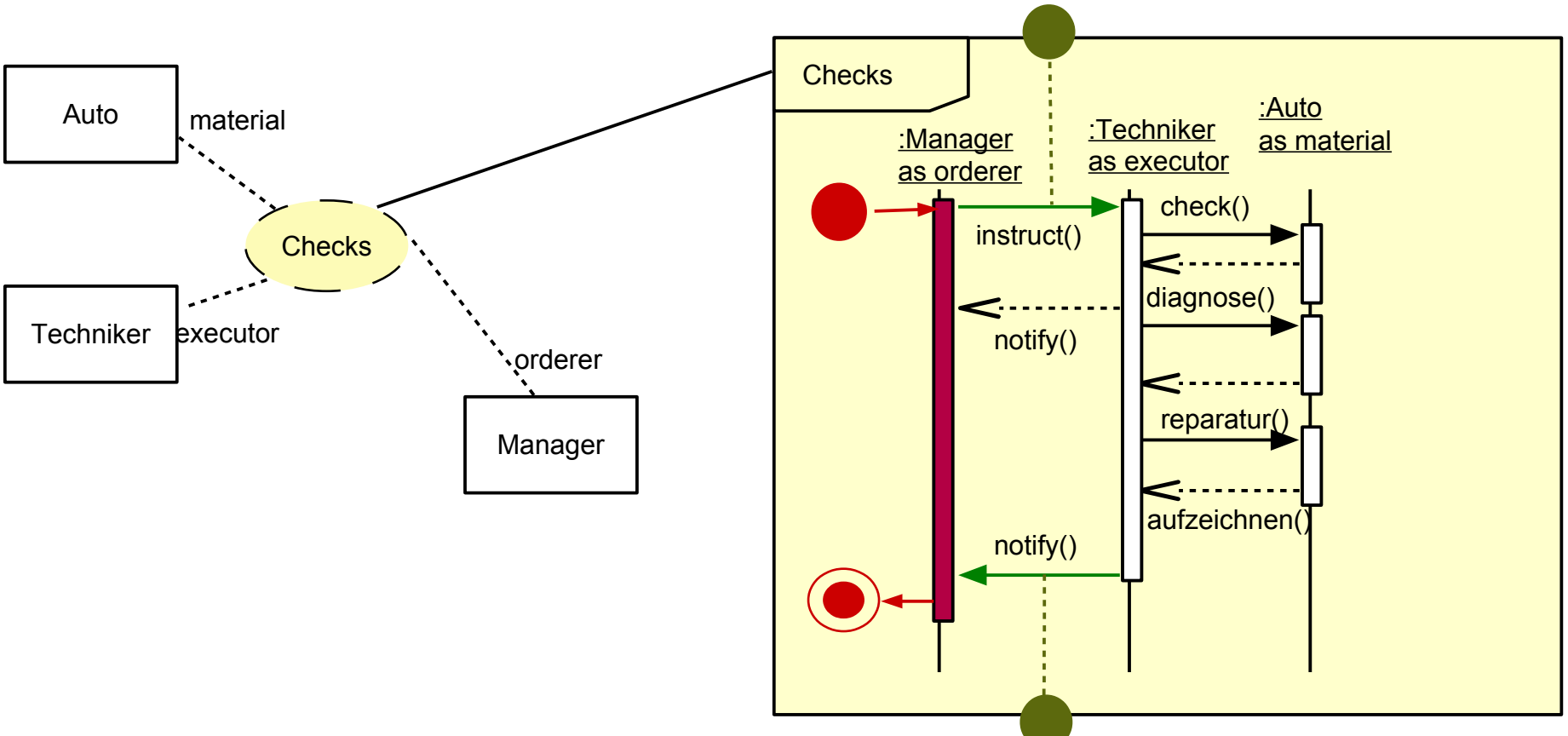
- Aus dem Sequenzdiagramm kann nun ein verfeinertes Anwendungsfalldiagramm erstellt werden



35.2.3 Erstellung von Kollaborationen aus Szenarien

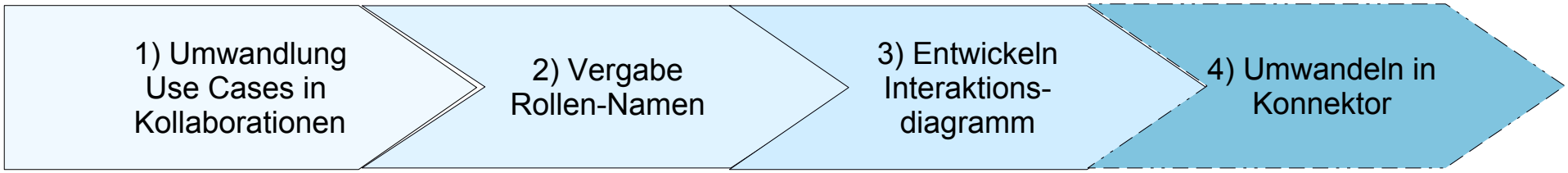
Ableitung von Kollaborationen aus der Szenarienanalyse

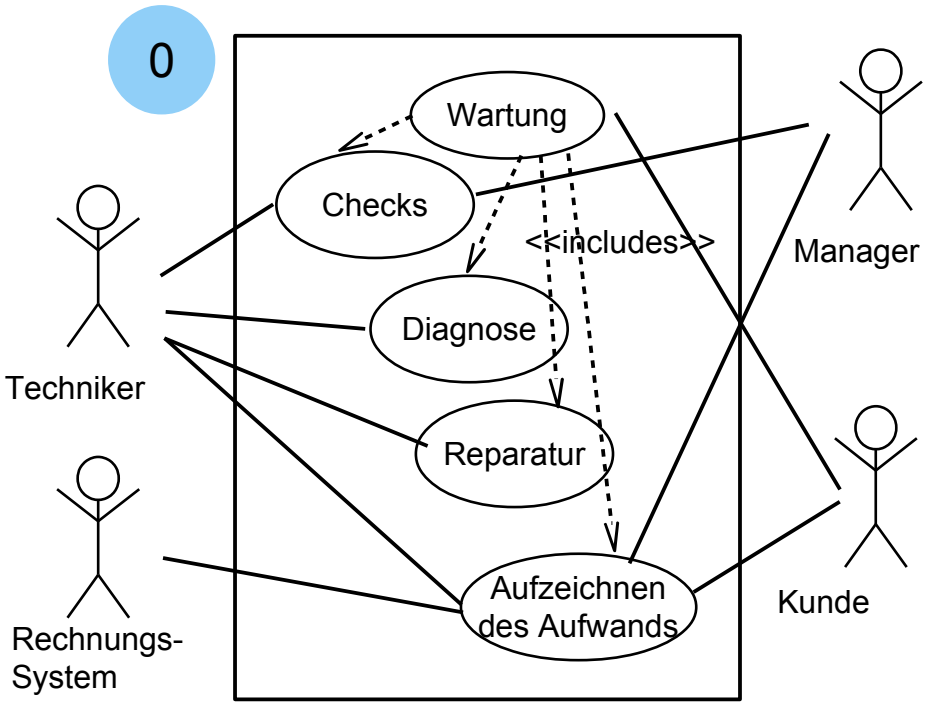
- Ein Sequenzdiagramm einer Kollaboration definiert:
  - Lebenslinien beschreiben das Verhalten der Rollen
  - Die Lebenslinie mit dem Anfangszustand kennzeichnet den **Initiator** mit **Initialzustand**
  - Die Lebenslinie mit dem Endzustand kennzeichnet den **Terminator** mit **Endzustand**;
  - **Initialbotschaft**: erste Botschaft, anliegend am Initialzustand
  - **Terminalbotschaft**: letzte Botschaft, anliegend am Endzustand

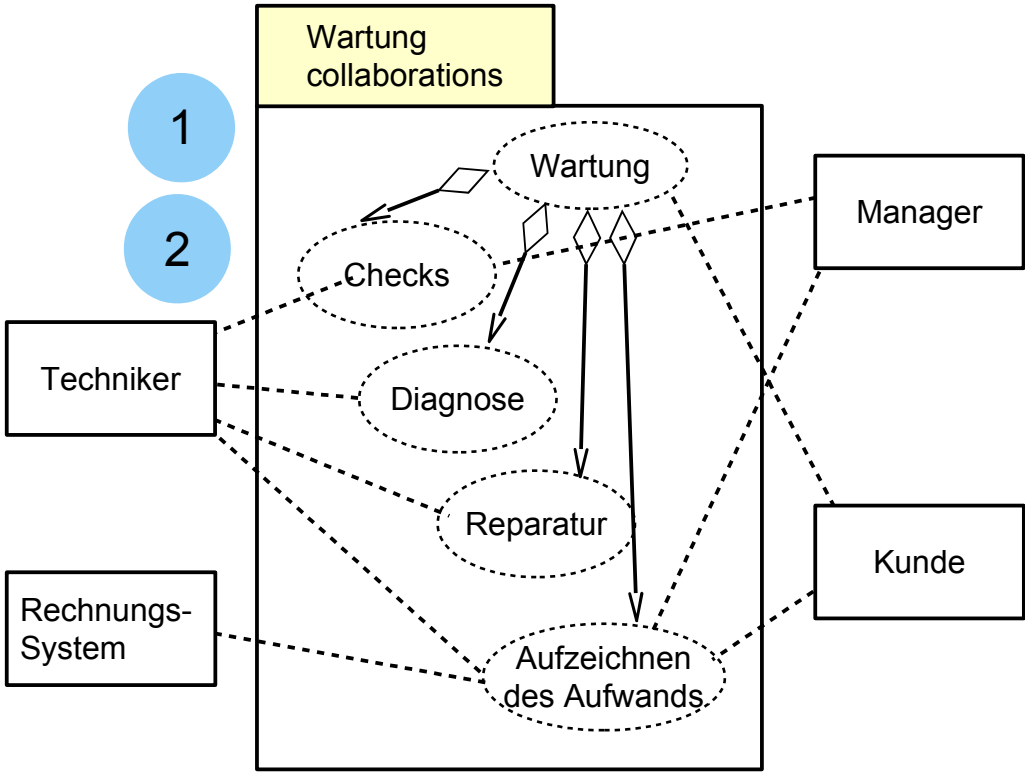


Umwandlung Anwendungsfall-Diagramm in Kollaborationen

- Aus dem Anwendungsfalldiagramm kann schrittweise eine Menge von Kollaborationen erstellt werden

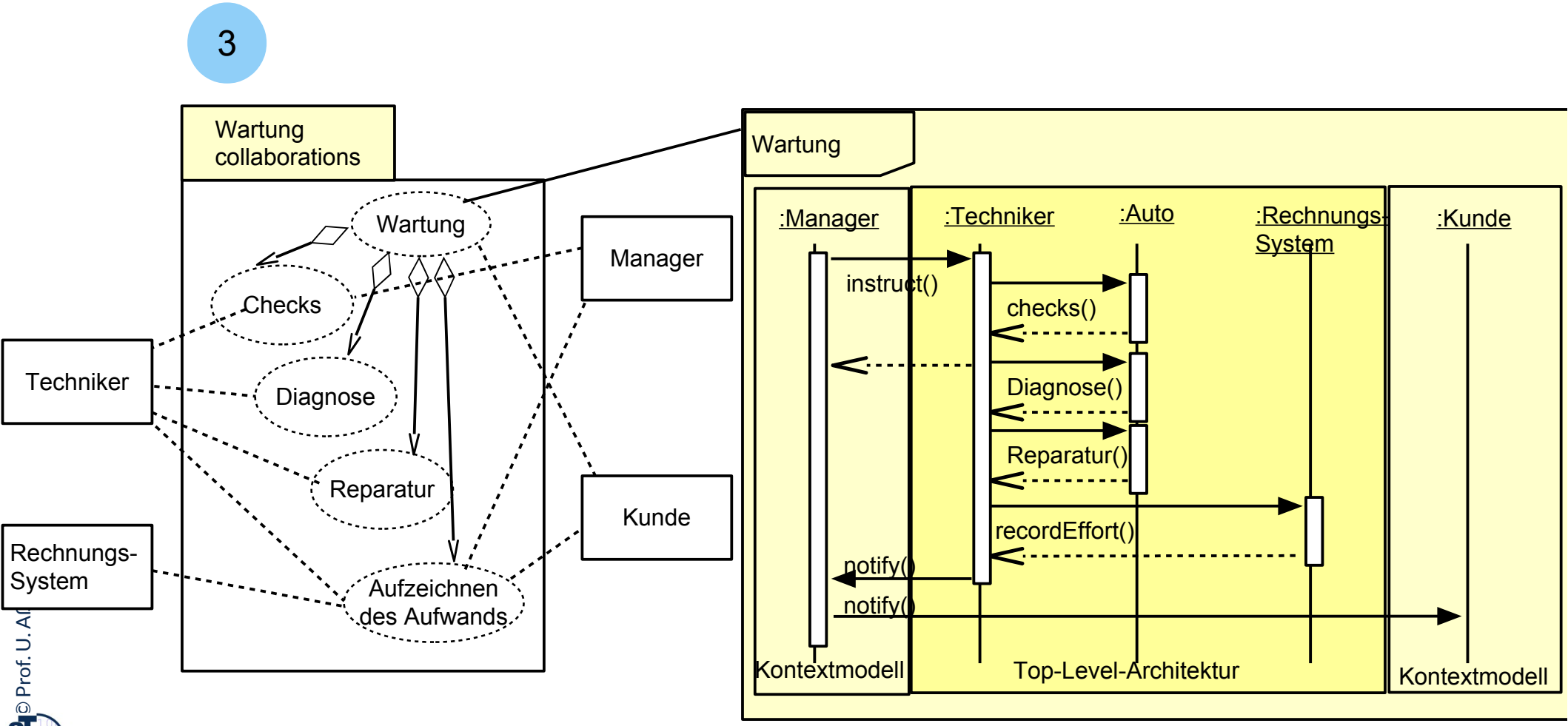






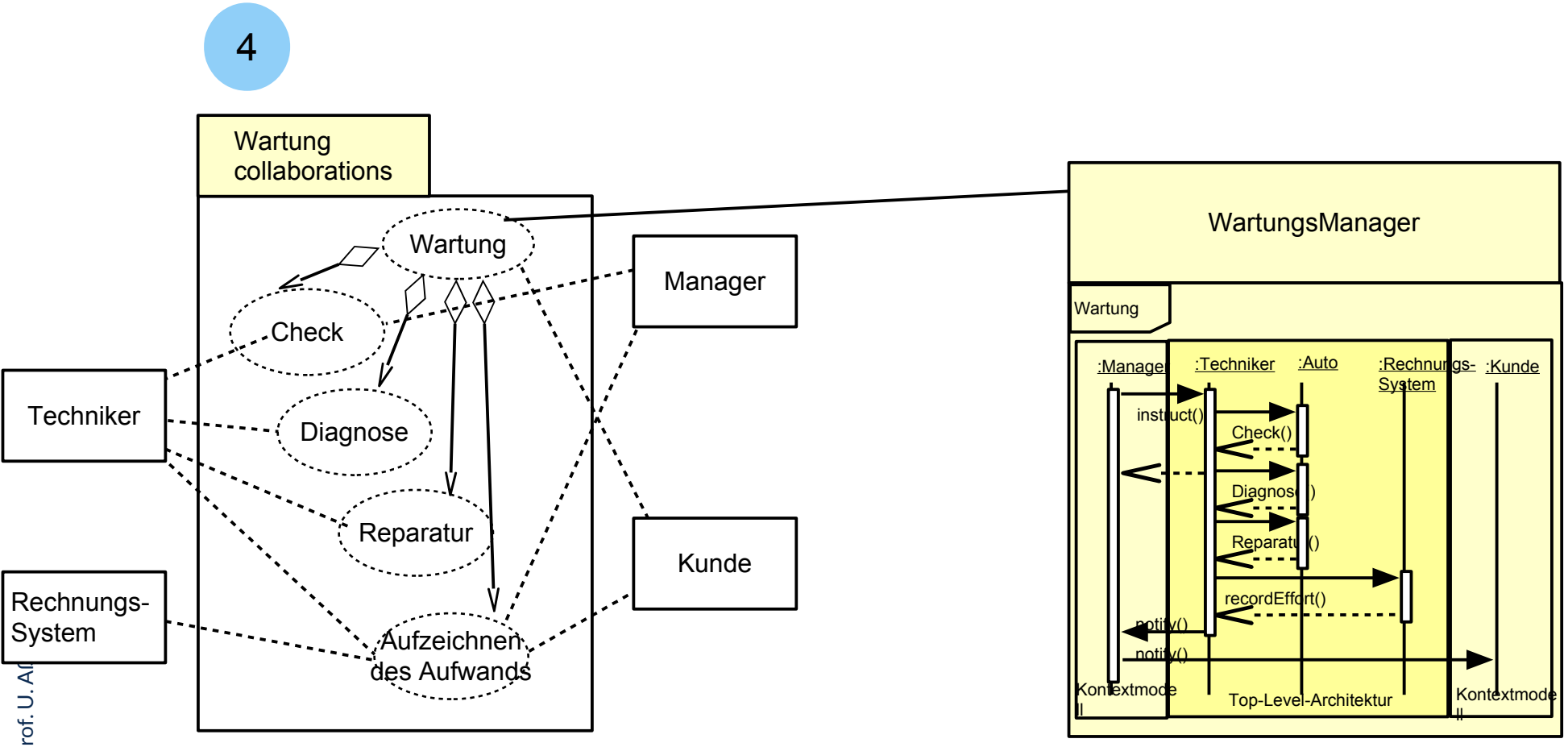
Umwandlung Anwendungsfall-Diagramm in Kollaborationen

- 3) Anhängen des Interaktionsdiagramms (hier Sequenzdiagramm) an einen Anwendungsfall



Umwandlung Anwendungsfall-Diagramm in Konnektor

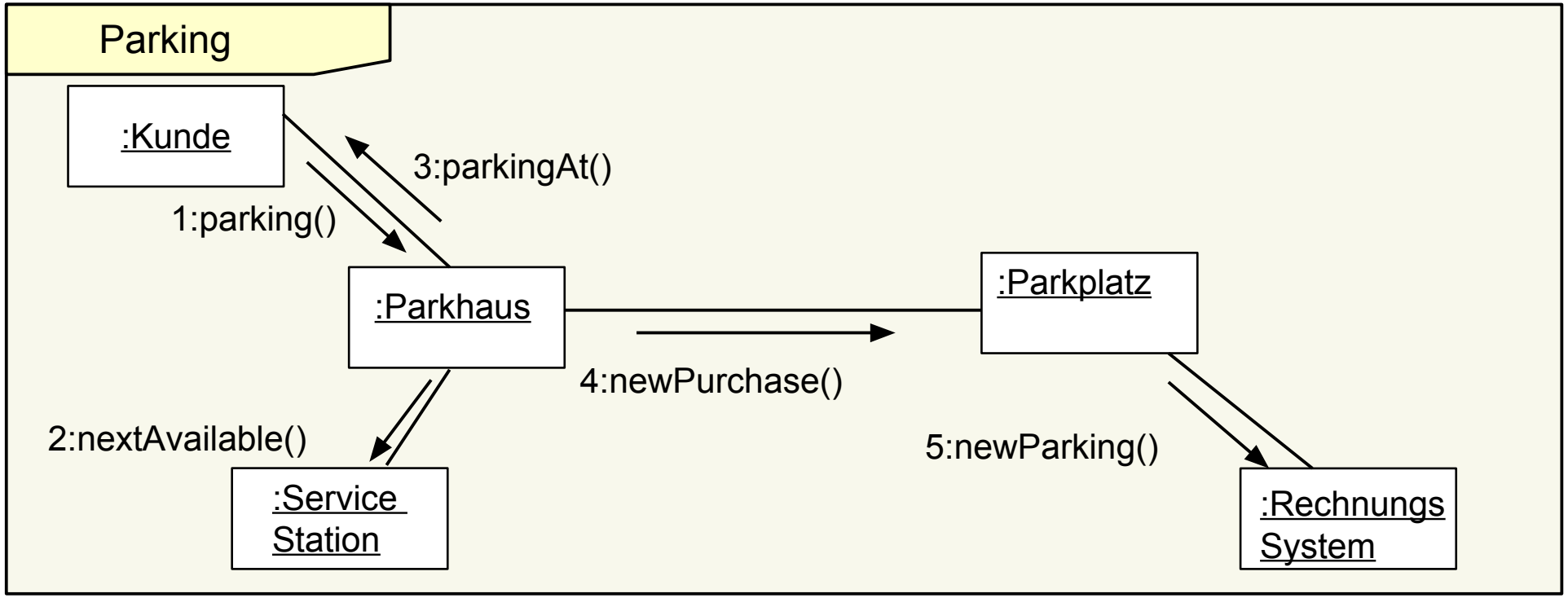
- 4) Umwandlung der Kollaboration in Konnektorklasse, die die querschneidende Kollaboration steuert (“Reifikation”)



35.3 Szenarienanalyse mit Kommunikationsdiagrammen

Kommunikationsdiagramm (Communication Diagram)

- Ein **Kommunikationsdiagramm** ist ein Interaktionsdiagramm, das den Fluss der Aufrufe zwischen Objekten über der Zeit aufzeichnet
  - Sequenzdiagramm „von oben gesehen“
  - Ohne Objektlebenslinien, flexibles Layout
  - Hierarchische Nummerung drückt die Zeit aus (zeitliche Abfolge der Nachrichten und Aufrufe)
  - Geeignet für Objektnetze mit komplexem Verbundverhalten



35.4. Szenarienanalyse mit Schwimmbahnen in Aktionsdiagrammen

Szenarienanalyse funktioniert auch mit

Aktionsdiagrammen: Aktivitätendiagramme (UML-AD), Statecharts (UML-SC)

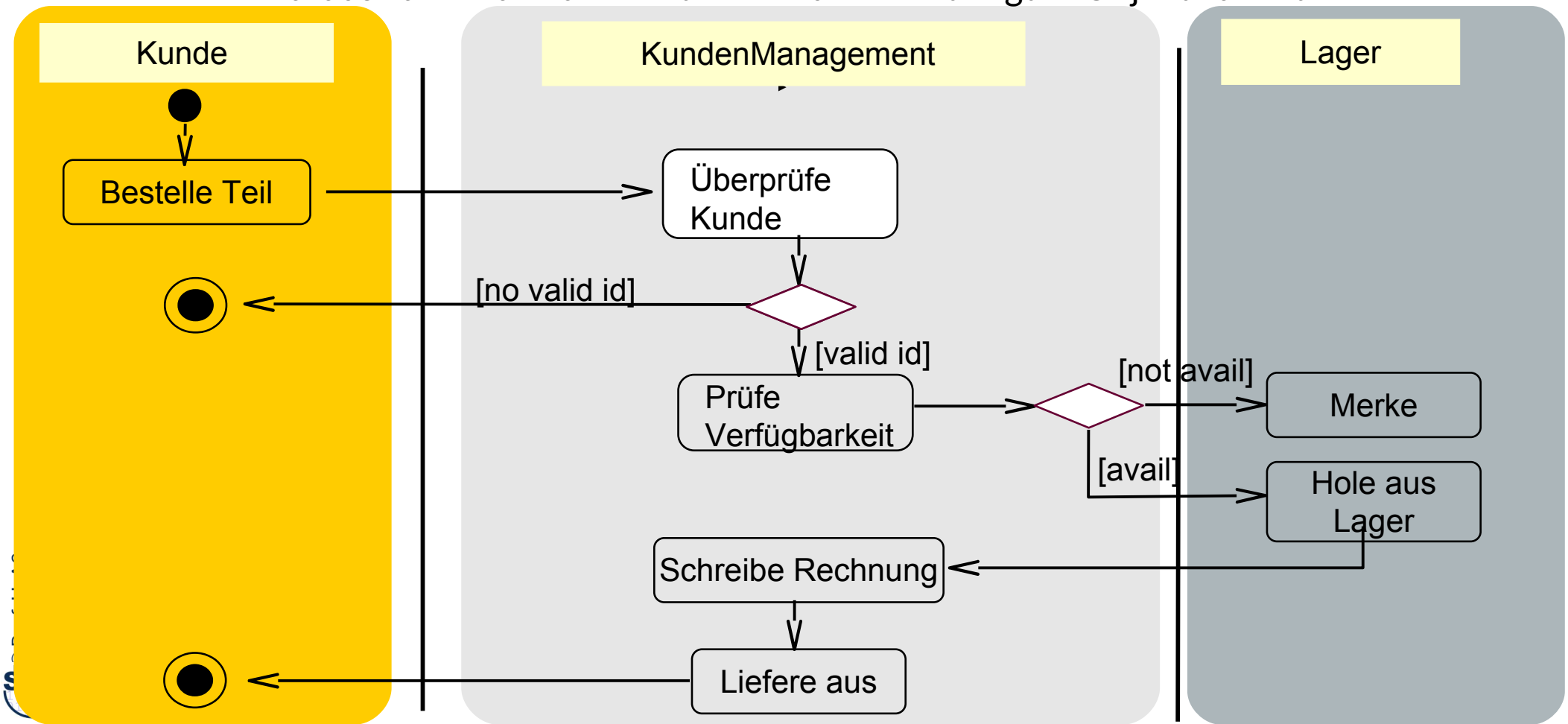
Querscheidende dynamische Modellierung mit Szenarienanalyse

- Mit Aktionsdiagrammen kann man **Lebenszyklen** von Objekten spezifizieren (punktweise Verfeinerung)
- Benutzt man ***Schwimmbahnen***, kann man das Zusammenspiel mehrerer Objekte oder Methoden untersuchen (*querschneidende dynamische Modellierung*, *querschneidende funktionale Verfeinerung*).
  - Dazu führt man eine *Szenarienanalyse* durch, die quasi die Draufsicht auf ein Szenario ermittelt

- *Achtung: in UML wird eine Aktivität genau wie ein Zustand mit einem abgerundeten Rechteck dargestellt..*

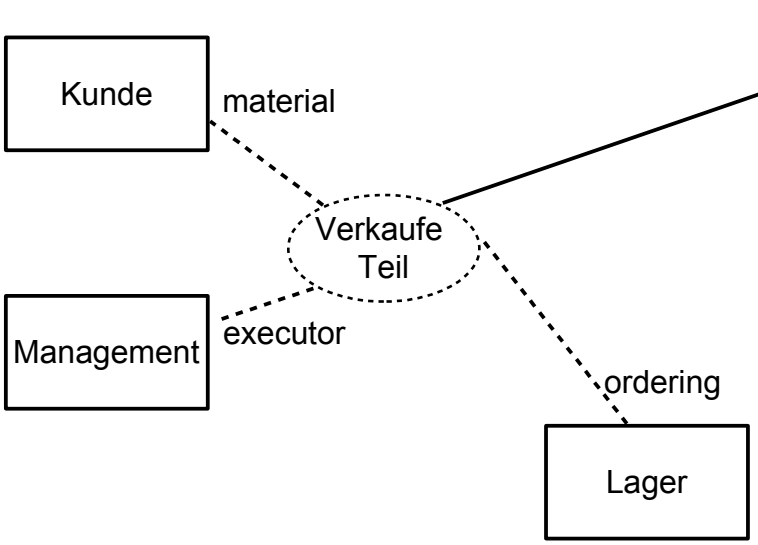
Szenarienanalyse mit UML-AD: Bearbeiten einer telefonischen Bestellung

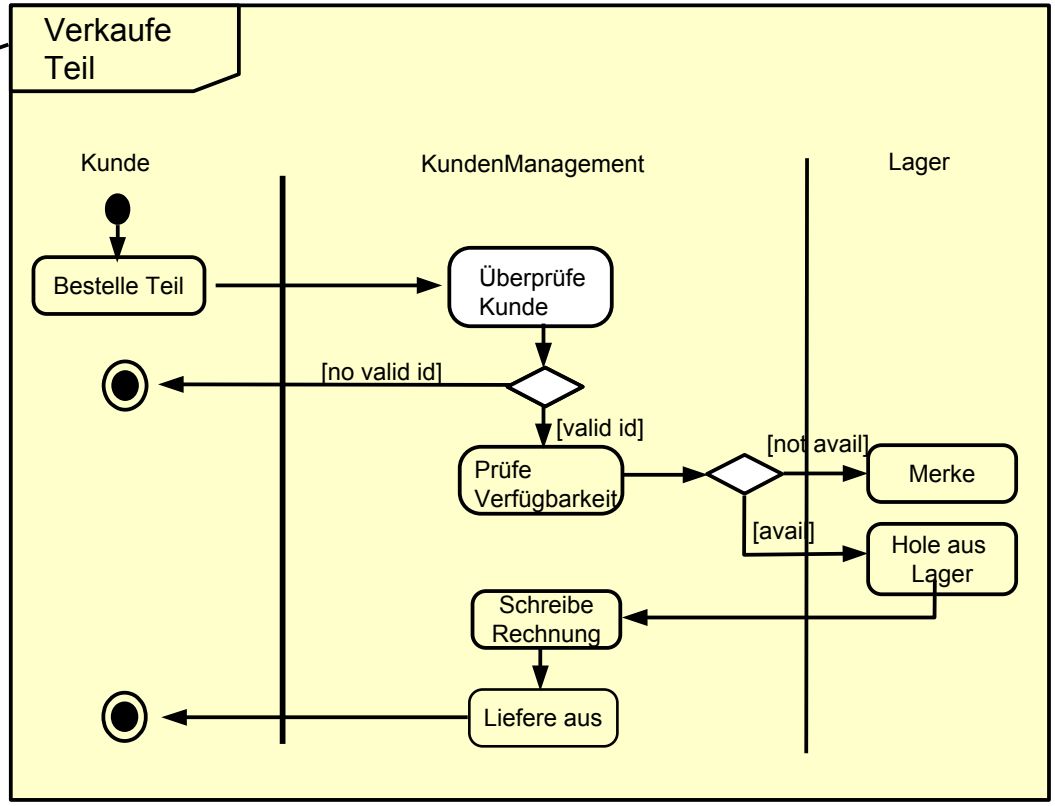
- Aktivitäten können durch **Schwimmbahnen (swimlanes)** gegliedert werden, die Objekten zugeordnet sind
  - Jede Schwimmbahn spezifiziert eine **Rolle** eines Objekts im Kontext des Szenarios
  - Daraus kann man dann Methoden für die beteiligten Objekte ableiten



Aktivitätendiagramme als Verhalten von Kollaborationen

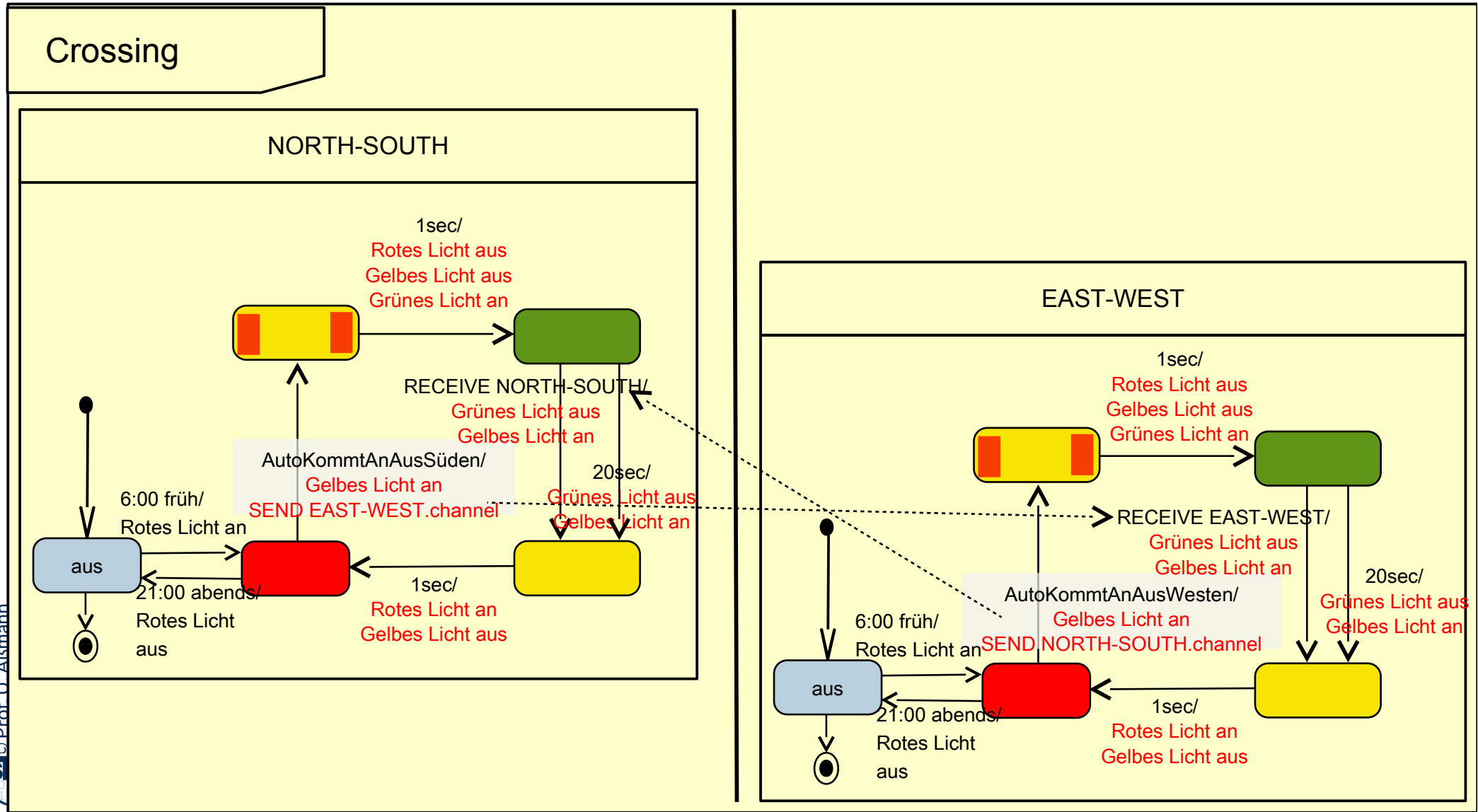
- Aktivitätendiagramme mit Schwimmbahnen können, ähnlich wie Sequenzdiagramme, zu Kollaborationen als Implementierung hinzugefügt werden
  - Die einzelnen Schwimmbahnen geben das Verhalten einer Rolle der Kollaboration an
  - Wieder gibt es Initiator und Terminator-Lebensbereich mit Initial- und Finalzustand





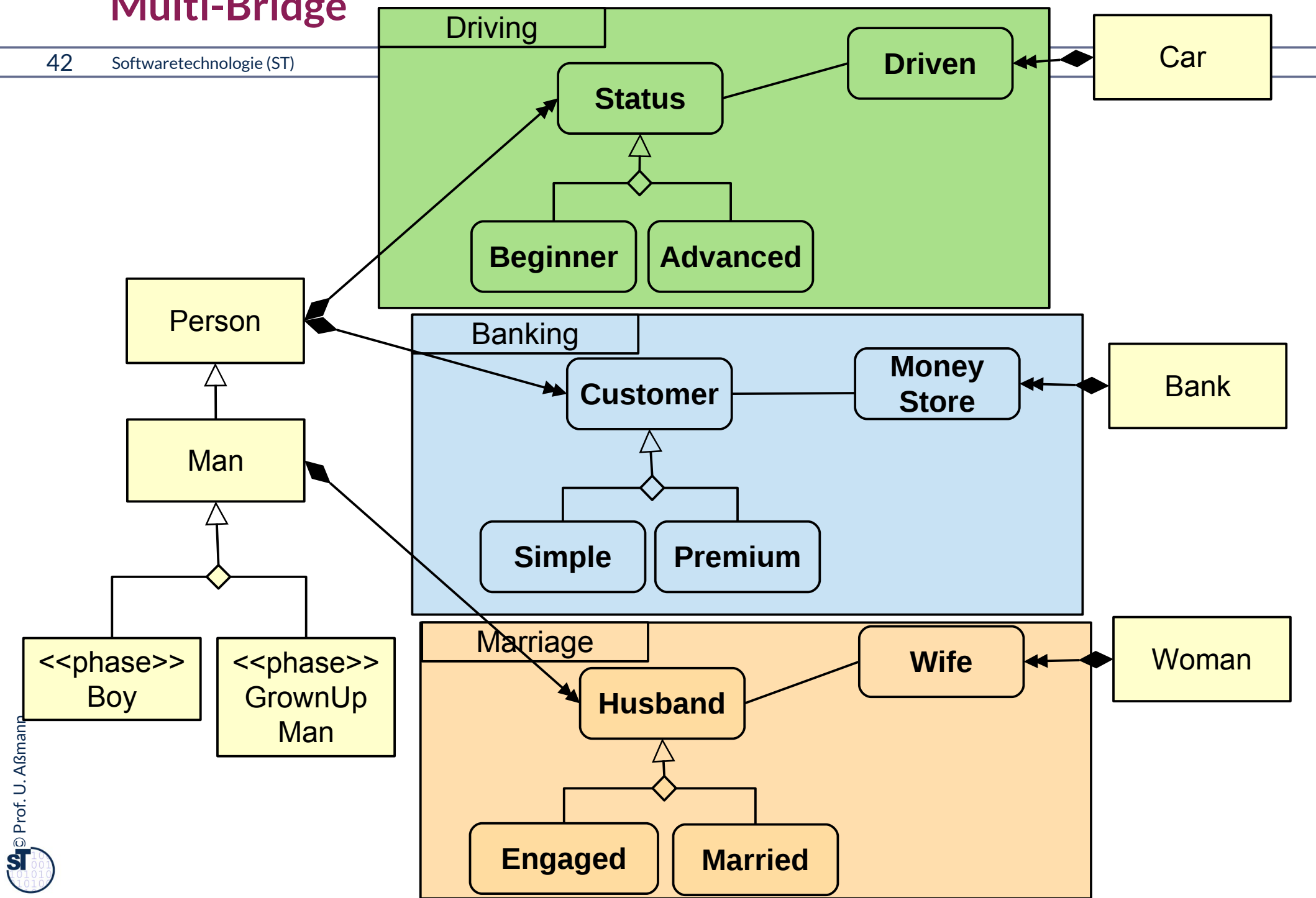
35.4.2. Kopplung zweier Ampeln an einer Kreuzung durch kooperierende Automaten

- Szenarienanalyse mit Statecharts funktioniert ähnlich; es entstehen Netze von kommunizierende Verhaltensmaschinen



35.5 Wozu braucht man querschneidende Verfeinerung mit Szenarienanalyse, Kollaborationen und Konnektoren?

Beipiel: Querschneidende Erweiterung von Objekten mit Multi-Bridge

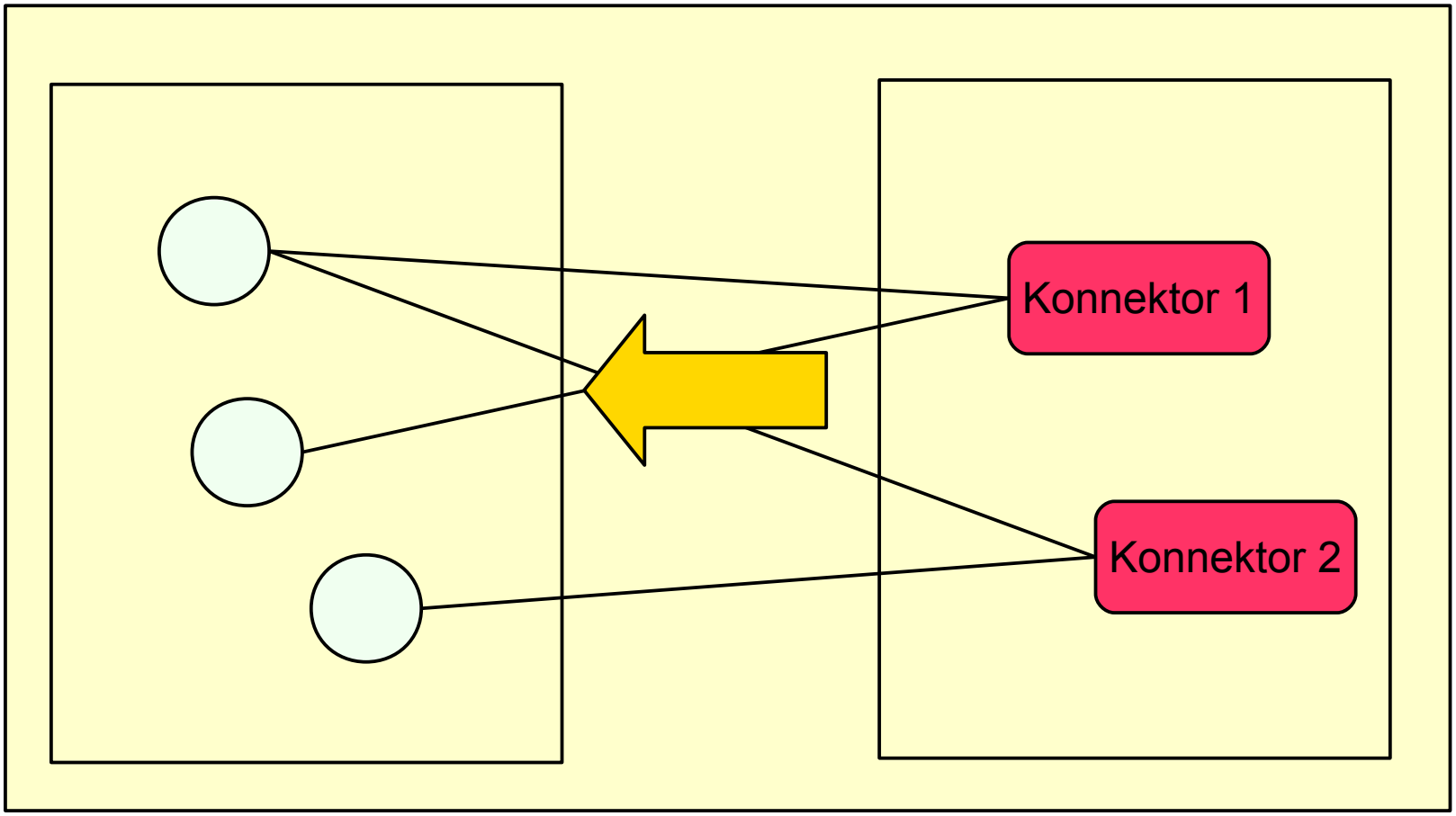


Wozu braucht man querschneidende Verfeinerung mit Szenarienanalyse, Kollaborationen und Konnektoren?

- **Szenarienanalyse** ermittelt *querschneidendes Verhalten durch eine Menge von Klassen bzw. Objekten.*
  - Genau wie Klassen bilden Kollaborationen und Konnektoren *modulare Wiederverwendungeinheiten*, also spezielle Komponenten.
  - Konnektoren besitzen ein *Steuerobjekt*, das eine Kollaboration steuert
- Aber: Kollaborationen und Konnektoren bilden *relationale*, *querscheidende* Module, die die Ergebnisse von Szenarienanalysen kapseln können
- Einsatz:
  - zur Verfeinerung
  - zur Erweiterung
  - zur Ersetzung
  - zur Variabilität
  - zum Umwickeln anderer Kollaborationen (Wrapping)

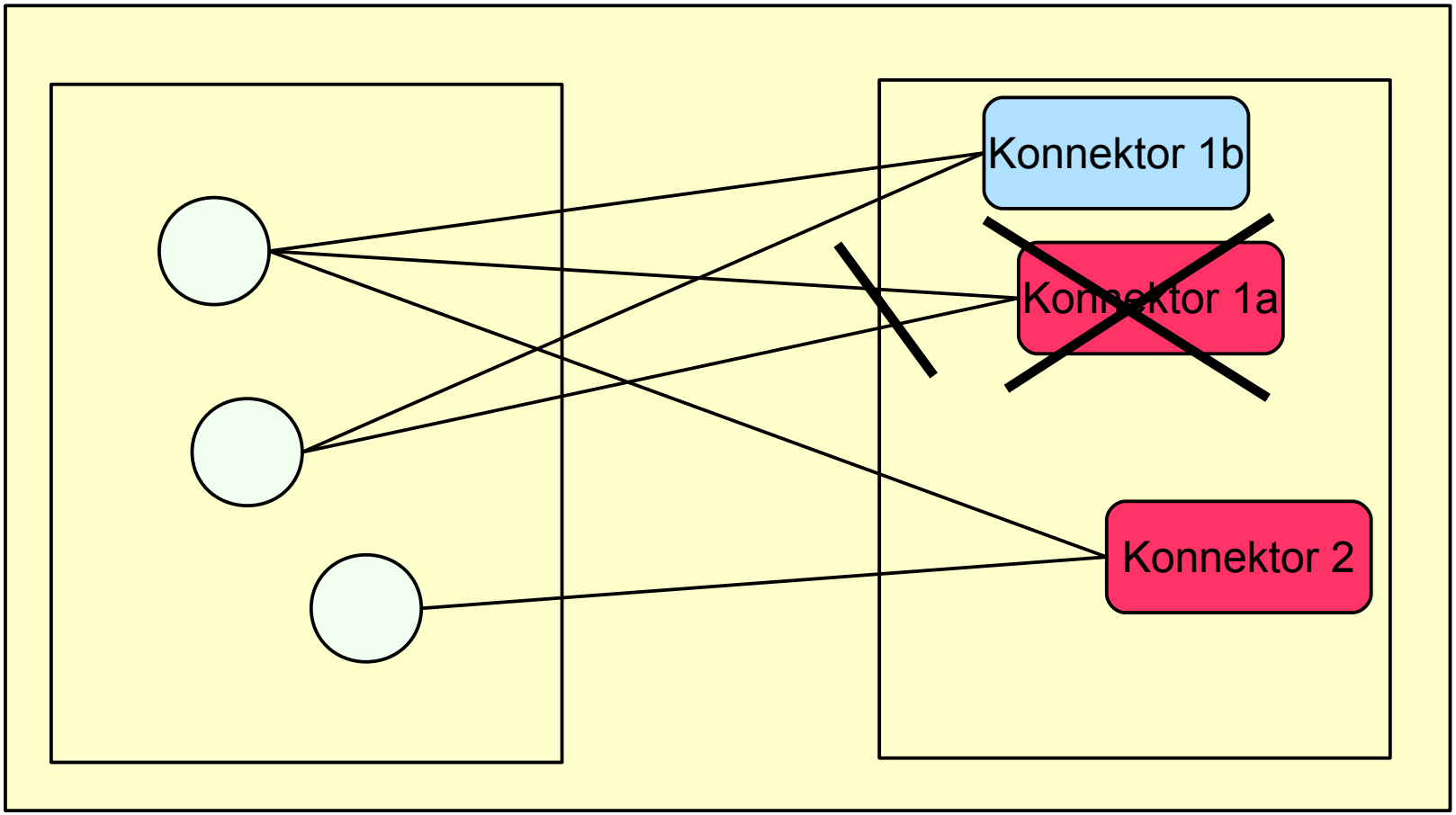
Erweiterung

- **Querscheidende Erweiterung** erweitert *mehrere Punkte* eines bestehenden Systems



Ersetzbarkeit

- Querschneidende Ersetzbarkeit ersetzt *Schnitte* durch ein System



Was haben wir gelernt?

- Ein Anwendungsfall (use case im use case diagram) kann durch Szenarienanalyse verfeinert werden
  - Aus dem Anwendungsfall kann eine Kollaboration abgeleitet werden
  - Sowie ein Interaktionsdiagramm, das das Protokoll zwischen den Rollen der Kollaboration beschreibt (Sequenzdiagramm oder Kommunikationsdiagramm)
  - Oder ein Aktionsdiagramm, das ebenfalls das Protokoll zwischen den Rollen der Kollaboration beschreibt (Aktivitätendiagramm oder Statechart mit Schwimmbahnen)
- Szenarienanalyse verfeinert querschneidend, i.G. zu punktweiser Verfeinerung
- Kollaborationen sind querscheidende Module, die für Erweiterung, Ersetzung, Variabilität und Umhüllung eingesetzt werden können
- Konnektoren sind spezielle Kollaborationen versehen mit einer Hauptklasse
- Man lagert einen Konnektor in eine Anwendung ein, in dem man alle Spieler mit den Rollen der Kollaboration überlagert.

The End

- Warum ergibt die Szenarienanalyse querschneidende Verfeinerungen der Analysemodelle?
- Wieso kann eine Kollaboration mehrere Objekte erweitern?
- Wie kann das Entwurfsmuster Bridge genutzt werden, auf ein Kernobjekt einen Rollensatelliten aufzuprägen (zu superimponieren)?
- Wie superimponiert eine Kollaboration mehrere Spieler durch neue Rollen?
- Warum ergibt sich aus der Superimposition mehrere Kollaborationen für jedes beteiligte Objekt eine Multi-Bridge?
- Beschreiben Sie die Schritte vom Use Case Diagramm über die Szenarioanalyse zur querschneidenden Verfeinerung.
  - Wie viele Ausprägungen des Musters Bridge erhält man?
  - Wie viele Ausprägungen von Multi-Bridge?

35.A.1 Konnektoren als spezielle Kollaborationen

- Im Entwurf werden Kollaborationen zu *Konnektoren*, d.h. speziellen Kommunikationsobjekten, die querschneidend Kommunikation kontrollieren

35.4.3 Konnektoren

- Ein **Konnektorobjekt** ist ein Assoziationsobjekt, das aus bisher nicht kooperierenden Objekten ein Netz aufbaut, ihre Interaktion und Kommunikation leitet und dann wieder auflöst.
- Eine **Konnektorklasse** ist also eine Kollaborationsklasse, die definiert:
  - ein Konnektor-Hauptklasse für das Konnektorobjekt
  - die Spieler als innere Klassen (Rollenklassen) mit dem Multi-Bridge-Muster
  - Netzaufbau-Methoden, die das Konnektorobjekt mit den Spielern verbinden
  - Netzabbau-Methoden
  - Kommunikationsmethoden, die auf die inneren Objekte delegieren
  - Kanäle, die Daten zwischen den Objekten hin- und herschieben
- Die Verhalten des Konnektorobjekts wird durch eine Kollaboration beschrieben
  - Sie kann **konnektorgetrieben** erfolgen, so dass auf ein Ereignis hin alle Objekte angestoßen werden (passive Objekte werden exogen vom Konnektor angesteuert)
  - Die Bearbeitung kann **spielergetrieben** erfolgen, sodass ein oder mehrere Objekte aktiv über den Konnektor kooperieren
- In Java implementiert man eine Kollaboration immer als **Konnektor**

Schematische Realisierung von Konnektoren mit inneren Klassen in Java

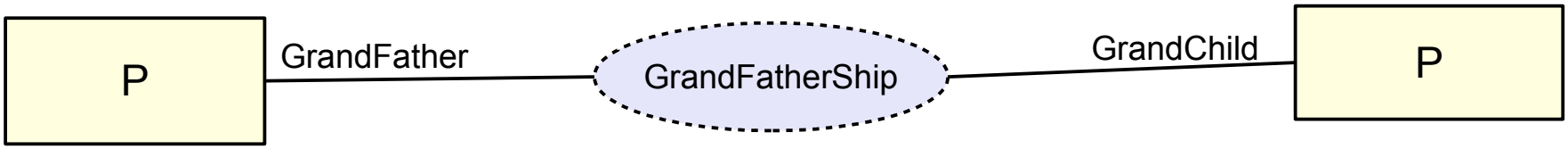

```
class Connector {
  /* role as inner class */
  class RoleA {
     do() {..}
  }
  /* role as inner class */
  class RoleB {
     do() {..}
  }

// Definition of inner objects
  PlayerA player_A;
  PlayerB player_B;
  RoleA role_A;
  RoleB role_B;
}
```

```
// Net construction
void link(PlayerA a, PlayerB b) {
  player_A = a;
  player_B = b;
}
// Net destruction
void unlink() {
  player_A = null;
  player_B = null;
}
// Delegation methods
void doA() { role_A.do(); }
void doB() { role_B.do(); }
```

Bsp.: Konnektoren mit inneren Klassen in Java

- In Java implementiert man eine Kollaboration immer als **Konnektorklasse** mit **Konnektorobjekt** und ggf. **inneren Rollenobjekten**
- **Vorteil:** alle Spieler und Rollen sind gekapselt; Code kann zusammenhängend wiederverwendet werden




```
class Grandfathership {
  /* role as inner class */
  class GrandFather {
    void caressing ();   }
  /* role as inner class */
  class GrandChild {
    void visiting ();   }
  Person player_gf;
  GrandFather role_gf = new
GrandFather();

  Person player_gc;
  GrandChild role_gc = new
GrandChild();
}
```

```
void linkGrandfatherAndGrandChild
 (Person gf, gc) {
  player_gf = gf;
  player_gc = gc;
}
void unlinkGrandfatherAndGrandChild
(Person gf, gc) {
  player_gf = null;
  player_gc = null;
}
// delegation method
void caressing() { role_gf.caressing();
}
void visiting() { role_gc.visiting(); }
```

Konnektoren als Teamklassen in ObjectTeams

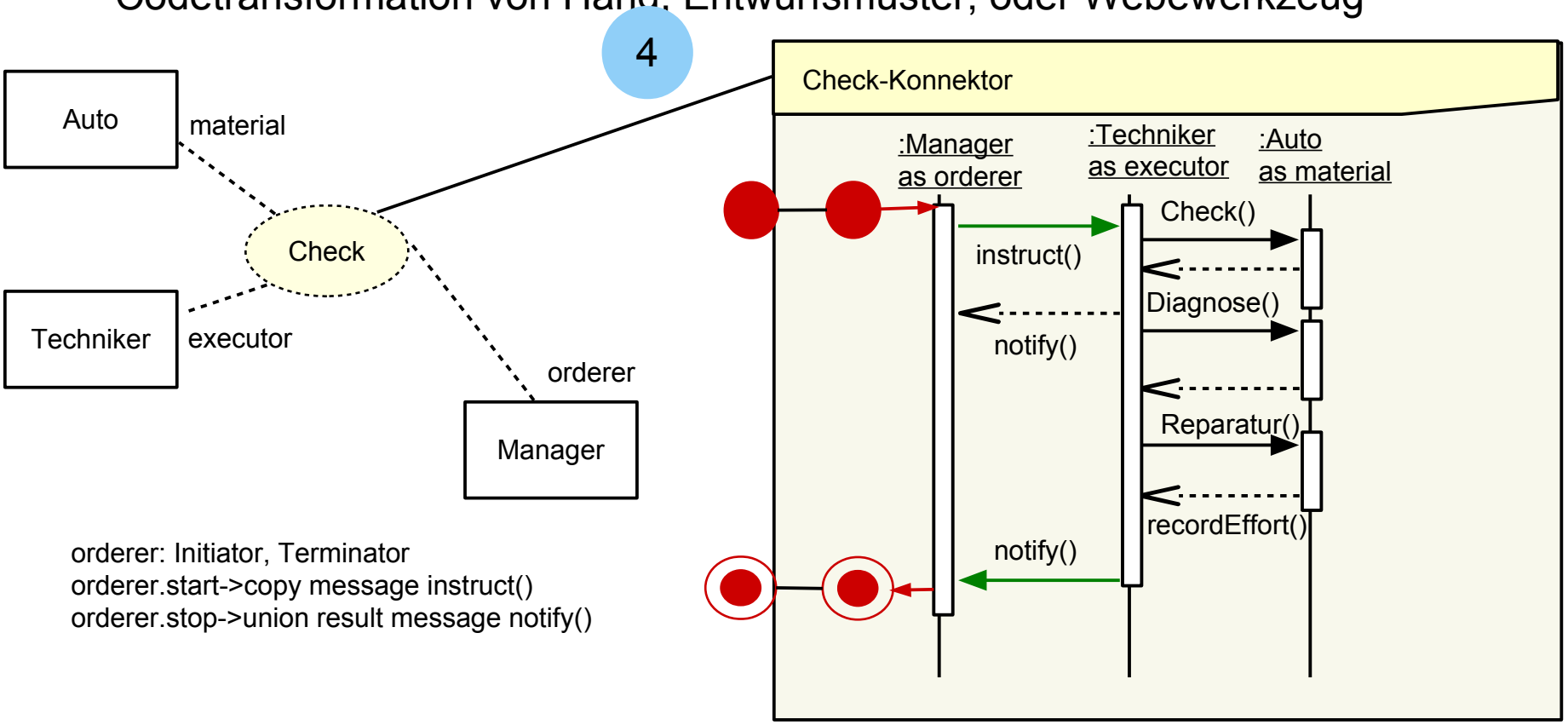
- In fortgeschrittenen Programmiersprachen bilden Kollaborationen und ihre Rollen Sprachkonzepte.
- So auch in der Sprache ObjectTeams der TU Berlin (www.objectteams.org).
  - Hier heißt eine Kollaboration *Team* (Notation als Block, ähnlich zur Klasse)
  - *Rollenklassen bilden innere Klassen* des Teams

```
team Grandfathership {
  /* role class */
  class GrandFather {
    void caressing ();
  }
  /* role class */
  class GrandChild {
    void visiting ();
  }
}
```

```
team NewspaperReading {
  Readable buy();
  /* role class */
  class Reader {
    void breakfast () {
      Readable rd = buy();
      rd.read();
    }
  }
  /* role class */
  class Readable {
    void read();
  }
}
```

Einbau von Kollaborationen in Klassendiagramm

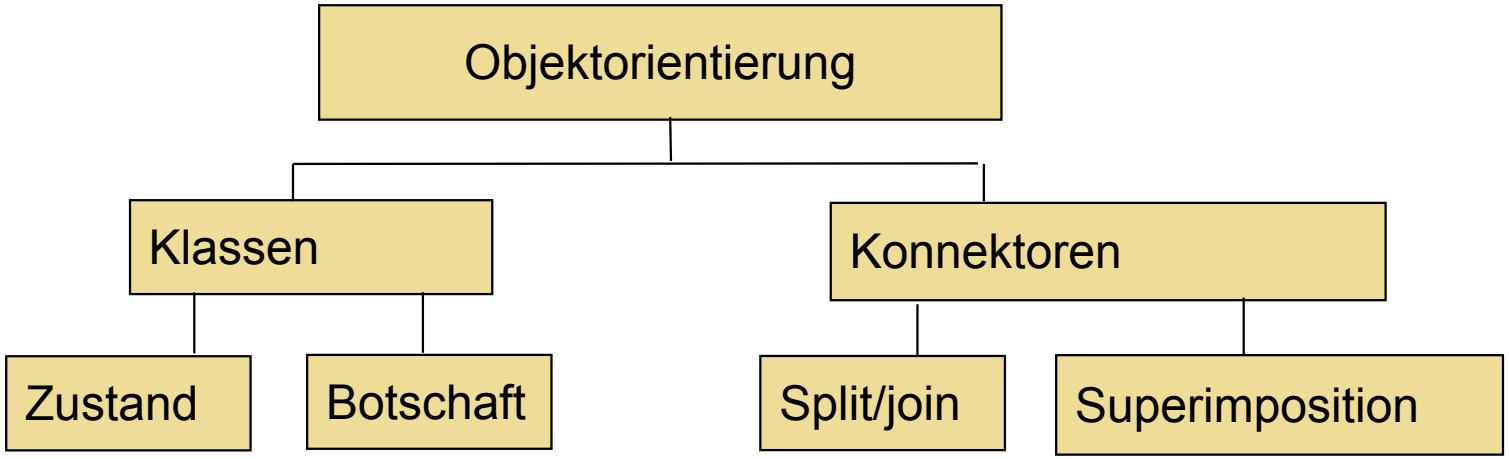
- **Kollaborationsanknüpfung** (Komposition, Superimposition) an Kernklassen in einem Klassendiagramm:
  - Man bettet die Initial- und Terminalzustände bzw. -Botschaften der Lebenslinie einer Initiator- und Terminator-Rolle in die Lebenslinie des Kernobjekts ein
  - Damit werden auch die Initial- und Terminalbotschaften eingebettet
- Der **Einbau** einer Kollaboration auf ein Klassendiagramm erfolgt **statisch** durch Codetransformation von Hand, Entwurfsmuster, oder Webewerkzeug



orderer: Initiator, Terminator
orderer.start->copy message instruct()
orderer.stop->union result message notify()

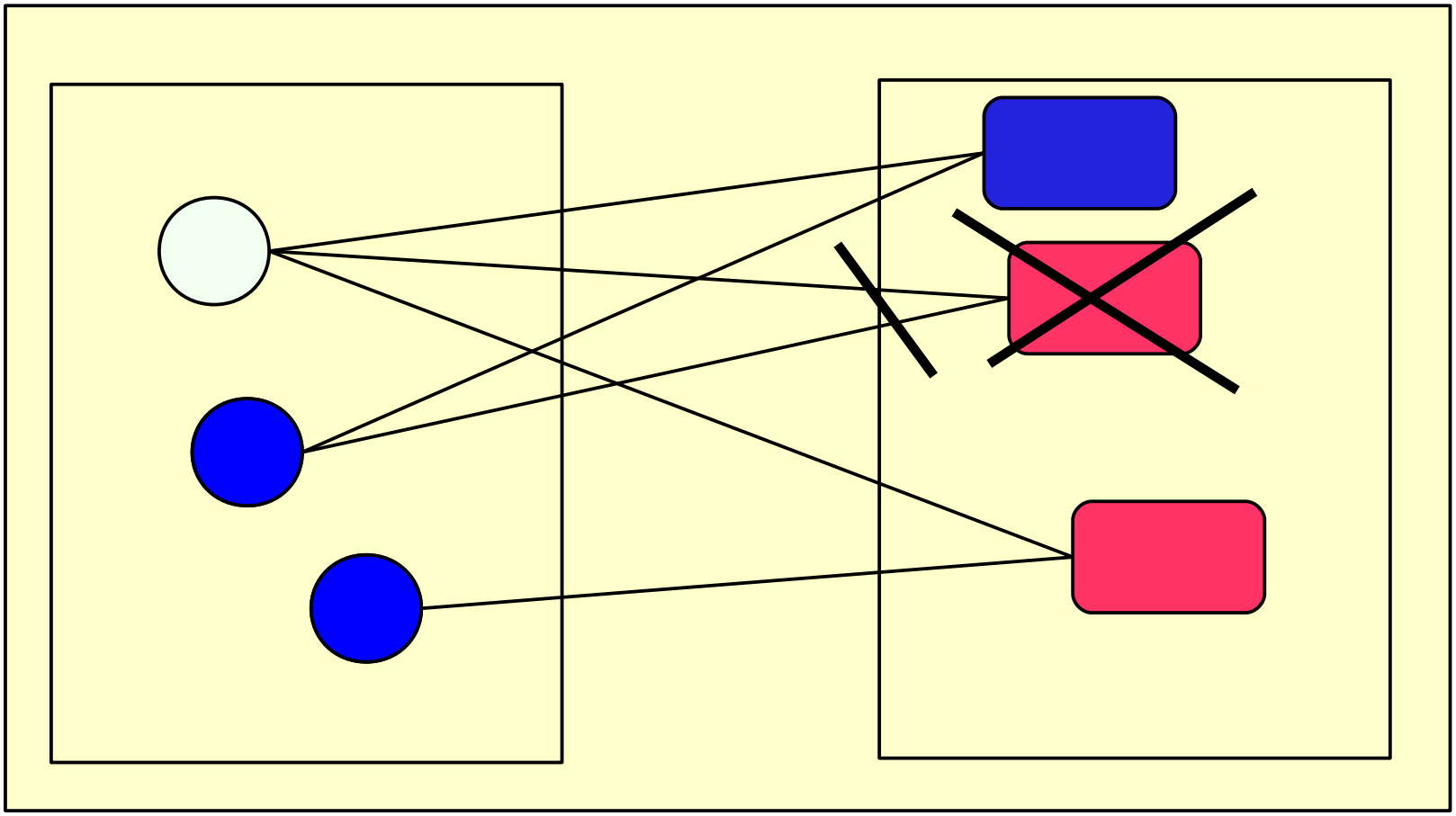
Mögliche Kompositionen von Initial- und Terminal-Botschaften eines Konnektors

- In Konnektoren liegt der Initial- und Endzustand immer auf einer Rolle, NIE auf dem Kern
  - In Objekten liegt das immer im Kern
- Konnektoren werden durch split- und join-Kompositionsoperationen auf existierende Lebenslinien von Klassen superimponiert.
  - Basisoperation von Klassen: Senden einer “Message”
  - Basisoperation von Konnektoren: split und join von Messages in Lebenslinien
- **parallel split and join:** copy control flow of core → start || join control flow with core
- **copy and union:** copy message m of core → start || union result message r into collection of core → stop
- copy data flow d of core → start || copy data flow r into core → stop
- copy/union call c from and into core → stop; start (analog Aspect/J proceed)
- **parallel split and join of control and data:** copy message m and control flow of core → start || union result message r into collection of core and join control flow → stop
- **wrap (call-in, call-out):** wrap message of core



Variabilität

- Management von vielen Varianten von Schnitten, zum Bilden von Produktlinien



Umhüllung (Wrapping)

- Wie umhüllt man bestehende Software durch neue Module, z.B. für Sicherheit?