Umfrage zum CHE-Ranking

- Wichtigstes Hochschulranking in Deutschland
- Fächerbezogene Bewertung
- Kombination aus Fakten und Meinungen
- Jeweils 1/3 der Fächer wird jährlich aktualisiert
- Dieses Jahr: Informatik
- Veröffentlichung im ZEIT-Studienführer (Nächste Ausgabe Mai 2009)
- Durchführung der Erhebung ab August 2008
- Aktuell: Befragung der Studierenden

Wir bitten alle Studierenden, die angeschrieben werden, dringend, an der Befragung teilzunehmen und den Fragebogen verantwortungsbewusst und fair auszufüllen und zurückzusenden.

Methodische und Praktische Grundlagen der Informatik (MPGI 3)

WS 2008/09

Softwaretechnik

Steffen Helke

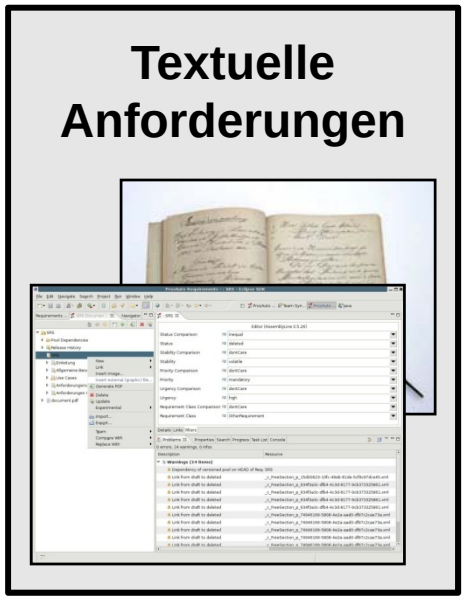
Andreas Mertgen (Organisation)

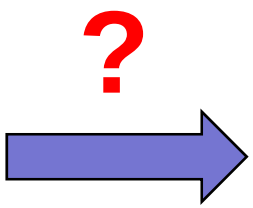
Rojahn Ahmadi, Georgy Dobrev, Daniel Gómez Esperón, Simon Rauterberg, Jennifer Ulrich (Tutoren)

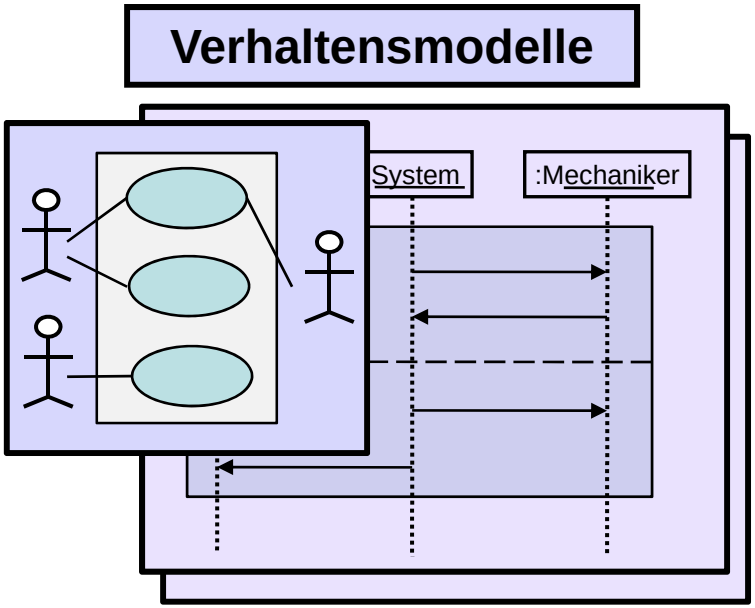
Was machen wir heute?

- **Wiederholung**
  - Use-Case-Modelle
  - Sequenzdiagramme
- **Fortsetzung Analyse**
  - **Aktivitätsdiagramme**
  - **Analyse-Klassendiagramme**

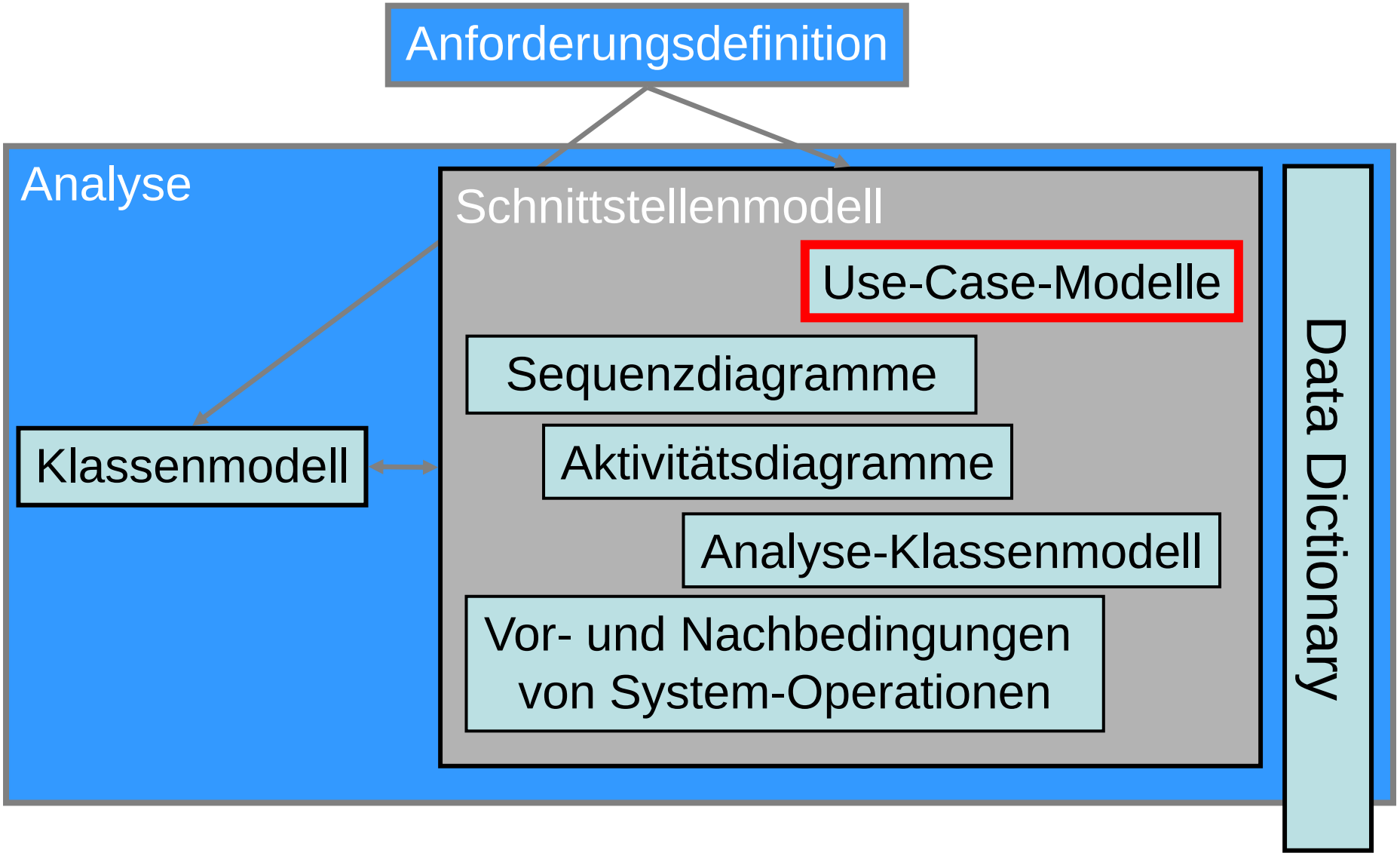
Wiederholung: Use-Case-Modelle und Sequenzdiagramme



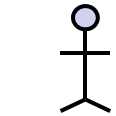
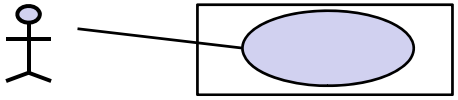
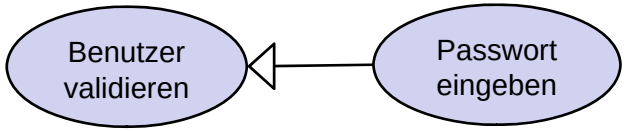
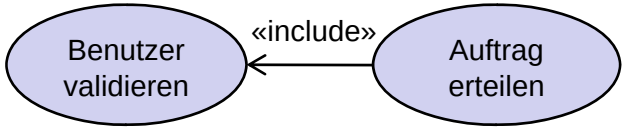
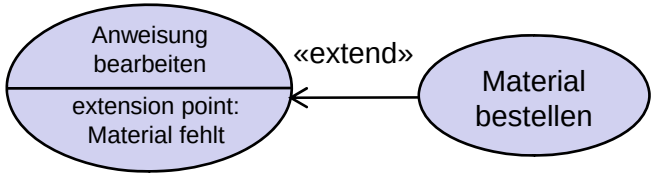




Analyse, 2. Schritt: Use-Case-Modelle



Notationen für Use-Case-Modelle

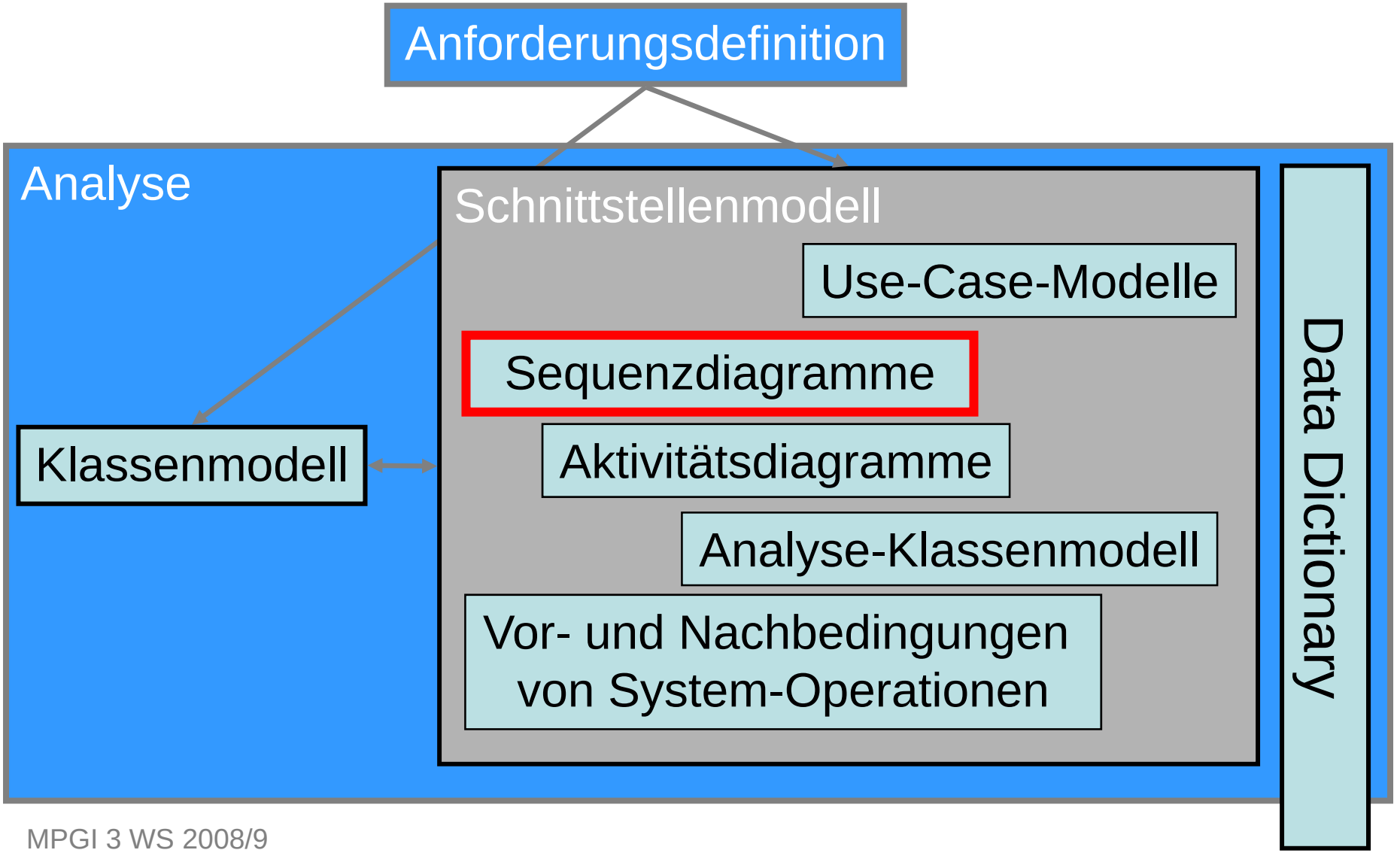
	**Use-Case:** Funktionsgruppe mit komplexem Verhalten, Menge von Szenarien
	**Akteur:** Personen oder Softwaresysteme, die mit dem System interagieren
	**Kommunikation:** zwischen Akteur und Use-Case (Multiplizitäten möglich)
	**Generalisierung/Spezialisierung:** Realisierung von abstrakten Verhalten
	**«include» - Beziehung:** importierender Use Case kann **nicht** allein vorkommen, sinnvoll zur Ausfaktorisierung
	**«extend» - Beziehung:** Basis-Use-Case **kann allein** vorkommen, optionales Verhalten, Einbindung über extension points

Konventionen (Achtung: Korrektur)

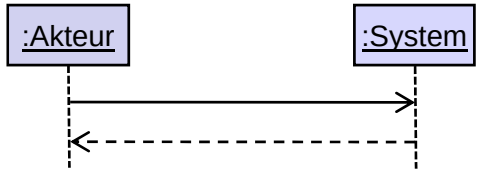
Namen für Use-Cases

- Bezeichner, die eine Tätigkeit beschreiben
- Beispiele:
  - als Substantiv (z.B. „Materialbeschaffung“)
  - oder als Verb (z.B. „Material beschaffen“)

Analyse, 3. Schritt: Sequenzdiagramme

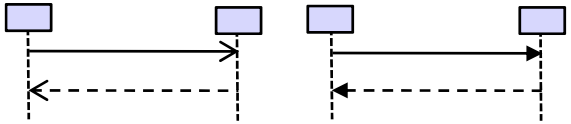


Notationen für Sequenzdiagramme



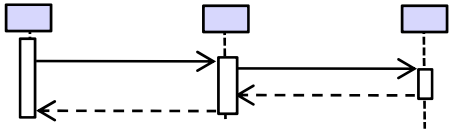
Grundaufbau:

Objekte, Lebenslinien und Nachrichten,
z.B. Systemoperationen und Ausgabeereignisse



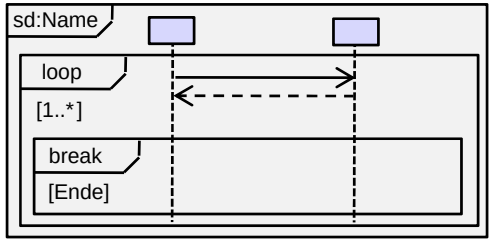
Kommunikation:

synchroner und asynchroner Nachrichtenaustausch



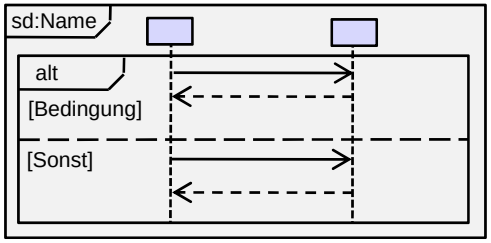
Aktivitätszonen:

synchrone verschachtelte Kontrolflüsse (für Entwurf)



Schleifen:

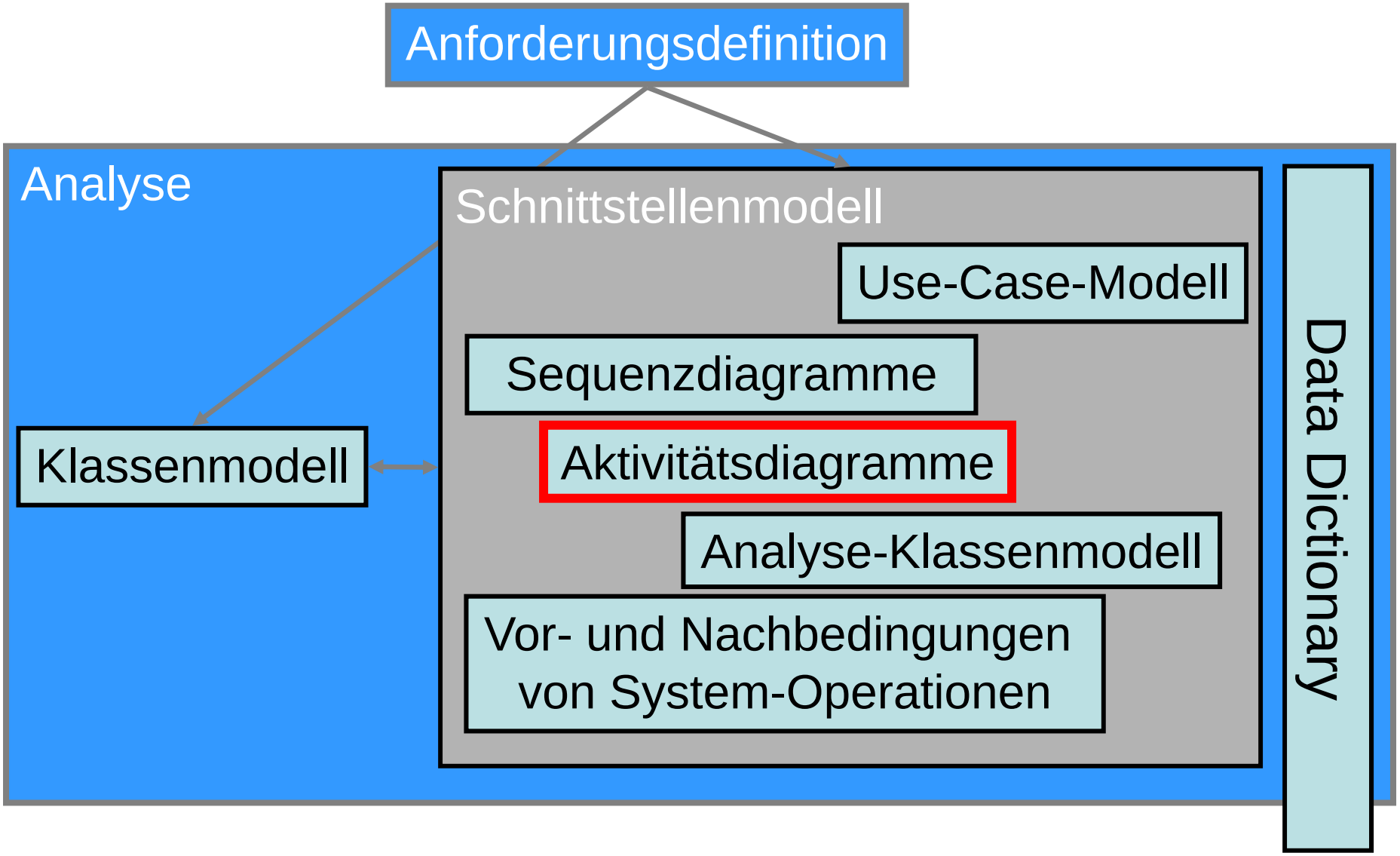
sich wiederholende Szenarien,
mit oder ohne Abbruchkriterium möglich



Alternativen:

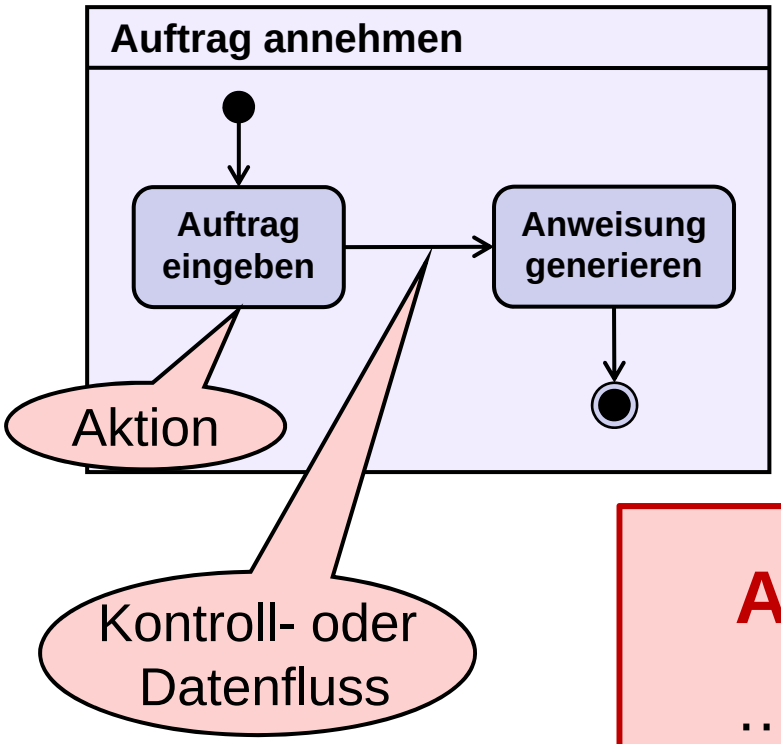
über Bedingungen mit else-Zweig spezifizierbar,
weitere Konstrukte wie z.B. **opt** (optional),
par (nebenläufig), **seq** (lose Reihenfolge) möglich

Analyse, 4. Schritt: Aktivitätsdiagramme



Was sind Aktivitätsdiagramme?

Grundaufbau

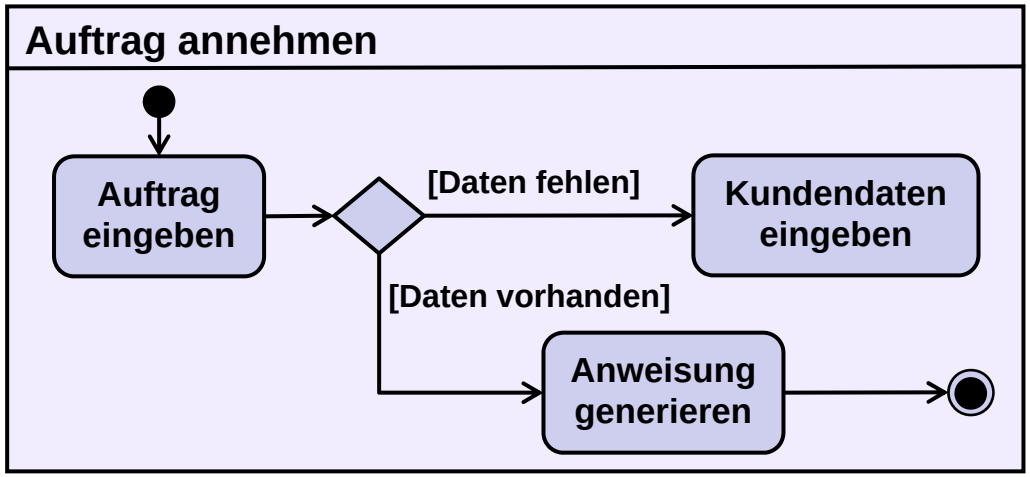


- **Herkunft**
  - Flussdiagramme
  - Programmablaufpläne
  - Petrinetze
- **Aktivitäten**
  - Folgen von Aktionen
  - Verlassen eines Zustands erst nach Beendigung der Aktion

Aktionen sind ...

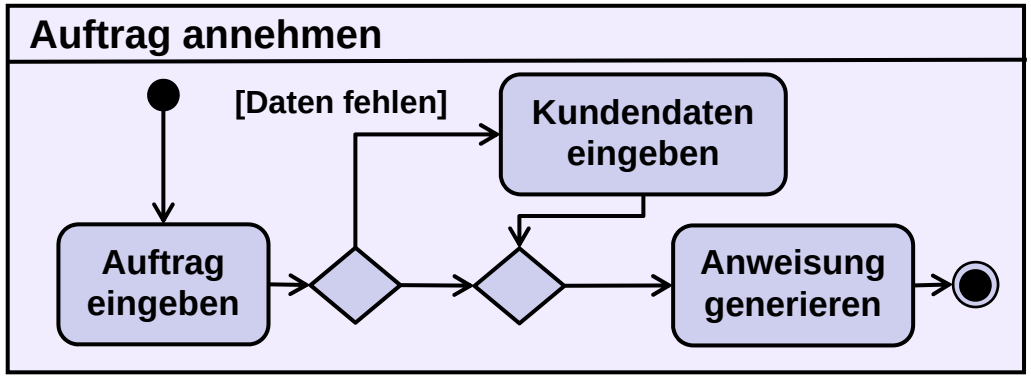
… Ausführungsschritte in einem Algorithmus, Geschäftsprozess oder Systemablauf

Entscheidungsknoten (Decision)



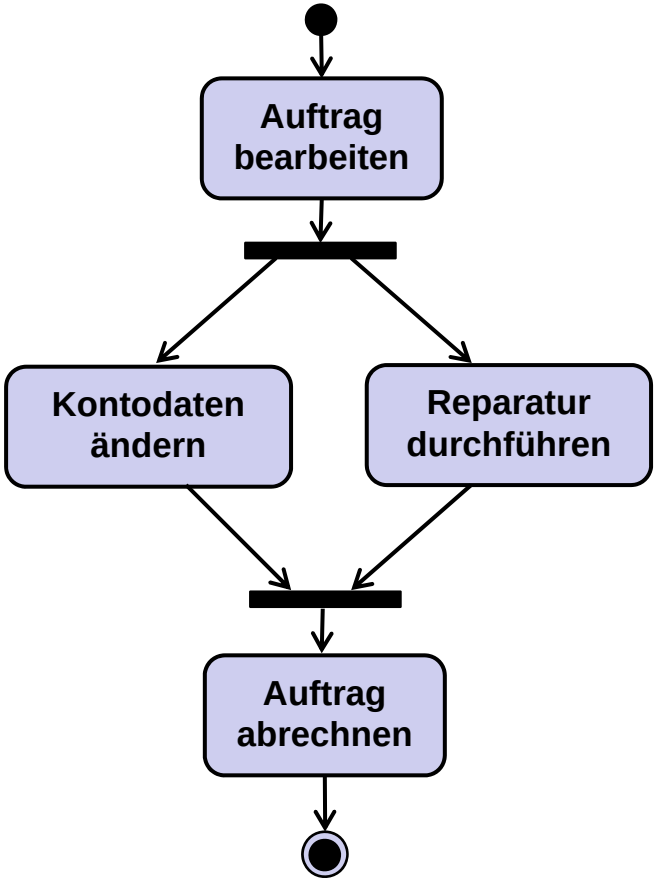
- nur ein Zweig kann weiterverfolgt werden
- **disjunkte** Bedingungen verwenden
- ***else***-Konstrukte möglich (default unbeschriftet)

Zusammenführung (Merge)



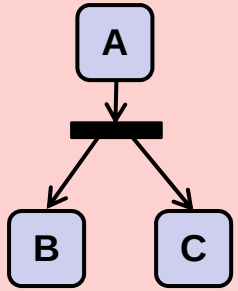
- Zusammenführung mehrerer **alternativer** Daten bzw. Kontrollflüsse

Nebenläufige Aktionen



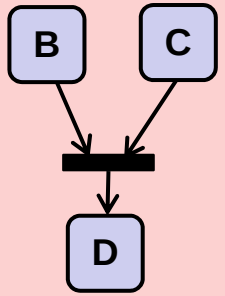
Aufsplittung (Fork)

- B und C beginnen nach A
- B und C sind unabhängig

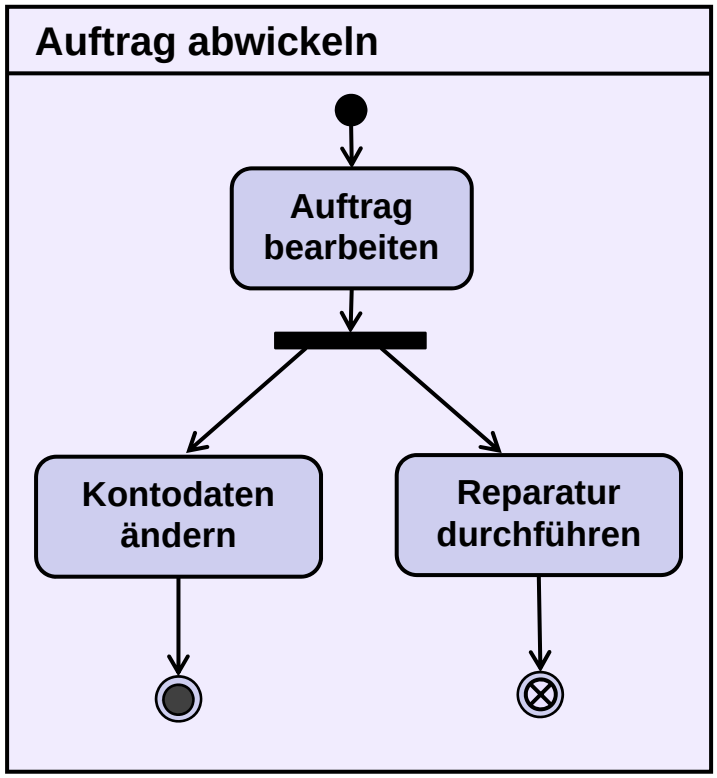


Sychronisierung (Join)

- D beginnt erst nach B und C
- unabhängige Kontollflüsse werden synchronisiert



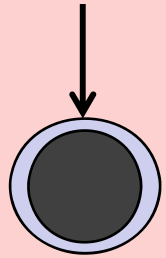
Terminierungsknoten



Aktivität beenden

– gesamte Aktivität wird beendet

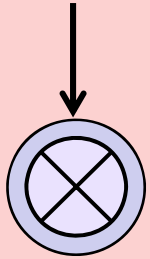
– ***activity final***



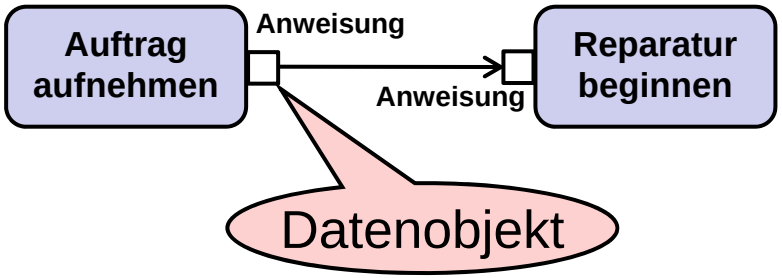
Kontrollfluss beenden

– nebenläufiger Kontrollfluss wird beendet

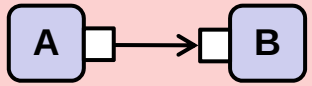
– ***flow final***



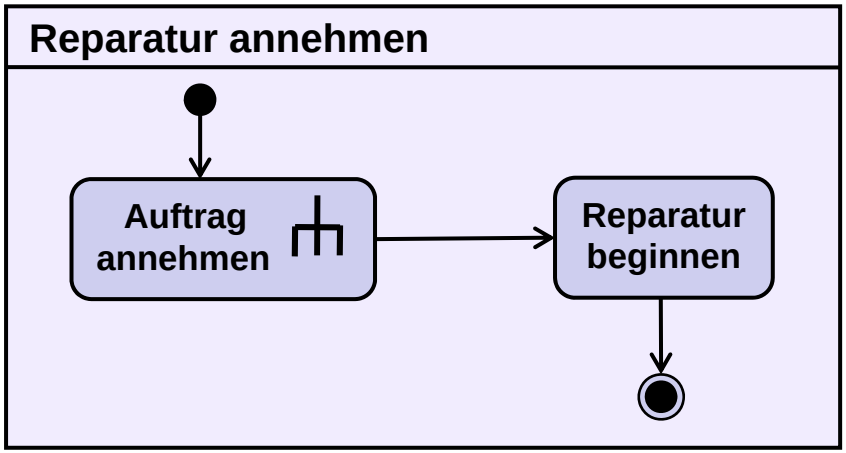
Objektflussgraphen

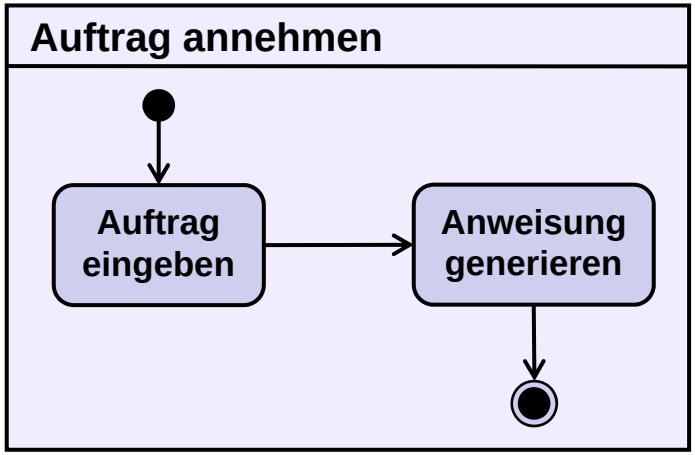


- Aktion A erzeugt Datenobjekt
- Aktion B konsumiert Datenobjekt

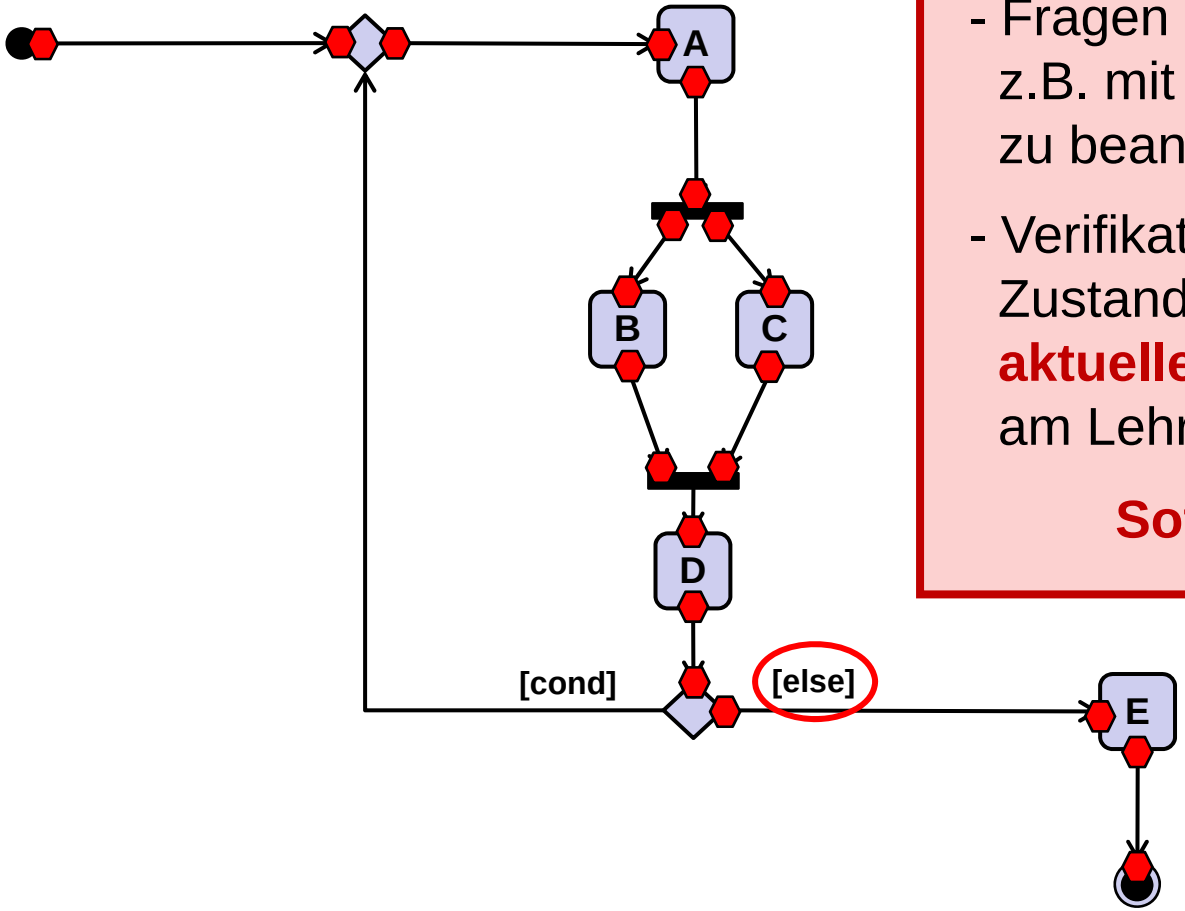


Geschachtelte Aktivitäten





Definition der Semantik mit Hilfe von Token



- Fragen nach Erreichbarkeit z.B. mit Model-Checkern zu beantworten
- Verifikation Hierarchischer Zustandssysteme ist **aktuelles Forschungsthema** am Lehrstuhl

Softwaretechnik

Modellierung mit Verantwortungsbereichen

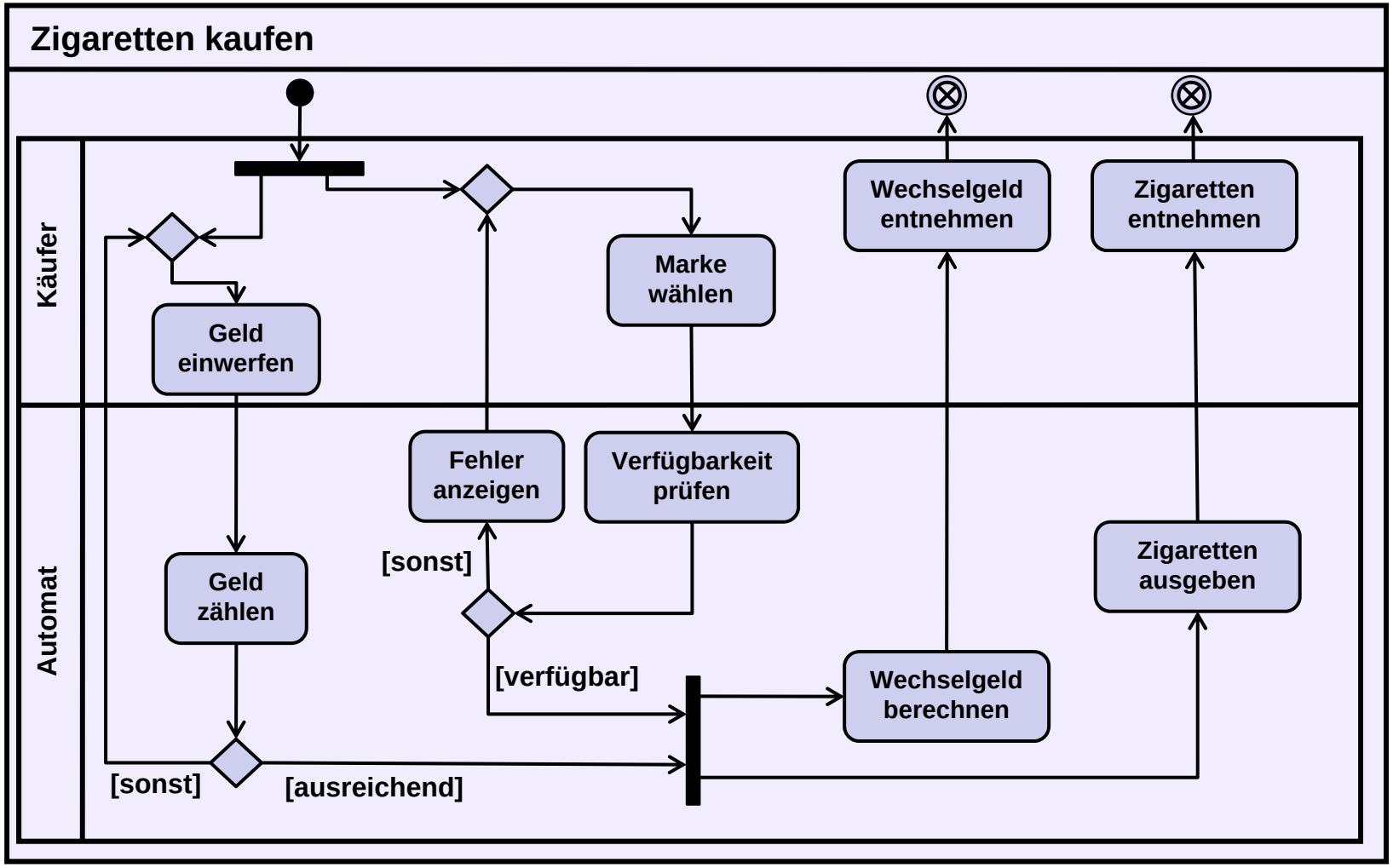
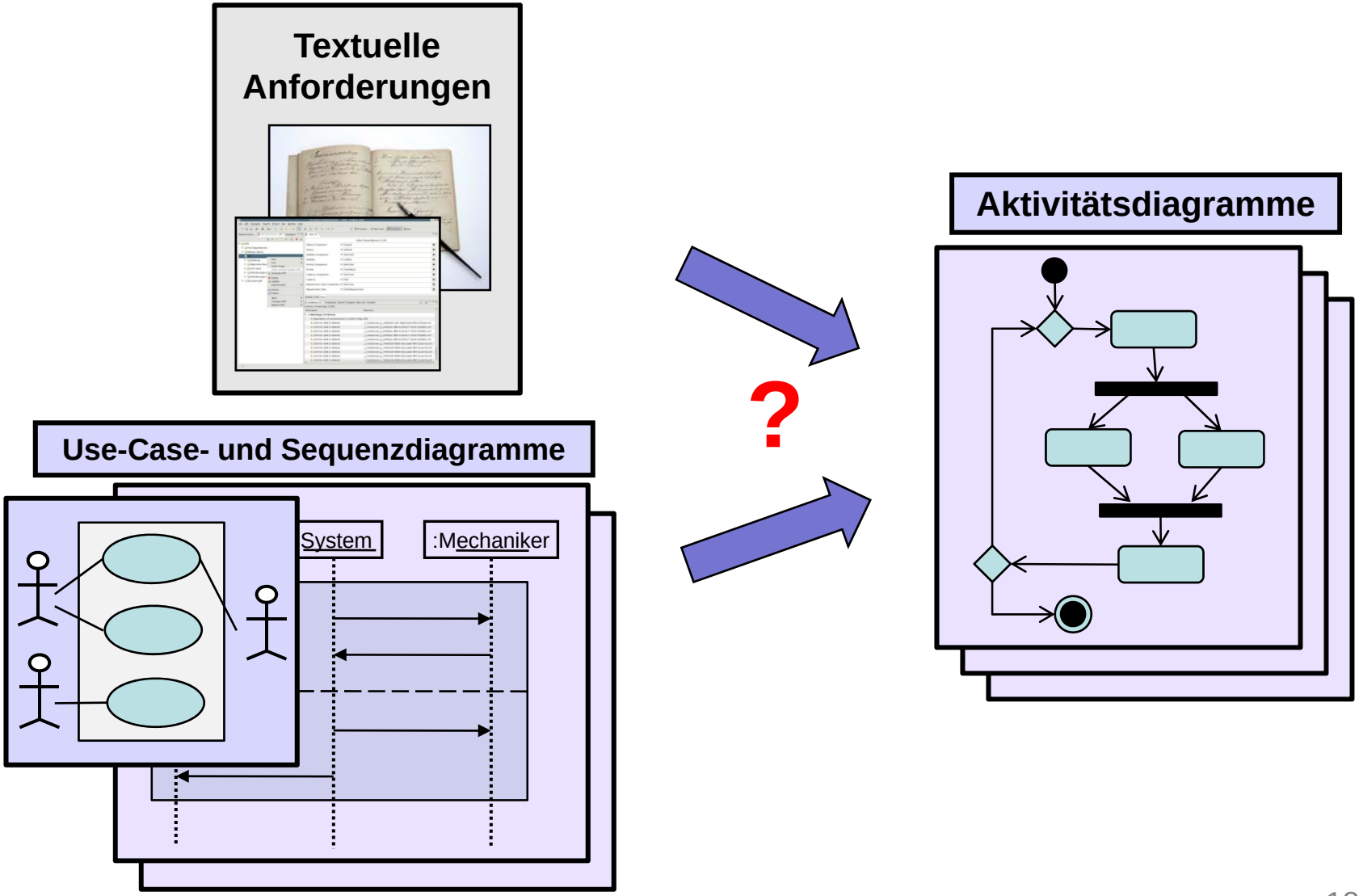


Abbildung nach M. Jeckle: www.jeckle.de, 2004.

Wie kommt man zu Aktivitätsdiagrammen?



Verhaltensmodellierung mit Aktivitätsdiagrammen

Intention

- Beschreibung, die mehrere Sequenzdiagramme in einen Zusammenhang bringt
- ausgehend vom Initialzustand des System **alle** Aktionen des Systems beschreiben

Praktischer Nutzen

Grundlage zur Implementierung von GUI-Skizzen, um beim Kunden die erhobenen Anforderungen zu validieren

Konsistenzverpflichtung

Sequenzdiagramme $\subseteq$ Aktivitätsdiagramme

- Aktivitätsdiagramme enthalten alle Szenarien aus den Sequenzdiagrammen
- Aktivitätsdiagramme können noch zusätzliche Abläufe beinhalten

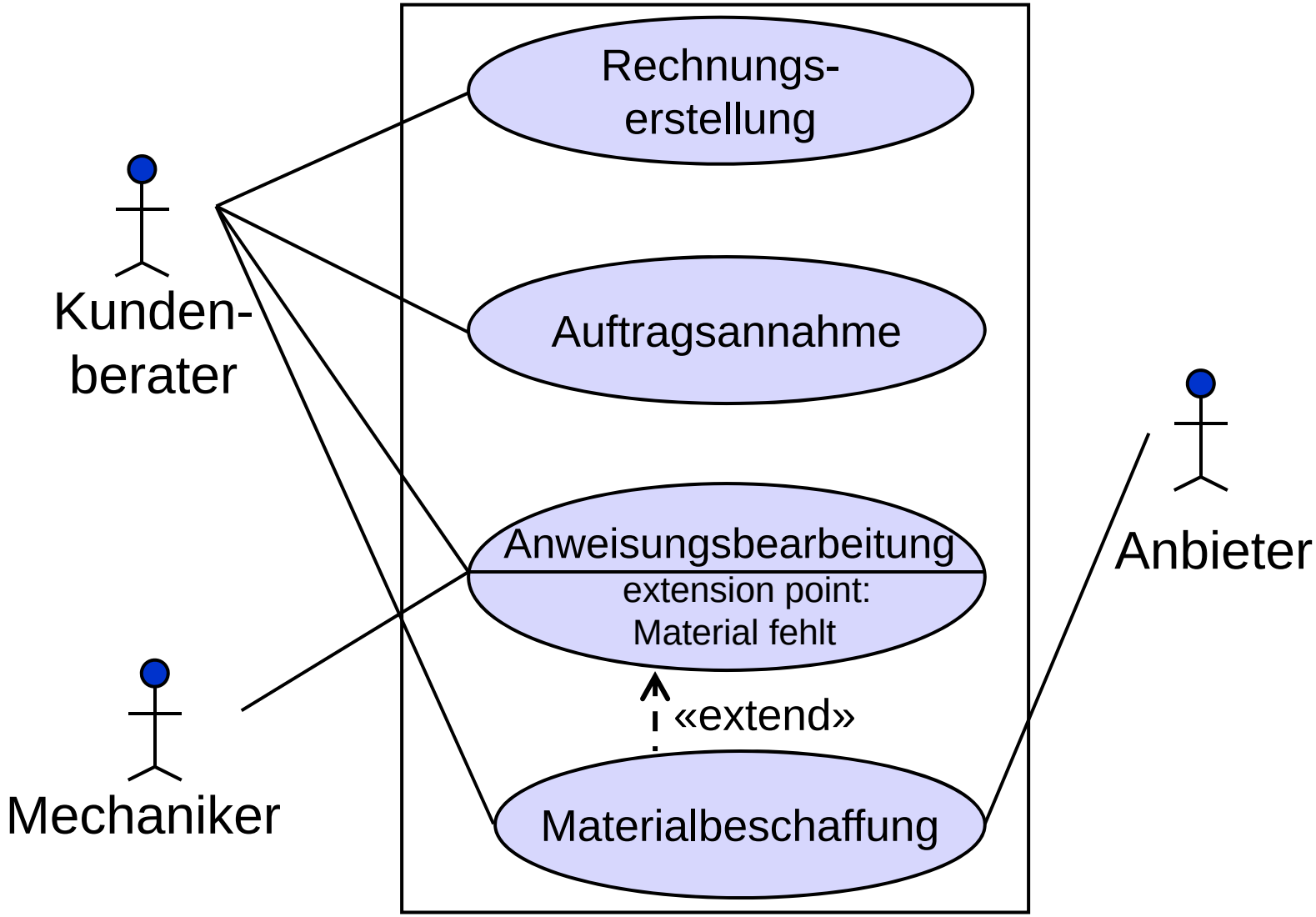
Hinweis

Sequenzdiagramme beschreiben in unserer Methode nur einige wichtige Abläufe des Gesamtverhaltens, aber nicht alle Abläufe.

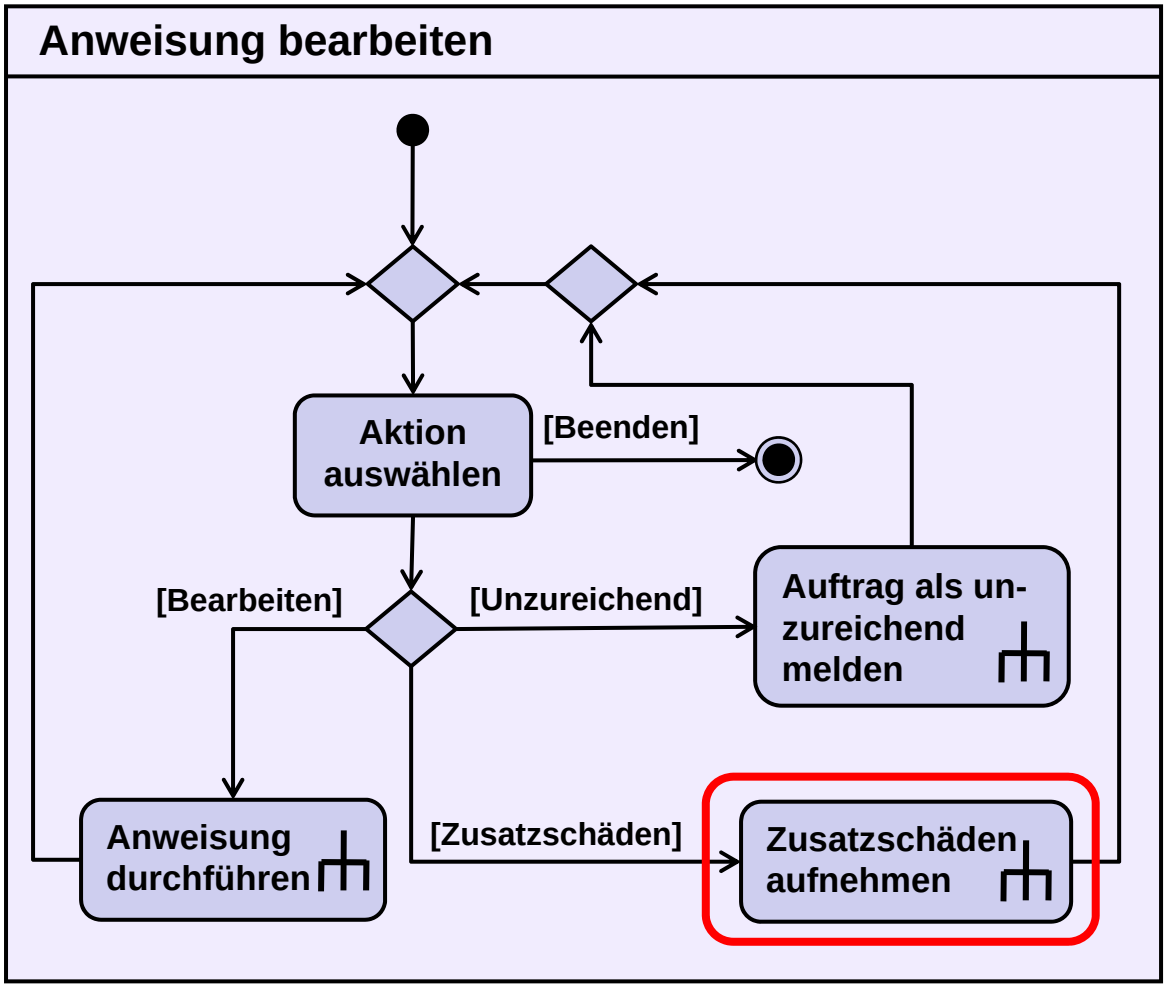
Vorgehen: Erstellung Aktivitätsdiagramme

1. **Schaffe Aktivitätsdiagramm für jeden Use-Case!**
   - Daumenregel: abhängig von der Komplexität des Use-Case
2. **Füge explizite Aktion für GUI-Auswahl ein!**
   - Grundlage für zu implementierende GUI-Skizzen
3. **Repräsentiere Systemoperationen als Aktionen!**
   - Achtung nur Systemsicht, **keine Ausdifferenzierung in Verantwortungsbereiche für Akteur/System**
   - ähnliche Namen verwenden
4. **Beachte Abhängigkeiten zwischen Systemoperationen!**
   - Vorbedingungen von Systemoperationen beachten!
5. **Arbeite Schleifen, Alternativen und Schachtelungen ein!**
   - Guards, wie z.B. Abbruchkriterium bei Schleifen aus Sequenzdiagrammen übernehmen

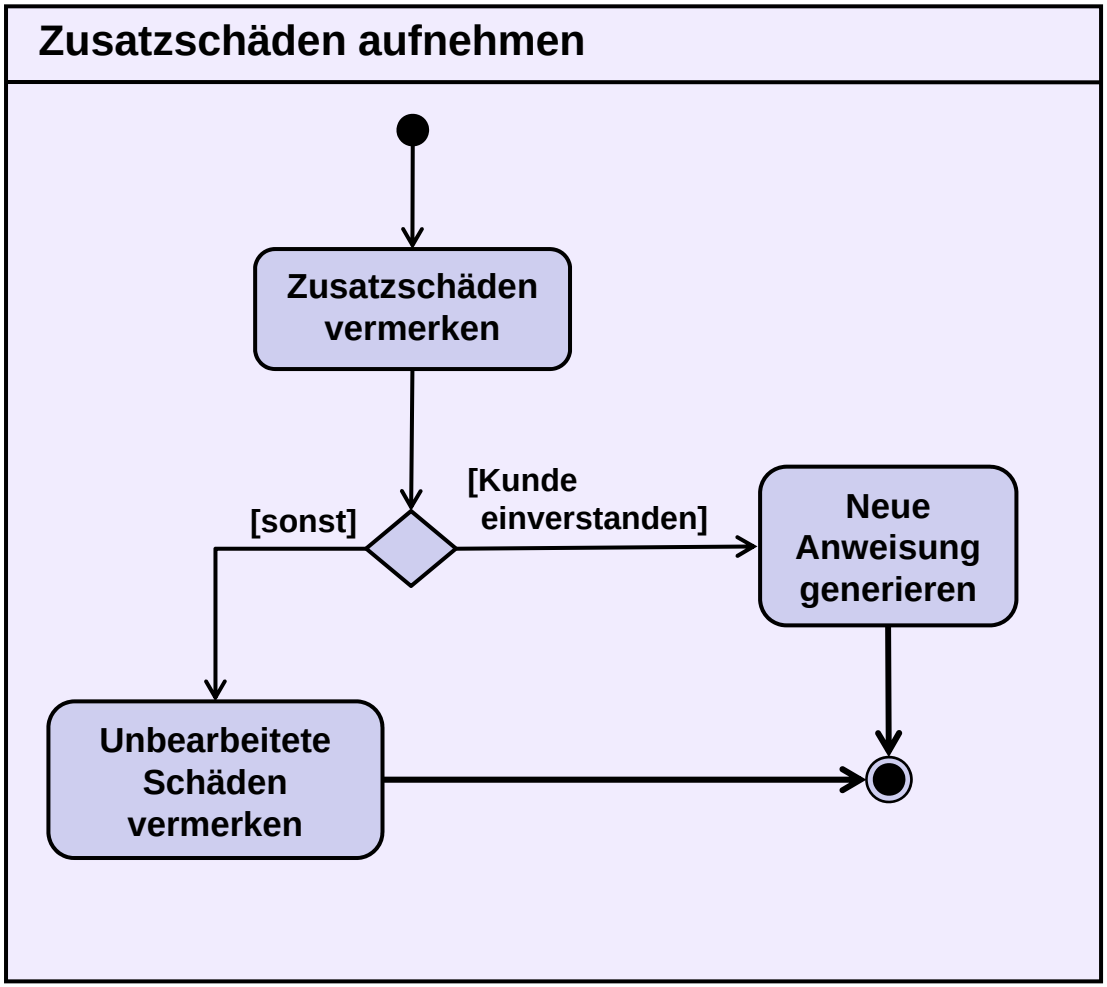
Erinnerung: Use-Case-Modell



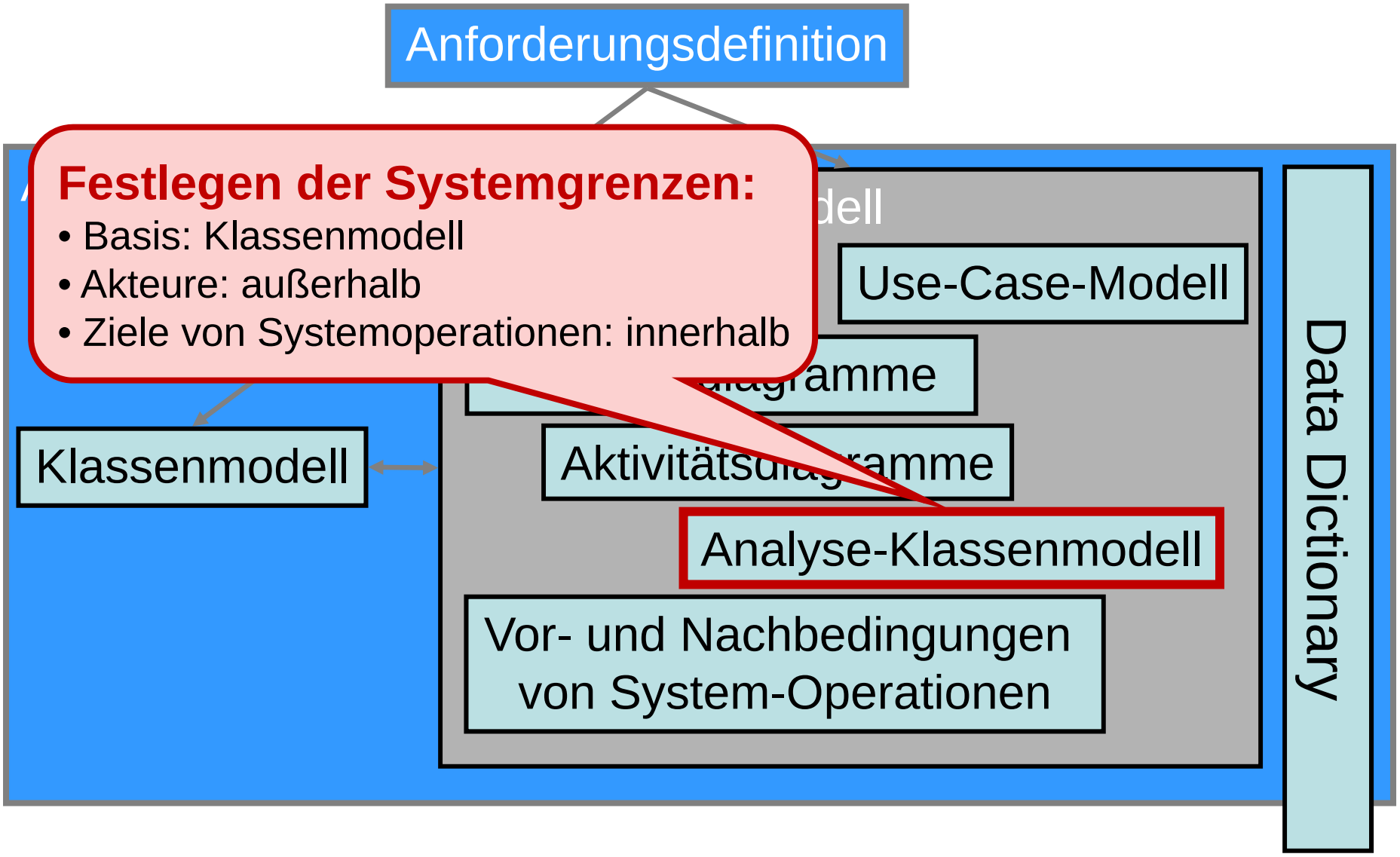
Aktivitätsdiagramm für UC Anweisung bearbeiten



Verfeinerte Aktivität: Zusatzschäden Aufnehmen



Analyse, 5. Schritt: Systemklassenmodell



Klassen im Systemklassenmodell

RUP unterscheidet drei Arten von Klassen

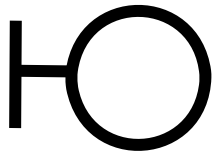
1. Übergangsklassen **<<Boundary>>**
2. Gegenstandsklassen **<<Entity>>**
3. Steuerungsklassen **<<Control>>**

Stereotypen zur Unterscheidung

- eine Art Etikett, auch als Bild darstellbar
- Qualifizierende Ergänzung eines Modellelements
- UML/RUP gibt eine Reihe von Stereotypen vor

Boundary-Klassen (Übergangsklassen)

- Kapselung des Systems zu seiner Umwelt
- **Präsentationsschicht** in Form von GUI-Komponenten

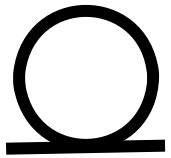


- Schnittstellen zu anderen Systemen
- Sensoren oder Schalter zur Steuerung externer Geräte
- eine Boundary-Klasse pro Akteur

Akteure kommunizieren nur mit ihrer Boundary-Klasse

Entity-Klassen (Gegenstandsklassen)

- zur Speicherung von Daten, Informationen,
  Datenhaltungsschicht
- persistente Datenhaltung,
  z.B. durch Zugriffe auf Datenbank



- werden von Steuerungsklassen angesprochen
- z.B. Daten der Akteure durch Entity-Klassen beschrieben

Für Akteure prüfen, ob ihre Daten in einer Entity-Klasse gespiegelt werden müssen.

Control-Klassen (Steuerungsklassen)

- Kapselung von Abläufen und Geschäftslogik in einer **Logikschicht**
- Bindeglied zwischen Boundary- und Entity-Klassen
- Modellieren komplexe Funktionalitäten (Algorithmen), die keiner anderen Klasse zugeordnet werden können.

Für jeden Use-Case prüfen,
ob eine Control-Klasse einzuführen ist.

Tips zur Erstellung des Systemklassenmodells

- Ableitung aus dem Klassenmodell für den Gegenstandsbereich
  - Einige Klassen sind offensichtlich Daten im System.
  - Für andere kann sich die Klasseninterpretation von „reales Objekt“ zu „Datensatz“ ändern.
  - Klassen an der Systemgrenze werden eventuell in interne und externe zerlegt.

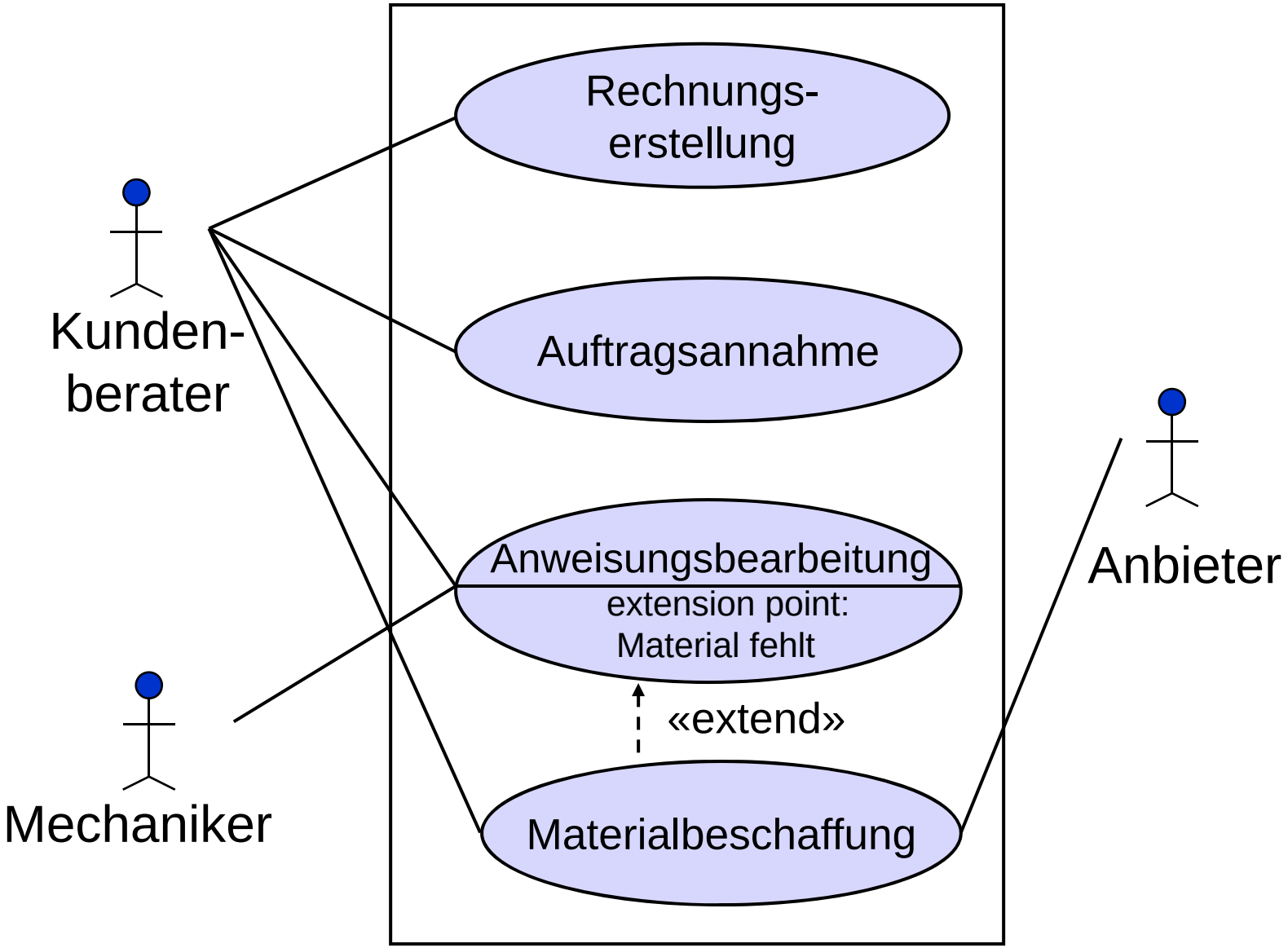
- Unterscheidung zwischen Klassen in und außerhalb des Systems wird aus

  Use Cases und Szenarien abgeleitet

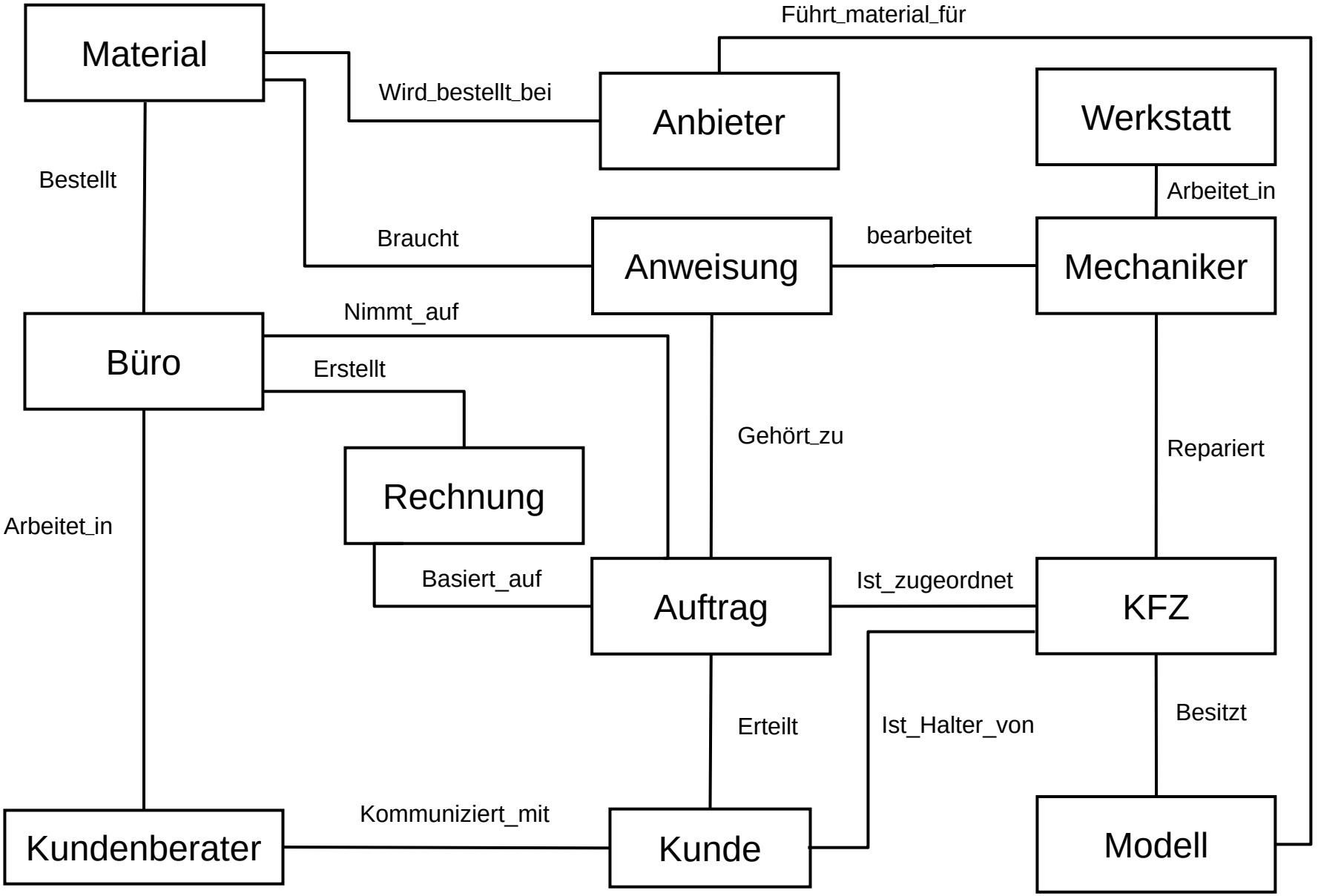
Vorgehen: Erstellung Systemklassenmodelle

1. **Identifiziere Akteure im Klassenmodell**
   - nur, wer direkt mit dem System interagiert
2. **Schaffe Boundary-Klassen für Akteure**
   - repräsentiert Benutzerschnittstelle
3. **Identifiziere Entity-Klassen für das System!**
   - Prüfen der Semantik von Klassen (z.B. Material-Art?)
4. **Bilde Entity-Klassen für Akteure, falls nötig!**
   - Datenspiegelung von Personen im System
5. **Streiche Assoziationen oder biege diese um!**
   - Assoziationen von Akteuren nur zur Boundary
6. **Füge Control-Klassen für Use-Cases ein!**
   - Klassen zwischen Boundary- und Entity-Klassen
7. **Vervollständige Attribute in den Klassen**

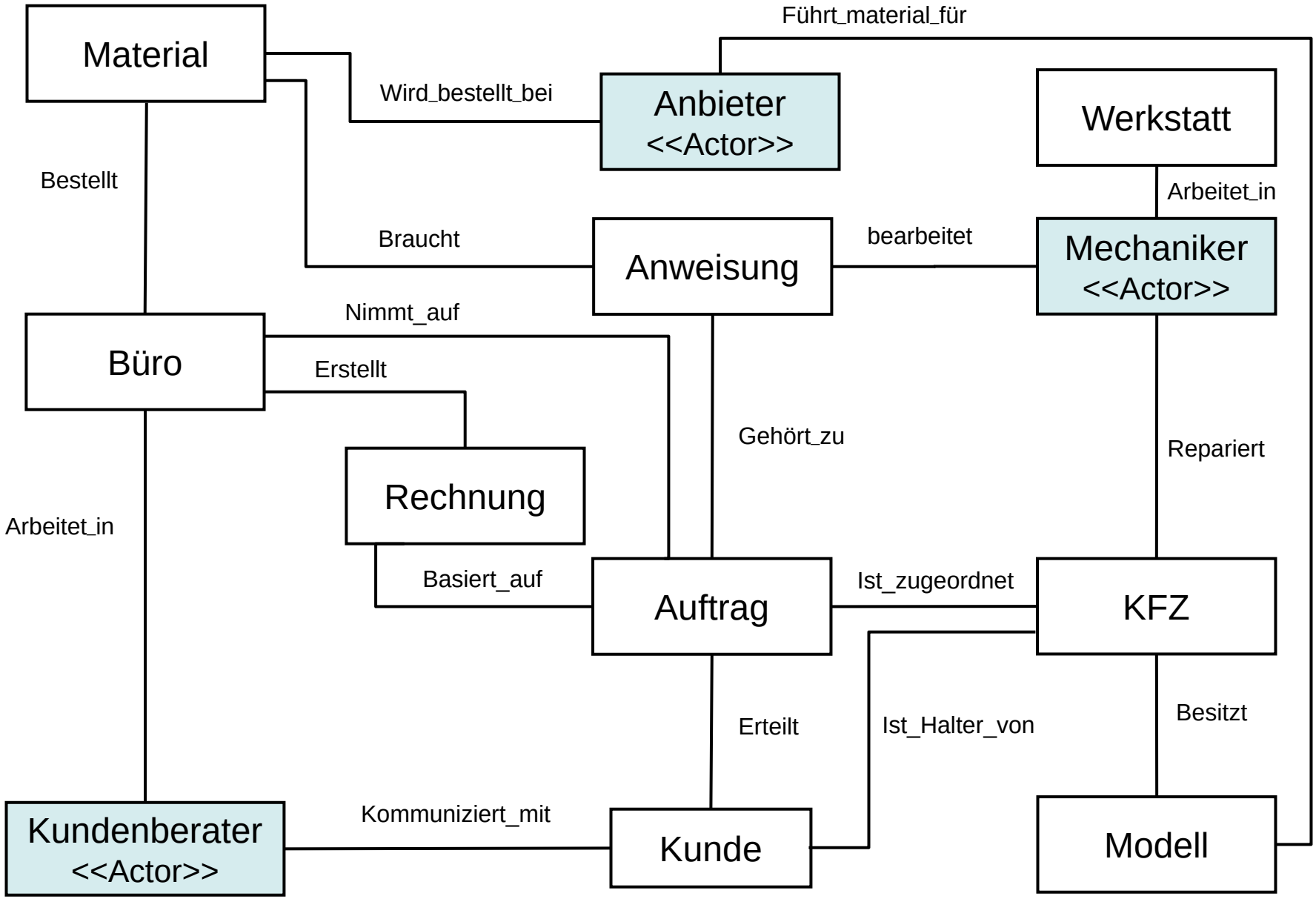
Rückblick: Use-Case-Modell der Werkstatt



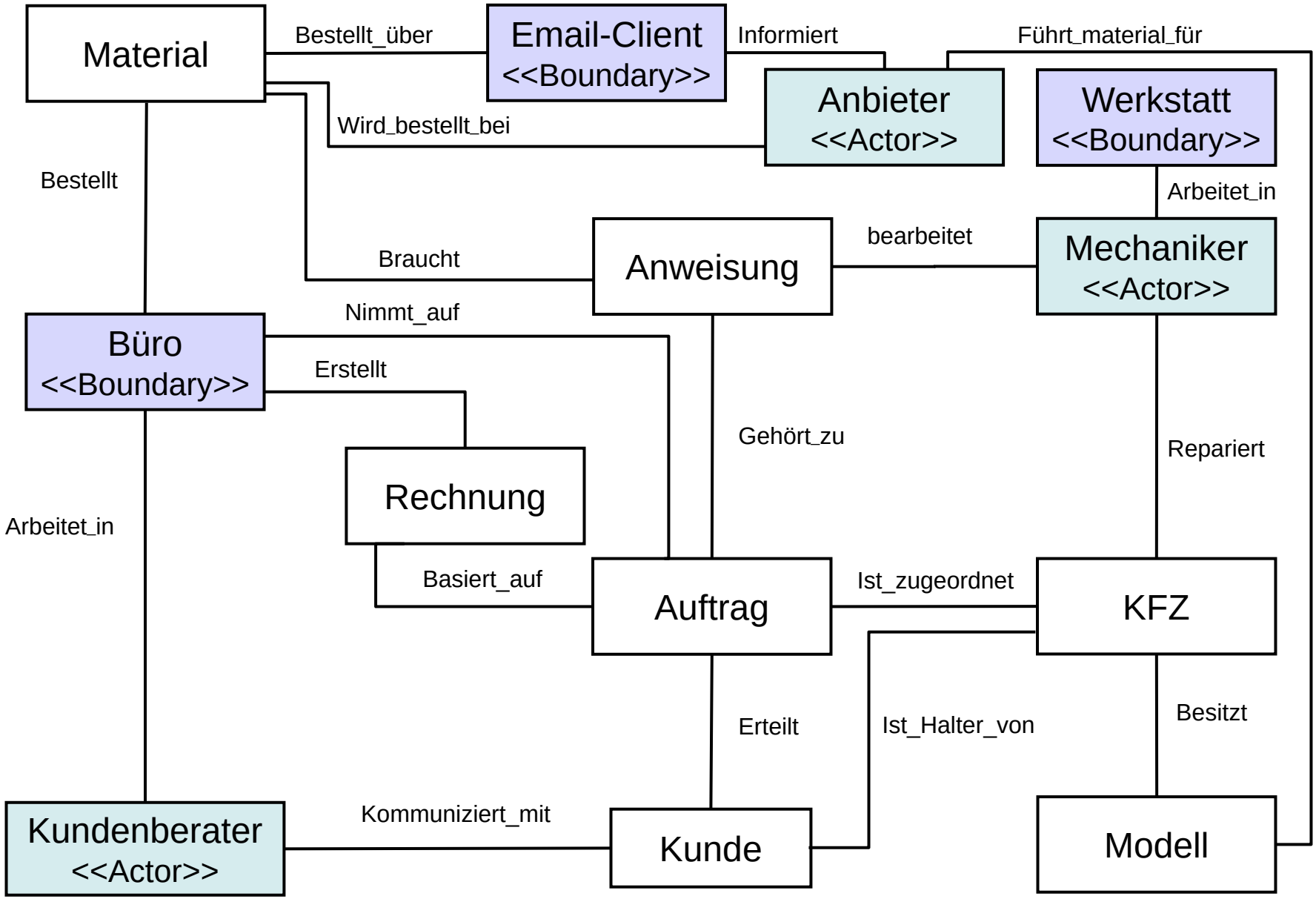
Rückblick: Klassenmodell der Werkstatt



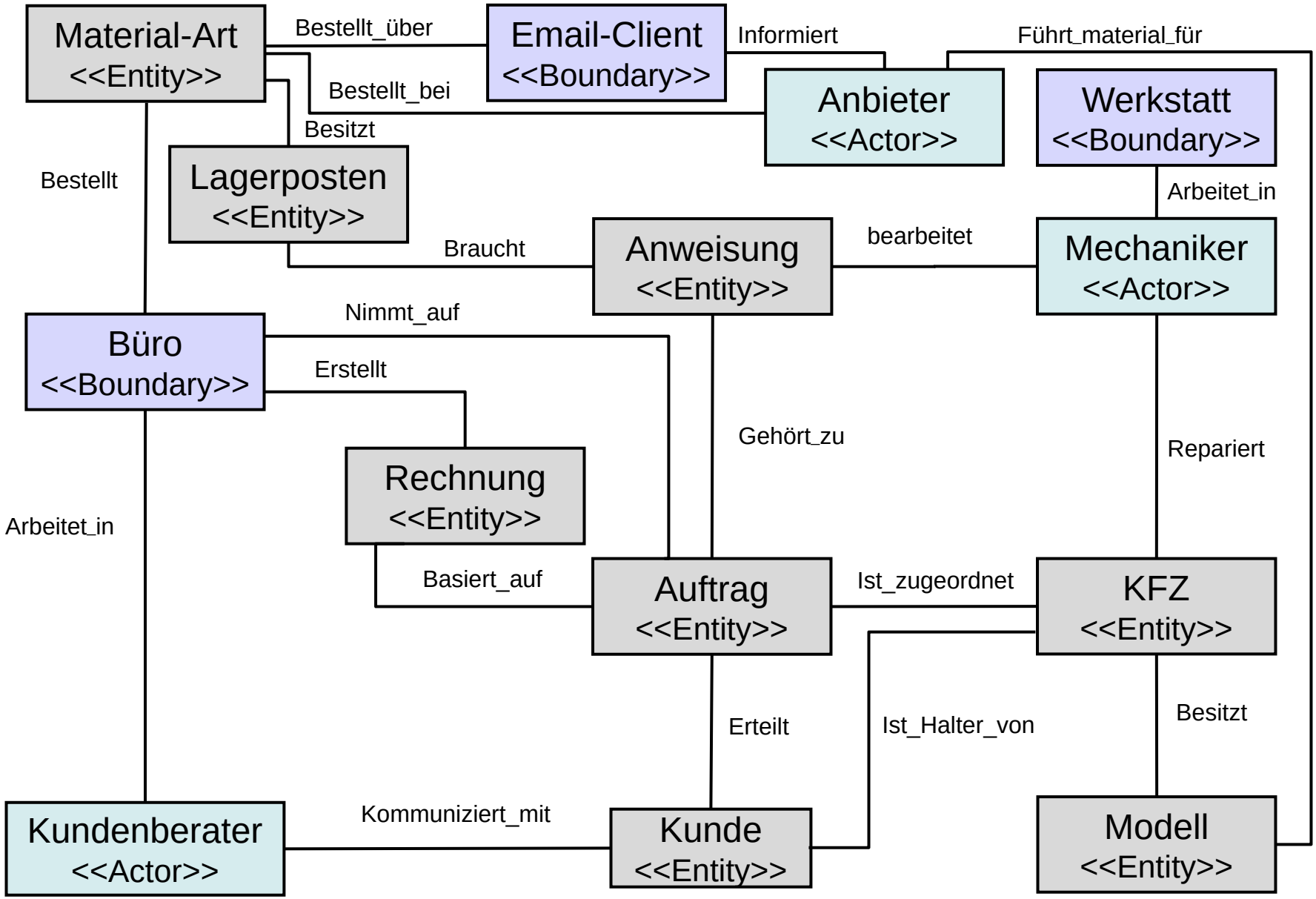
1. Identifiziere Akteure im Klassenmodell!



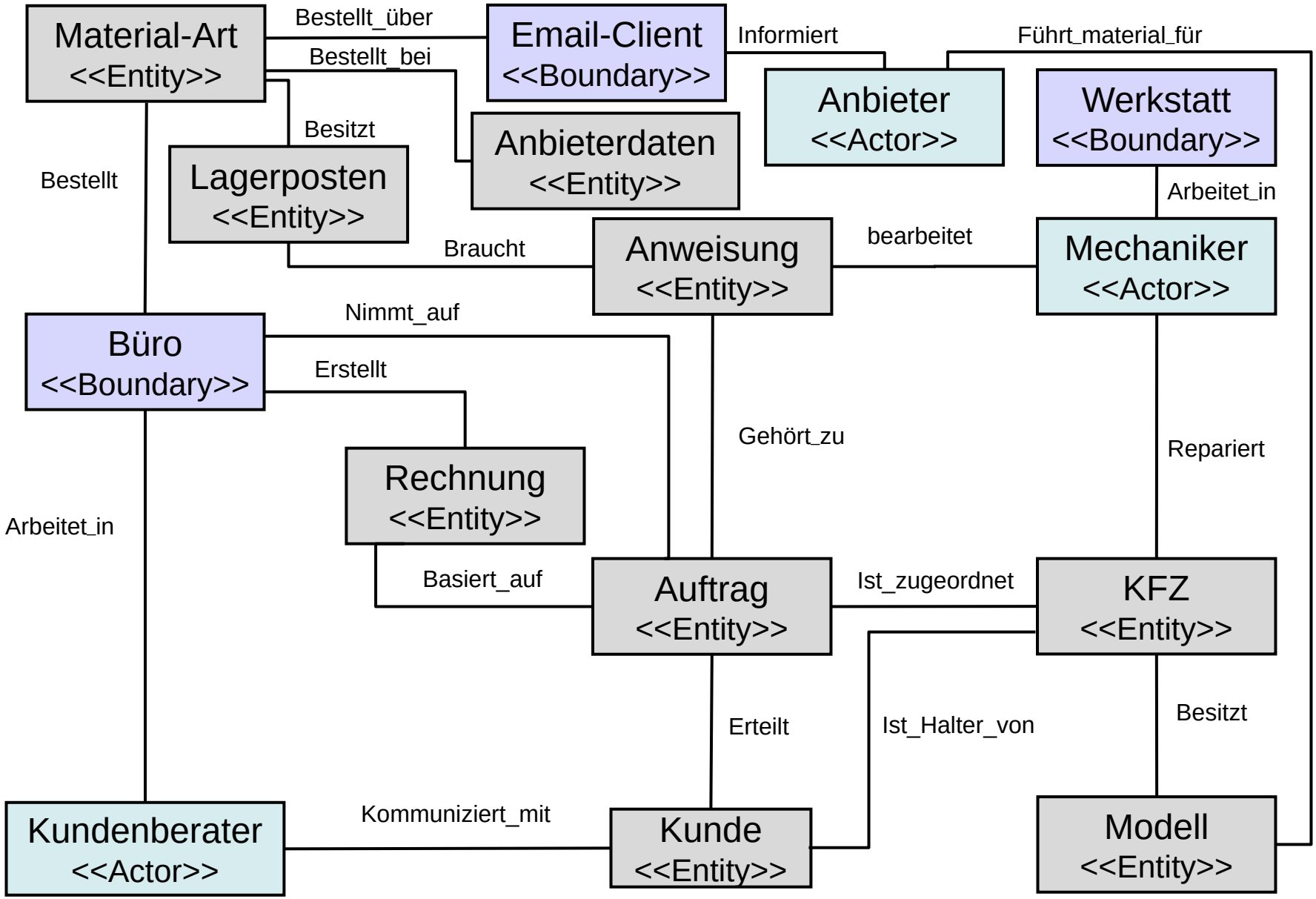
2. Schaffe Boundary-Klassen für Akteure!



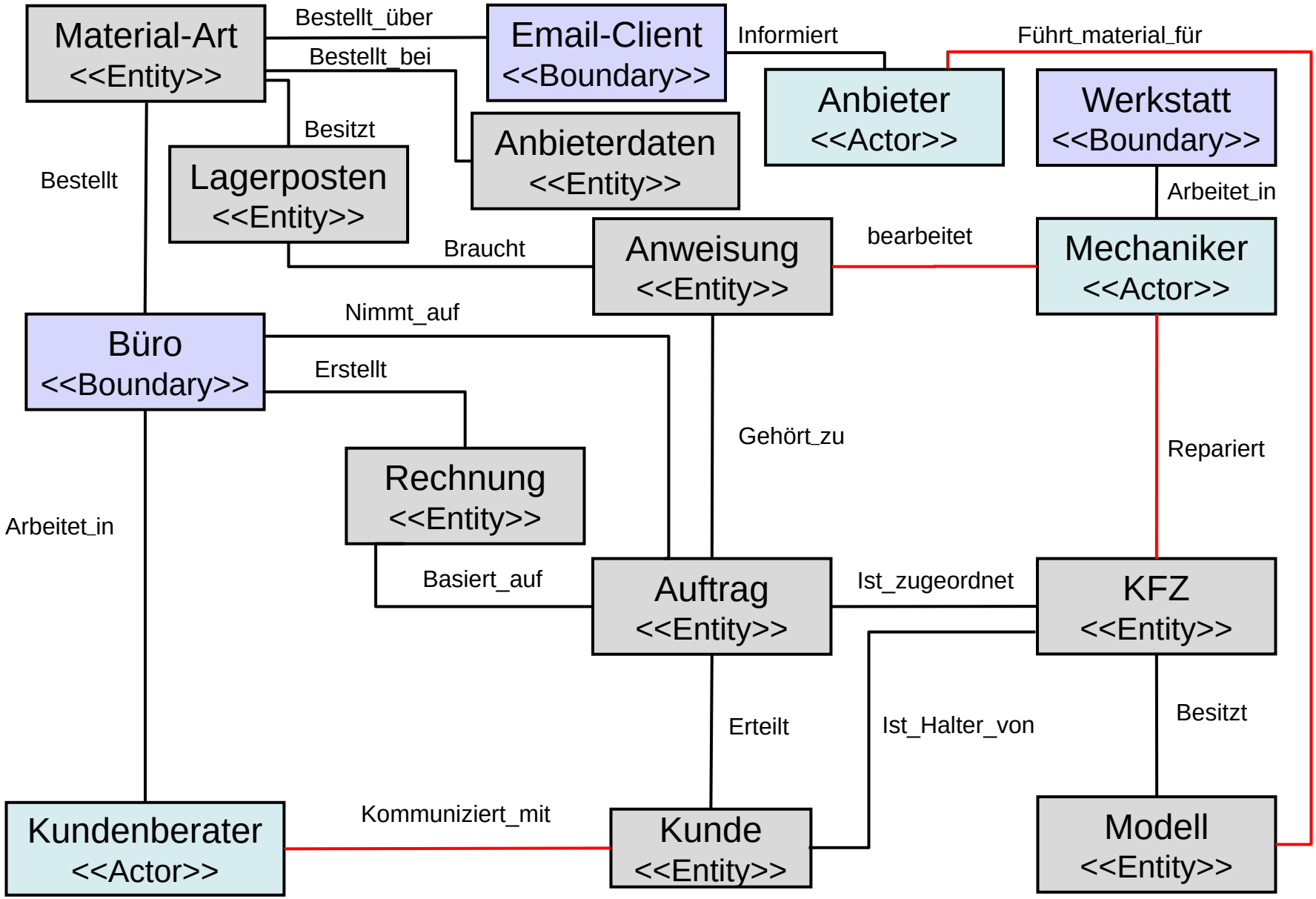
3. Identifiziere Entity-Klassen für das System!



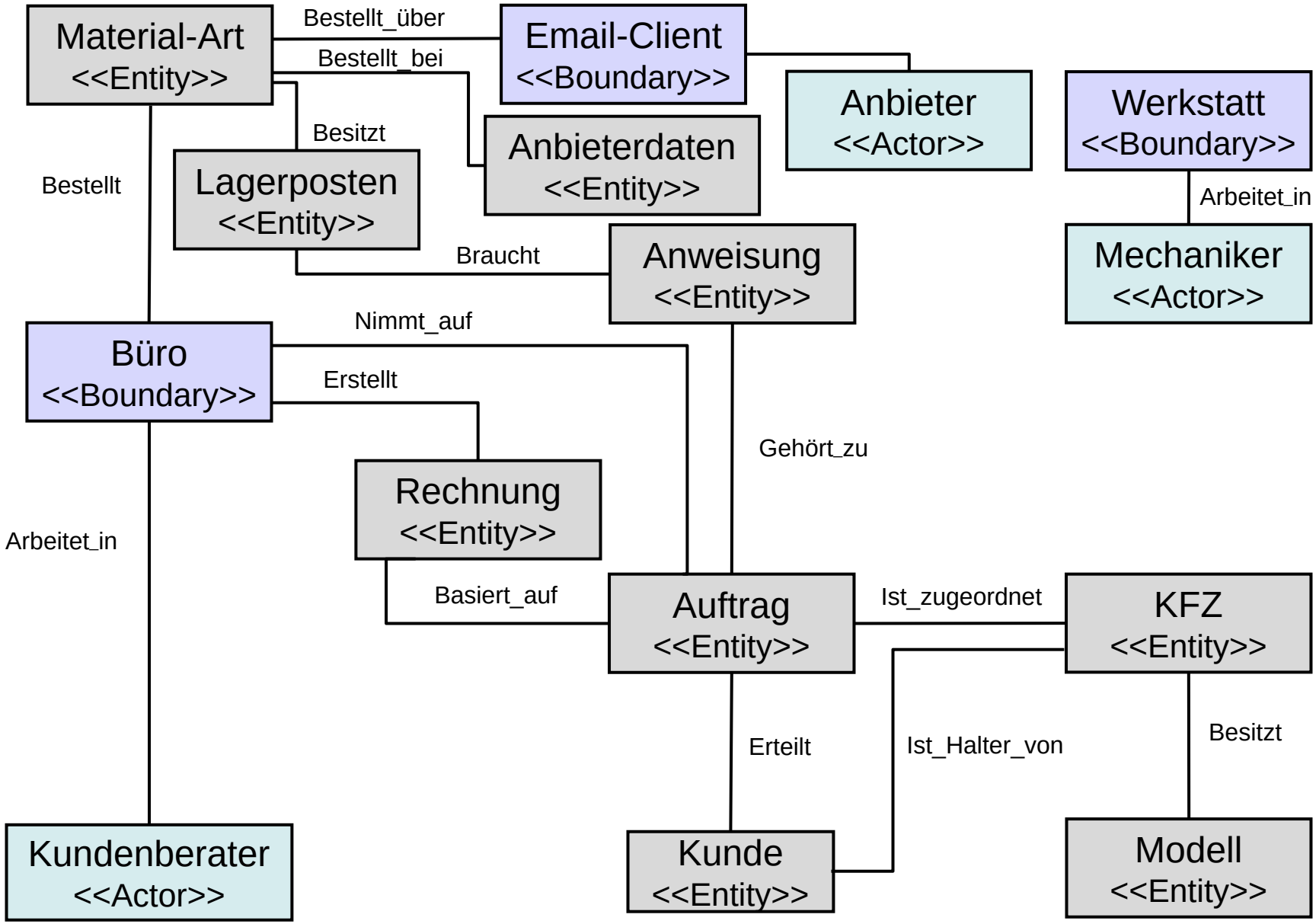
4. Bilde Entity-Klassen für Akteure, falls nötig!



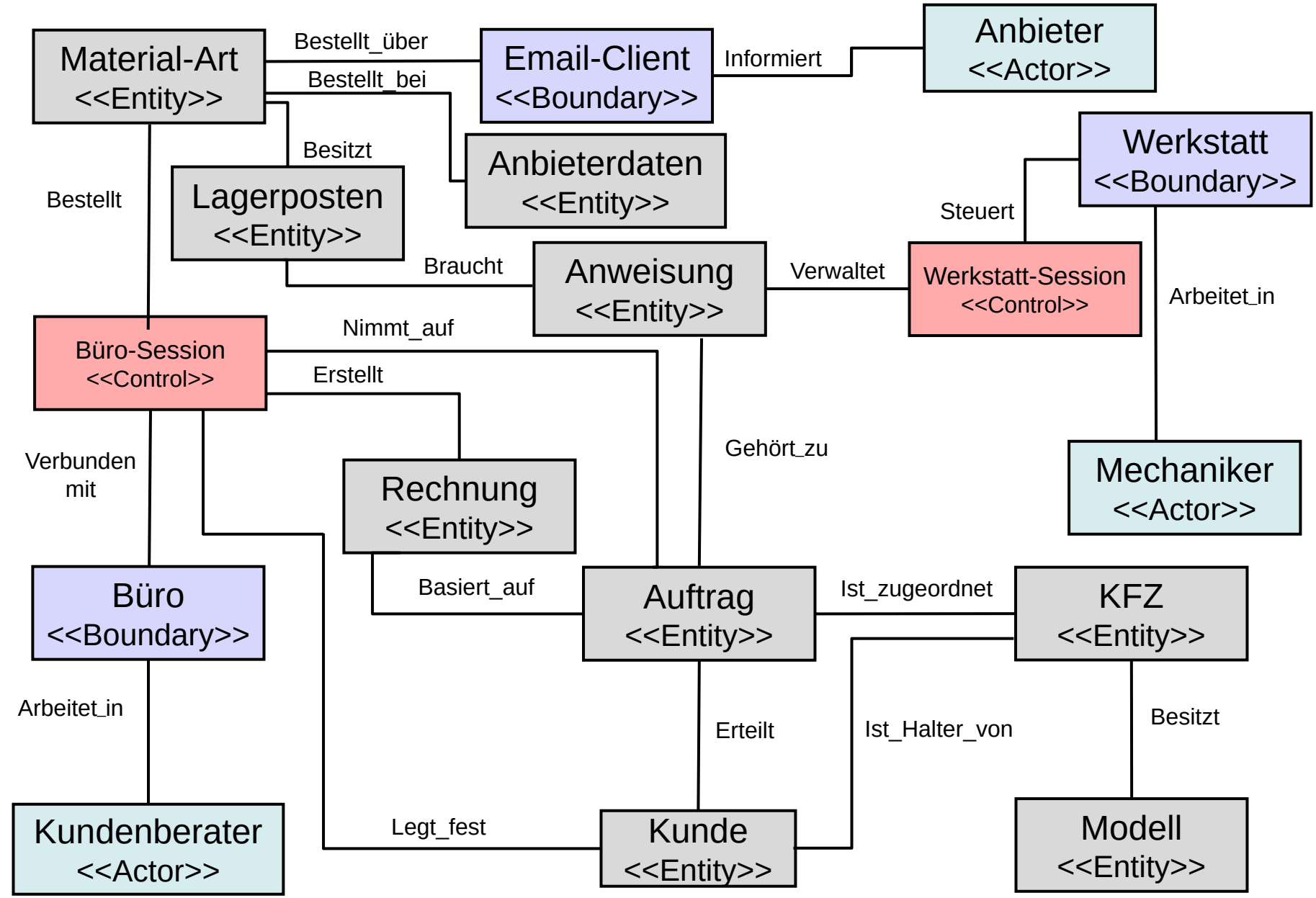
Kommunizieren Akteure nur über Boundary?



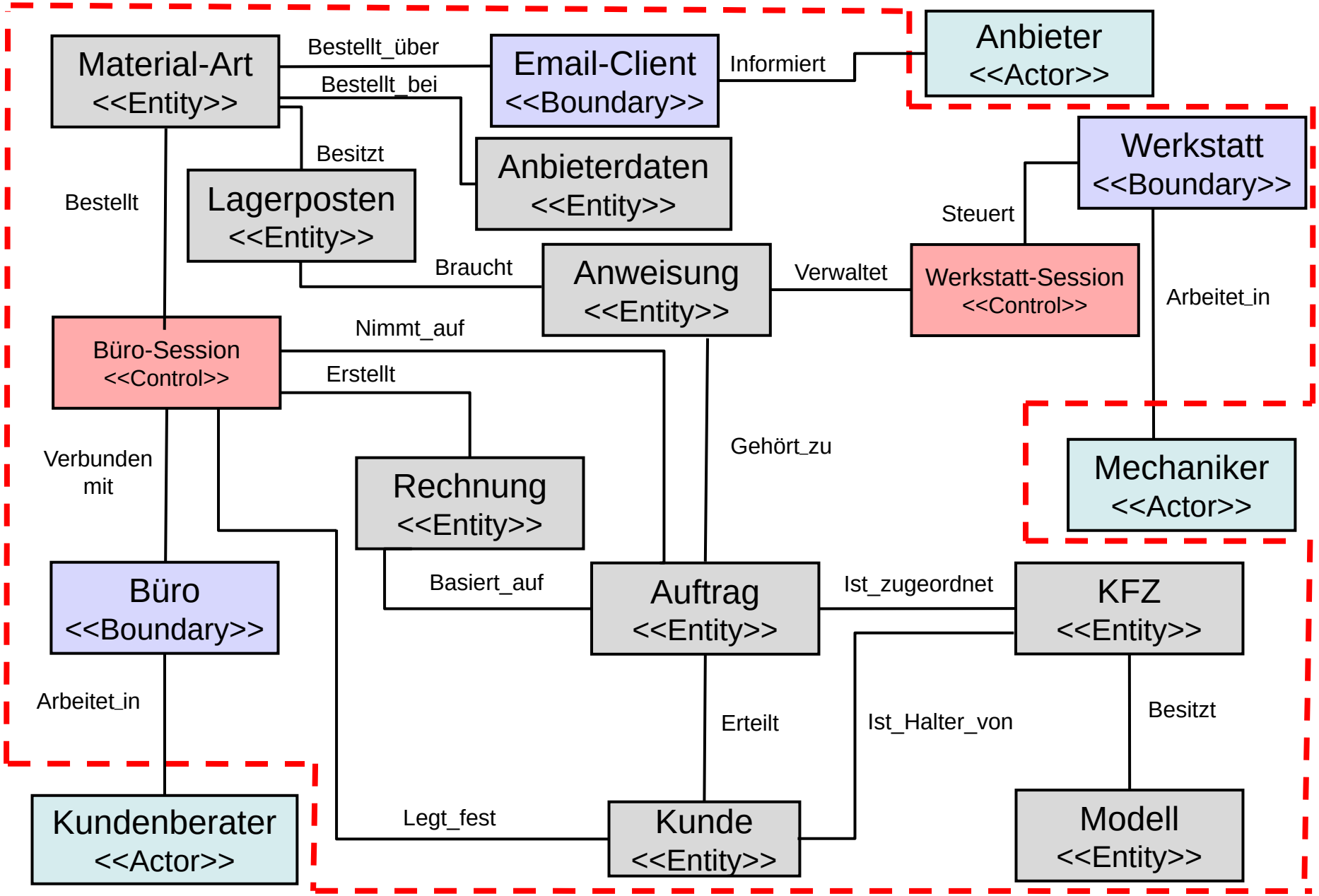
5. Entferne notfalls verbotene Assoziationen!



6. Füge Control-Klassen für Use-Cases ein!



Ergebnis: Eine virtuelle Systemgrenze entsteht!



7. Vervollständige Attribute in den Klassen!

Anbieterdaten	Anweisung	Auftrag	Büro	KFZ
adresse : ADRESSE	schäden : TEXT anid : ANID zusatzschäden : TEXT status : STATUS std : NAT	aid : AID ist_beendet : BOOL ist_probefahrt _erfolgreich : BOOL hat_zusatzschäden : BOOL	stdsatz : NAT	kennzeichen : KENNZEICHEN kfzid : KFZID

Kundendaten	Material-Art	Modell	Rechnung	Werkstatt
kdat : KDATEN kundenid : KUNDENID	matid : MATID preis : NAT	mid : MODID mdat : MDAT	betrag : NAT	stdsatz : NAT

Typdefinitionen:

ADRESSE	- Internetadresse		MID	-	Identifier des Modells
ANID	- Identifier derAnweisung		MDAT	-	Modelldaten
STATUS	- bereit_zur_bearbeitung | in_bearbeitung | beendet		NAT	-	natürliche Zahl
			BOOL	-	boolscher Wert
AID	- Identifier des Auftrags		KFZID	-	Identifier des KFZ
KENNZEICHEN	- KFZ-Kennzeichen		KUNDENID	-	Identifier des Kunden
KDATEN	- Kundendaten		TEXT	-	Text
MATID	- Identifier des Materials				

Was haben wir bis jetzt erreicht?