

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)

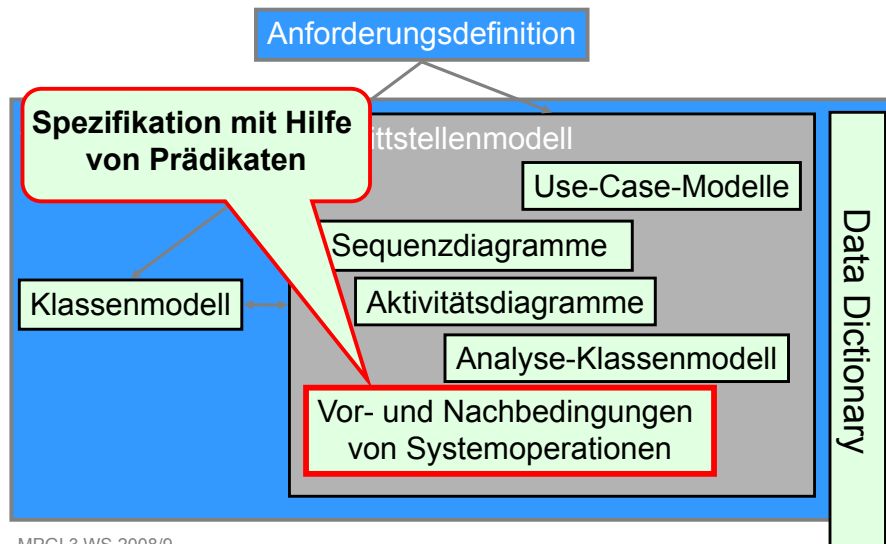
Was machen wir heute?

- **Rückblick Analyse**
 - Operationsschema am Beispiel der Werkstatt
- **Entwurf**
 - Überblick
 - **Kommunikationsdiagramme**
 - Beispiele für die Werkstatt

MPGI 3 WS 2008/9

2

Analyse, letzter Schritt: **Operationsschema**



MPGI 3 WS 2008/9

Zusammenfassung: **Operationsmodell**

- Spezifikation von Vor- und Nachbedingungen auf dem Systemzustand
- Erstellung eines **Operationsschemas** für **jede** Systemoperation
- Methodische Einordnung zwischen Analyse- und Entwurfsphase
- Spezifikation der Prädikate auf Basis von Z

MPGI 3 WS 2008/9

4

Aufbau eines Operationsschemas

Operation	=	Name der Operation
Description	=	informelle Beschreibung
Input	=	Eingabeparameter
Sends	=	erzeugte Ausgabeereignisse
Reads	=	(nur) gelesene Objekte / Attribute
Changes	=	(möglicherweise) geänderte Objekte / Attribute
Pre	=	Annahmen an den Vorzustand
Post	=	Beziehung zwischen Vor- und Nachzustand

Notationen für Operationsschema

Changes = a: Auftrag type	Grundtyp einer Klasse: Deklaration von neu zu erzeugenden Objekten
Reads = a: Auftrag with a.id = aid	Qualifikation von Variablen: Deklaration von Variablen mit zusätzlichen Bedingungen (\Rightarrow implizite Vorbedingung)
Pre = implicit	Implizite Vorbedingungen falls with-Konstrukt in Reads- oder Changes
Changes = a: Auftrag type Post = a new	Erzeugen von Objekten: Abkürzungen für Auftrag' = Auftrag \cup { a }
Changes = a: Auftrag Post = a delete	Löschen von Objekten: Abkürzungen für Auftrag' = Auftrag \setminus { a }
Sends = Mechaniker :{Nachricht} Post = is_sent {Nachricht}	Versenden von Nachrichten: Sende Nachricht an Mechaniker
Changes = a: Auftrag Post = C ₁ \Rightarrow a.status' = aktiv C ₂ \Rightarrow no_effect	Nachbedingungen (partiell) ohne Effekt: Abkürzung für a.status' = a.status

Notationen in Z: Schreibweise für Quantoren

Grundaufbau:

$$\forall x:T \bullet Qx \text{ oder } \exists x:T \bullet Qx$$

Beispiel:

$$\forall x:\mathbb{N} \bullet x > 5 \text{ oder } \exists x:\mathbb{N} \bullet x < 0$$

Optionale Erweiterung:

$$\forall x:T \mid Px \bullet Qx \text{ oder } \exists x:T \mid Px \bullet Qx$$

Beispiel:

$$(\forall x:\mathbb{N} \mid x < 2 \bullet Qx) \Leftrightarrow (\forall x:\{0,1\} \bullet Qx)$$

Notationen in Z: Schreibweise für Mengen

Grundaufbau *set-comprehension*:

$$\{x:T \mid Px\}$$

Beispiel:

$$\{x:\mathbb{N} \mid x < 5\} = \{0,1,2,3,4\}$$

Optionale Erweiterung:

$$\{x:T \mid Px \bullet Qx\}$$

Beispiel:

$$\{x,y:\mathbb{N} \mid x < 5 \wedge y < 1 \bullet x * y\} = \{0\}$$

Notationen in Z: Fallunterscheidungen

Grundaufbau:

$$(C_1 \Rightarrow P_1) \wedge (C_2 \Rightarrow P_2) \wedge \dots$$

Alternativ *if-then-else* Konstrukte:

if C_1 then P_1 else (if C_2 then P_2 else ...)

Beispiel für **notwendig disjunkte** Bedingungen:

$$(C_1 \Rightarrow x' = x) \wedge (C_2 \Rightarrow x' = \neg x)$$

Erstellung eines Operationsschemas

1.	Operation	=	Name der Operation
	Description	=	informelle Beschreibung
	Input	=	Eingabeparameter
	Sends	=	erzeugte Ausgabeereignisse
	Reads	=	(nur) gelesene Objekte / Attribute
	Changes	=	(möglicherweise) geänderte Objekte / Attribute
	Pre	=	Annahmen an den Vorzustand
	Post	=	Beziehung zwischen Vor- und Nachzustand

Schritt 1: Name und Informelle Beschreibung

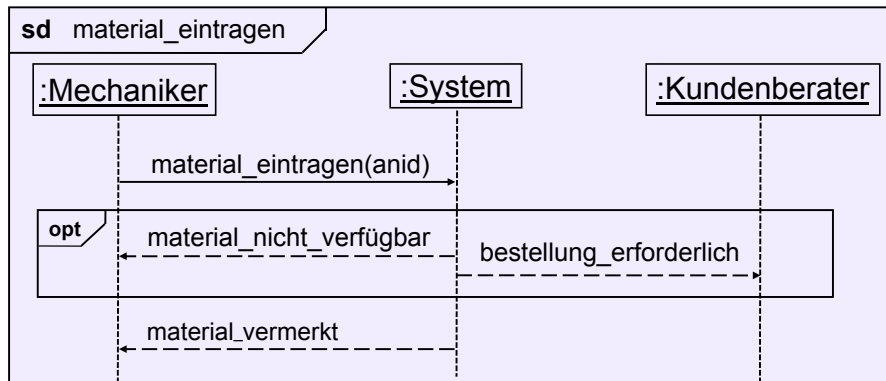
Operation = material_eintragen

Description = Ein Mechaniker stellt bei der Bearbeitung einer Anweisung fest, dass Material benötigt wird. Er **trägt** dieses **Material unter Angabe der** benötigten **Anzahl** im System **ein**. **Sollte** im Lager **nicht genug** Material **verfügbar sein**, **so wird** das **Büro informiert** und dem Mechaniker eine entsprechende Meldung gegeben.

Erstellung eines Operationsschemas

	Operation	=	Name der Operation
	Description	=	informelle Beschreibung
2.	Input	=	Eingabeparameter
	Sends	=	erzeugte Ausgabeereignisse
	Reads	=	(nur) gelesene Objekte / Attribute
	Changes	=	(möglicherweise) geänderte Objekte / Attribute
	Pre	=	Annahmen an den Vorzustand
	Post	=	Beziehung zwischen Vor- und Nachzustand

Einschub: Betrachte Sequenzdiagramm



Schritt 2: Eingaben und Ausgabeereignisse

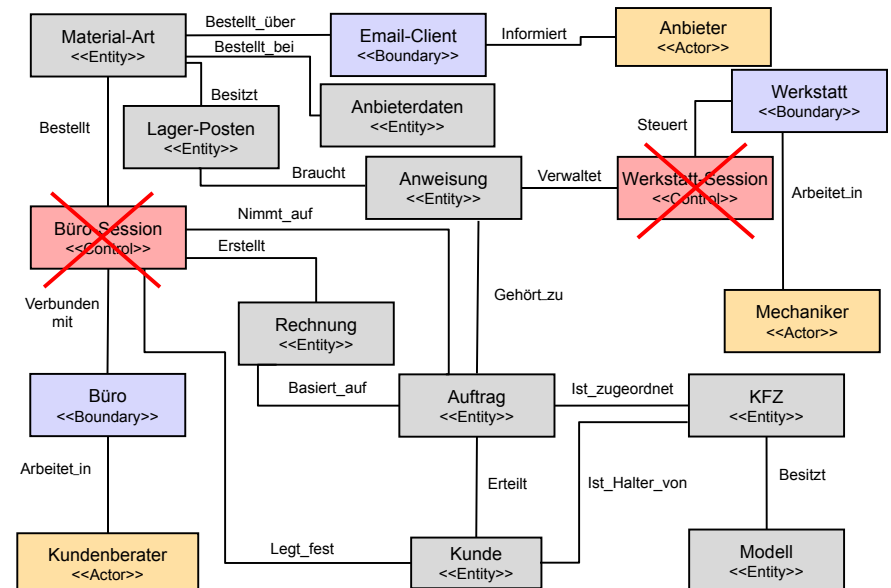
Input = anweisung_id : ANID,
 material_id : MATID,
 anzahl : NAT

Sends = Mechaniker : { material_nicht_verfügbar,
 material_vermerkt },
 Kundenberater : { bestellung_erforderlich }

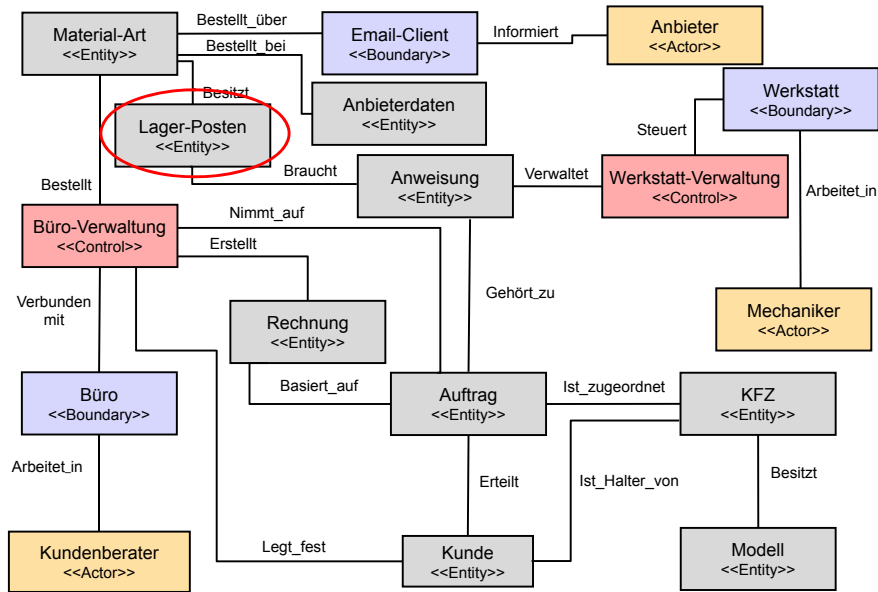
Erstellung eines Operationsschemas

- Operation** = Name der Operation
- Description** = informelle Beschreibung
- Input** = Eingabeparameter
- Sends** = erzeugte Ausgabeereignisse
- Reads** = (nur) gelesene Objekte / Attribute
- 3. Changes** = (möglicherweise) geänderte Objekte / Attribute
- Pre** = Annahmen an den Vorzustand
- Post** = Beziehung zwischen Vor- und Nachzustand

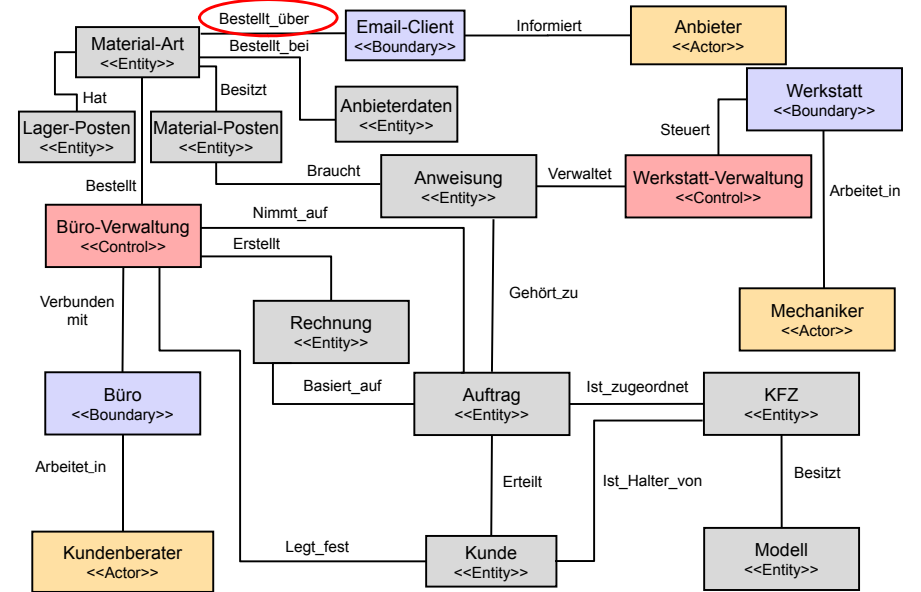
Einschub: Betrachte Systemklassenmodell



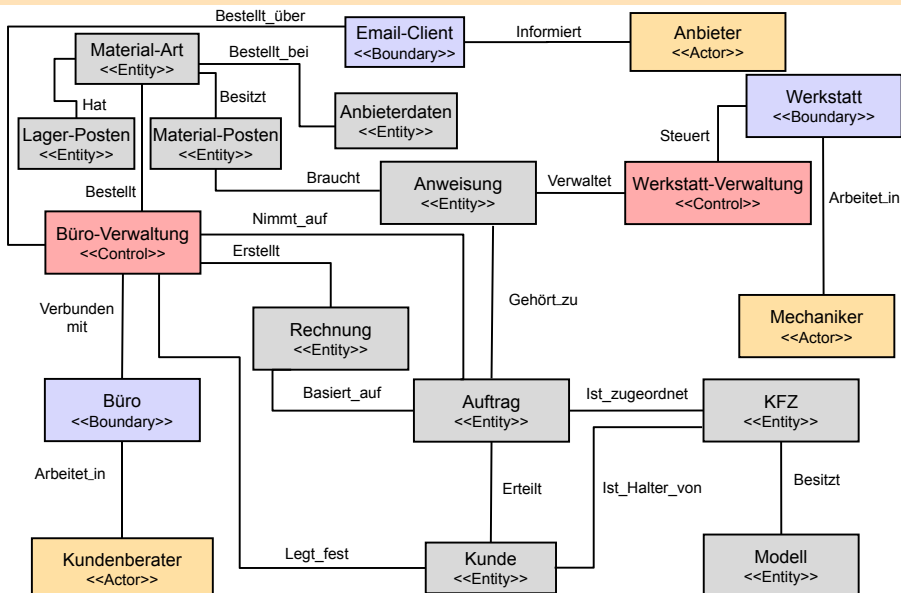
Einschub: Verändertes Systemklassenmodell



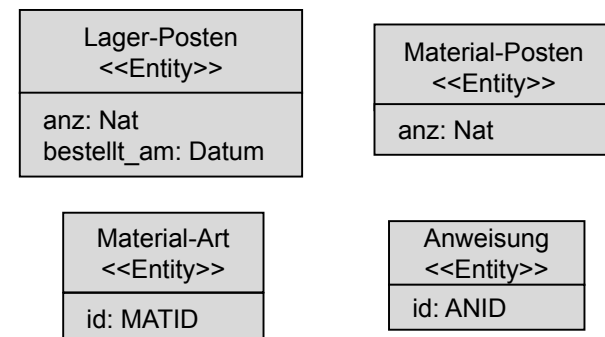
Einschub: Verändertes Systemklassenmodell



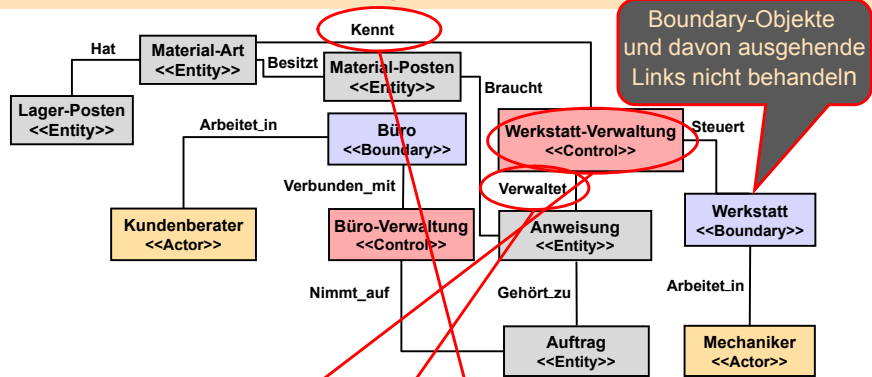
Einschub: Verändertes Systemklassenmodell



Einschub: Relevante Attribute im Klassenmodell

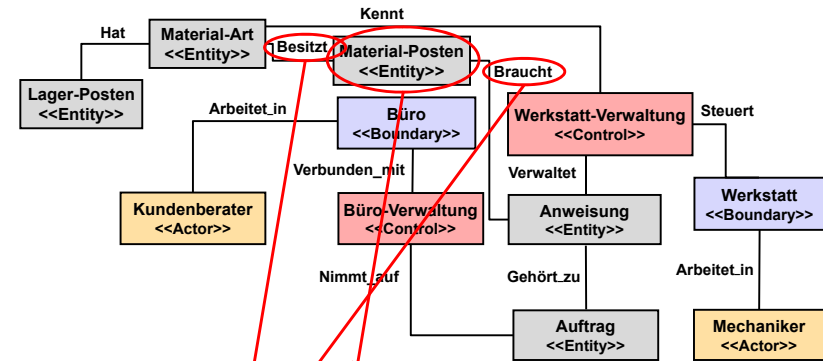


Schritt 3: Ableitung des Reads-Teiles



Reads = w : Werkstatt-Verwaltung, Anweisung, Lager-Posten, Verwaltet, Kennt, Material-Art, Hat,
 an : Anweisung **with** an.id = anweisung_id \wedge (w,an) \in Verwaltet
 mat : Material-Art **with** mat.id = material_id \wedge (w,mat) \in Kennt

Schritt 3: Ableitung des Changes-Teiles



Changes = mp: Material-Posten **type**, Material-Posten
 Braucht, Besitzt

Erstellung eines Operationsschemas

- Operation** = Name der Operation
 - Description** = informelle Beschreibung
 - Input** = Eingabeparameter
 - Sends** = erzeugte Ausgabeereignisse
 - Reads** = (nur) gelesene Objekte / Attribute
 - Changes** = (möglicherweise) geänderte Objekte / Attribute
- Pre** = Annahmen an den Vorzustand
 - 4. Post** = Beziehung zwischen Vor- und Nachzustand

Schritt 4: Ableitung des Pre- und Post-Teiles

Pre = **implicit**

Post = let

verfügbar == {
 benötigt == {
 anzahl_{LP} == \sum_{LP} verfügbar,
 anzahl_{MP} == \sum_{MP} benötigt
 • (anzahl_{LP} < anzahl_{MP} + anzahl \Rightarrow
 (**is_sent** { material_nicht_verfügbar } \wedge
 is_sent { bestellung_erforderlich })) \wedge
 mp **new** \wedge mp.anz' = anzahl \wedge
is_sent { material_vermerkt } \wedge
 Braucht' = Braucht \cup { (an, mp) } \wedge
 Besitzt' = Besitzt \cup { (mp, mat) }

Es gibt implizite Vorbedingungen aus Reads oder Changes (with) Besitzt};

Schritt 4: Ableitung des Pre- und Post-Teiles

Pre = **implicit**

Post = **let**

verfügbar == { l : Lager-Posten | (l,m) ∈ Hat };
benötigt == { m : Material-Posten | (m,mat) ∈ Besitzt };
anzahl_{LP} == ∑_{LP} verfügbar;
anzahl_{MP} == ∑_{MP} benötigt

(anzahl_{LP} < anzahl_{MP} + anzahl ⇒
(**is_sent** { material_nicht_verfügbar } ∧
is_sent { bestellung_erforderlich })) ∧
mp **new** ∧ mp.anz' = anzahl ∧
is_sent { material_vermerkt } ∧
Braucht' = Braucht ∪ { (an, mp) } ∧
Besitzt' = Besitzt ∪ { (mp, mat) }

Abkürzungen mit
let-Konstrukt einführen!

25

Schritt 4: Ableitung des Pre- und Post-Teiles

Pre = **implicit**

Post = **let**

verfügbar == { l : Lager-Posten | (l,m) ∈ Hat };
benötigt == { m : Material-Posten | (m,mat) ∈ Besitzt };
anzahl_{LP} == ∑_{LP} verfügbar;
anzahl_{MP} == ∑_{MP} benötigt

(anzahl_{LP} < anzahl_{MP} + anzahl ⇒
(**is_sent** { material_nicht_verfügbar } ∧
is_sent { bestellung_erforderlich })) ∧
mp **new** ∧ mp.anz' = anzahl ∧
is_sent { material_vermerkt } ∧
Braucht' = Braucht ∪ { (an, mp) } ∧
Besitzt' = Besitzt ∪ { (mp, mat) }

Fallunterscheidung
zur Bestellung von Material!

26

Schritt 4: Ableitung des Pre- und Post-Teiles

Pre = **implicit**

Post = **let**

verfügbar == { l : Lager-Posten | (l,m) ∈ Hat };
benötigt == { m : Material-Posten | (m,mat) ∈ Besitzt };
anzahl_{LP} == ∑_{LP} verfügbar;
anzahl_{MP} == ∑_{MP} benötigt

(anzahl_{LP} < anzahl_{MP} + anzahl ⇒
(**is_sent** { material_nicht_verfügbar } ∧
is_sent { bestellung_erforderlich })) ∧
mp **new** ∧ mp.anz' = anzahl ∧
is_sent { material_vermerkt } ∧
Braucht' = Braucht ∪ { (an, mp) } ∧
Besitzt' = Besitzt ∪ { (mp, mat) }

Erstellung eines
neuen Objekts
und Initialisierung

27

Schritt 4: Ableitung des Pre- und Post-Teiles

Pre = **implicit**

Post = **let**

verfügbar == { l : Lager-Posten | (l,m) ∈ Hat };
benötigt == { m : Material-Posten | (m,mat) ∈ Besitzt };
anzahl_{LP} == ∑_{LP} verfügbar;
anzahl_{MP} == ∑_{MP} benötigt

(anzahl_{LP} < anzahl_{MP} + anzahl ⇒
(**is_sent** { material_nicht_verfügbar } ∧
is_sent { bestellung_erforderlich })) ∧
mp **new** ∧ mp.anz' = anzahl ∧
is_sent { material_vermerkt } ∧
Braucht' = Braucht ∪ { (an, mp) } ∧
Besitzt' = Besitzt ∪ { (mp, mat) }

Versenden einer
Nachricht

28

Schritt 4: Ableitung des Pre- und Post-Teiles

Pre = implicit

Post = let

verfügbar == { l : Lager-Posten | (l, m) ∈ Hat };
 benötigt == { m : Material-Posten | (m, mat) ∈ Besitzt };

anzahl_{LP} == \sum_{LP} verfügbar;

anzahl_{MP} == \sum_{MP} benötigt

Definition neuer Operatoren ?

(anzahl_{LP} < anzahl_{MP} + anzahl ⇒
 (is_sent { material_nicht_verfügbar } ∧
 is_sent { bestellung_erforderlich })) ∧

mp new ∧ mp.anz' = anzahl ∧
 is_sent { material_vermerkt } ∧

Braucht' = Braucht ∪ { (an, mp) } ∧

Besitzt' = Besitzt ∪ { (mp, mat) }

Erweitern von Assoziationen

29

Einschub: Definitionen für Operatoren in Z

$\sum_{LP} : P \text{ Lager-Posten} \rightarrow N$

$\sum_{LP}(\{\}) = 0$

∀ M : P₁ Lager-Posten •

∀ m : M •

$\sum_{LP}(M) = \sum_{LP}(M \setminus \{m\}) + m.anz$

**Axiomatische Beschreibung
ohne generische Typen!**

MPGI 3 WS 2008/9

30

Einschub: Definitionen für Operatoren in Z

**Axiomatische Beschreibung
mit generischen Typen!**

[T]

$\sum : P \text{ T} \times (\text{T} \rightarrow N) \rightarrow N$

∀ op : (T → N) •

$\sum(\{\}, op) = 0 \wedge$

∀ M : P₁ T •

∀ m : M •

$\sum(M, op) = \sum(M \setminus \{m\}, op) + op(m)$

MPGI 3 WS 2008/9

31

Anwendung von generischen Operatoren

Pre = implicit

Post = let

verfügbar == { l : Lager-Posten | (l, m) ∈ Hat };

benötigt == { m : Material-Posten | (m, mat) ∈ Besitzt };

anzahl_{LP} == \sum verfügbar (λ l : Lager-Posten • l.anz);

anzahl_{MP} == \sum benötigt (λ m : Material-Posten • m.anz)

(anzahl_{LP} < anzahl_{MP} + anzahl ⇒

(is_sent { material_nicht_verfügbar } ∧

is_sent { bestellung_erforderlich })) ∧

mp new ∧ mp.n' = anzahl ∧

is_sent { material_vermerkt } ∧

Braucht' = Braucht ∪ { (an, mp) } ∧

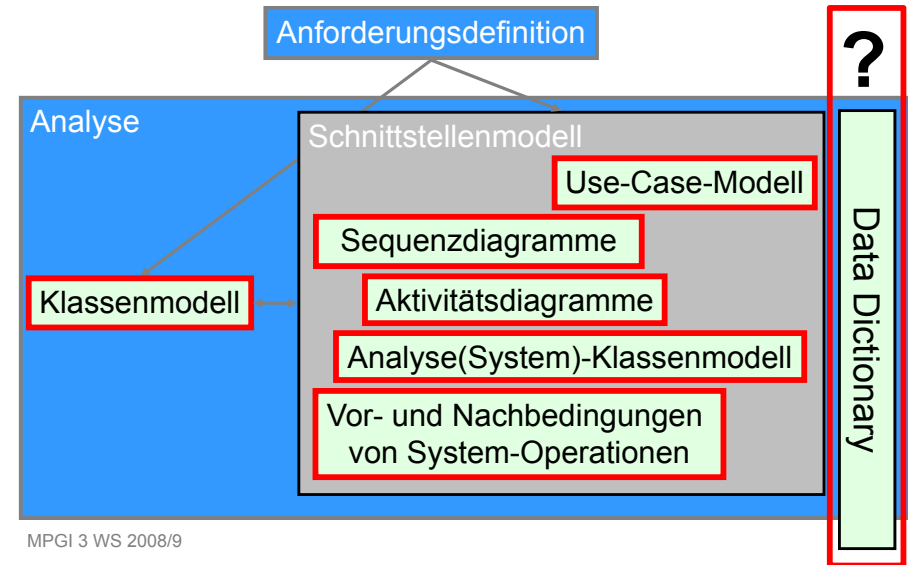
Besitzt' = Besitzt ∪ { (mp, mat) }

32

Die Analyse haben wir geschafft!

**Wir glauben jetzt zu wissen,
was das System tun soll.**

Diese Modelle dokumentieren unser Wissen ...



Aufbau des Datenlexikons (Data Dictionary)

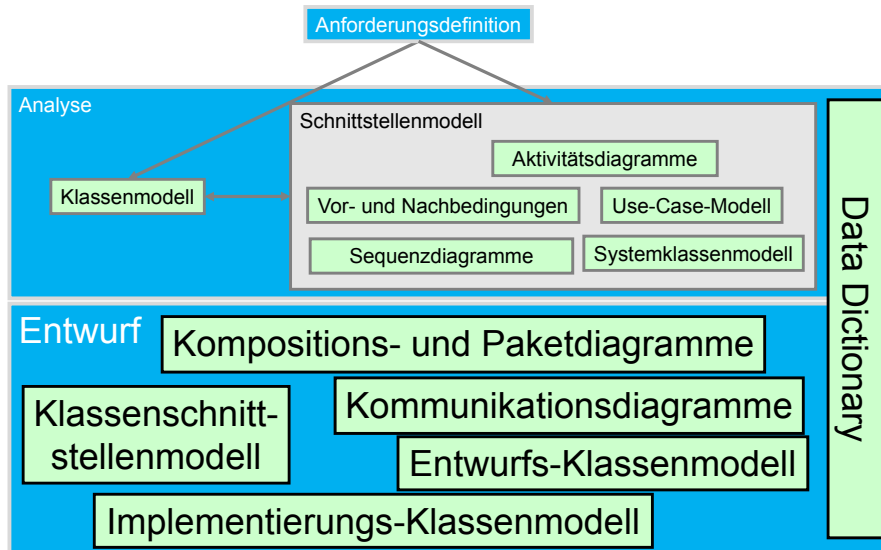
Name	Art	Beschreibung	Quelle
Büro	Boundary-Klasse	Schnittstelle, über die die Bürokräfte mit dem System kommunizieren	Systemklassenmodell
Werkstatt	Boundary-Klasse	Schnittstelle, über die die Mechaniker mit dem System kommunizieren	Systemklassenmodell
Auftrag	Entity-Klasse	Datenklasse, in der Informationen über einen Auftrag abgelegt sind	Systemklassenmodell
Status	Aufzählungstyp	Status einer Reparaturanweisung (bereit zur Bearbeitung, momentan in Bearbeitung, Bearbeitung beendet)	Systemklassenmodell
...			

Tut alles, damit die TutorInnen sehr leicht und schnell alle Begriffe finden!

Der Entwurf beginnt!

Wie erfüllt unser System die gewünschten Aufgaben?

Entwurfsmodelle in der UML



MPGI 3 WS 2008/9

37

Methodisches Vorgehen im Entwurf

1. Festlegen der Systemarchitektur

Kompositions- oder Paketdiagramme

2. Klassentwurf

Entwurfs-Klassenmodell

3. Spezifikation von Schnittstellen

Kommunikations- und Sequenzdiagramme

4. Detailentwurf

Implementierungs-Klassenmodell,
Klassenschnittstellenmodell

MPGI 3 WS 2008/9

38

Schritt 1: Festlegen der Systemarchitektur

- Identifiziere Komponenten (Subsysteme)
- Überprüfe alle Analyse-Klassen, ob sie Subsysteme repräsentieren
- Beschreibe die Architektur mit Kompositions- oder Paketdiagrammen
- Verwende Architektur-Muster, z.B.
 - Client/Server (technisch)
 - 3-Schichten-Architektur (sehr verbreitet)
 - Boundary/Control/Entity

MPGI 3 WS 2008/9

39

Schritt 2: Klassentwurf

- Verfeinere solche Analyse-Klassen, die als Komponenten identifiziert wurden
- Beschreibe diese Analyse-Klassen durch mehrere Entwurfs-Klassen
- Entwickle detaillierte Beschreibungen für die Entwurfs-Klassen
 - alle Attribute
 - alle Operationen/Methoden

MPGI 3 WS 2008/9

40

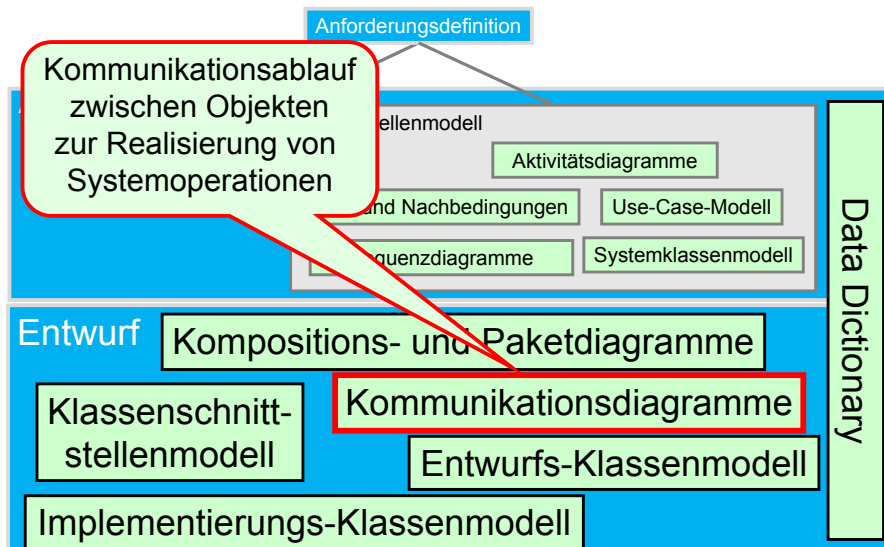
Schritt 3: Spezifikation von Schnittstellen

- Beschreibe Kommunikation zwischen Systemkomponenten und Objekten
- Entwickle hierfür Kommunikations- und Sequenzdiagramme
- Überprüfe Konsistenz zum Entwurfs-Klassenmodell
- Übertrage alle bei der Kommunikation verwendeten Attribute und Methoden

Schritt 4: Detailentwurf

- Berücksichtige erstmalig zu verwendende Programmiersprache (z.B. Mehrfachvererbung)
- Beschreibe alle Klasselemente mit Sichtbarkeiten und detaillierten Datentypen
- Erstelle Implementierungs-Klassenmodell
- Leite daraus Klassenschnittstellenmodell ab
- Beschreibe das Verhalten ausgewählter Methoden durch Zustandsautomaten zur Codegenerierung

Entwurfsmodelle in der UML



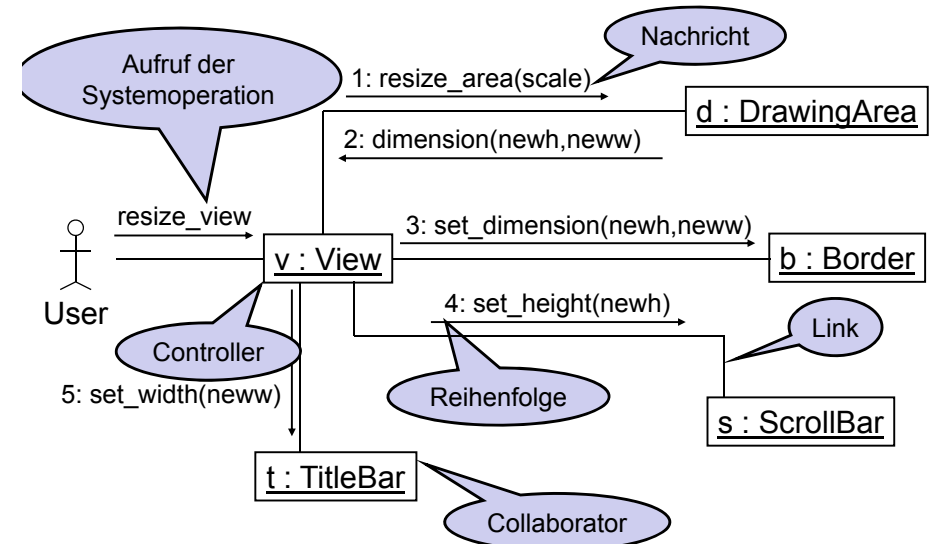
Entwurf mit Kommunikationsdiagrammen

- Modellierung auf Objektebene, nicht auf Klassenebene
- Zu **jeder** Systemoperation **ein** Kommunikationsdiagramm
 - Beschreibe **Kommunikationsstruktur** von allen beteiligten Objekten und Links
 - Untersuche **Ablauf der Kommunikation (synchron)** zwischen den Objekten entlang der Links zur Realisierung der Systemoperation

Bestandteile von Kommunikationsdiagrammen

- **Kontext (collaboration):**
 - Gruppe von Objekten (controller und collaborators)
 - durch Links verbunden
- **Nachricht (message):**
 - gerichteter Pfeil entlang eines Links
 - Name und Parameter der Nachricht
- **Sequenzierung:**
 - Nummern vor den Nachrichten

Beispiel für ein Kommunikationsdiagramm



Aufstellen eines Kommunikationsdiagramms

1. Bestimmen der Eingaben

- aus dem Operationsschema

2. Festlegen des Control-Objekts

- Kommunikation zwischen Boundary und Controller nicht behandeln

3. Behandeln von Fallunterscheidungen

- Identifizieren beteiligter Objekte
- Kommunizieren von Daten zwischen den Objekten
- Verarbeiten/Speichern von Daten in den Objekten
- Erzeugen/Löschen von Objekten
- Versenden von Rückmeldungen an Akteure

4. Prüfen auf Konformität zum Systemklassenmodell

- Aufbau eines Entwurf-Klassenmodells

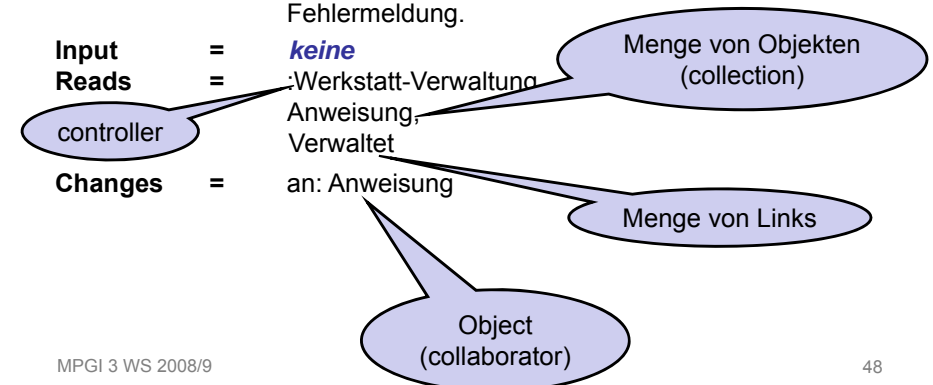
Ausgangspunkt: Analysiere Operationsschema

Beispiel: nächste_anweisung_holen

Operation = nächste_anweisung_holen
Description = Ein Mechaniker drückt sich die nächste vorhandene Anweisung aus. Ist keine aktive Anweisung vorhanden, so bekommt er eine entsprechende Fehlermeldung.

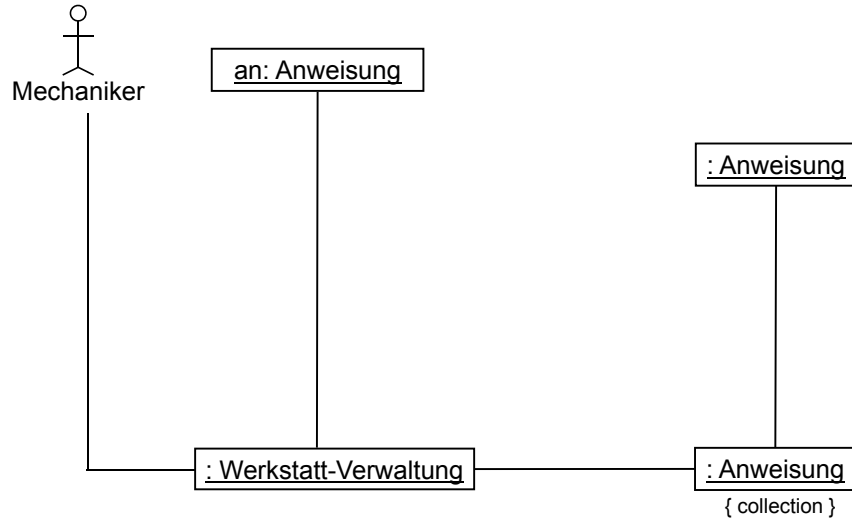
Input = keine
Reads = :Werkstatt-Verwaltung, Anweisung, Verwaltet

Changes = an: Anweisung



Struktur: Welche Objekte sind beteiligt?

Beispiel: `nächste_anweisung_holen`



Kommunikation: Dynamischer Nachrichtenfluss

- beginnt durch Aufruf der Systemoperation
- empfangendes Objekt ist **Control-Objekt**
- Nachrichten können **nur vom jeweils aktiven Objekt** ausgehen
- Nachrichten können **nur entlang von Links** gesendet werden

Notation: Objekterzeugung und -vernichtung

Nachrichten (Operationen) im Nachrichtenfluss

- **Objekterzeugung**
`create` - mit Parametern
- **Objektvernichtung**
`destroy`

Notation: Bedingter Nachrichtenfluss

`nr [Bedingung]: var := Name (Parameter)`

- Nachrichten können **bedingt** gesendet werden
 - Bedingung muss *lokal* überprüfbar sein
 - keine if-then-else *Blöcke*,
d.h. Bedingungen müssen u.U. mehrfach überprüft werden.
- Nachrichten können **Ergebnisse speichern**
 - in lokalen Variablen Speichern
- **bis auf Name und Nummerierung sind alle Bestandteile optional**

zurück zum Ausgangspunkt: **Operationsschema**

Beispiel: **nächste_anweisung_holen**

Sends = Mechaniker : { anweisung_gedruckt,
keine_anweisung_vorhanden}

Pre = implicit

Post = let

```

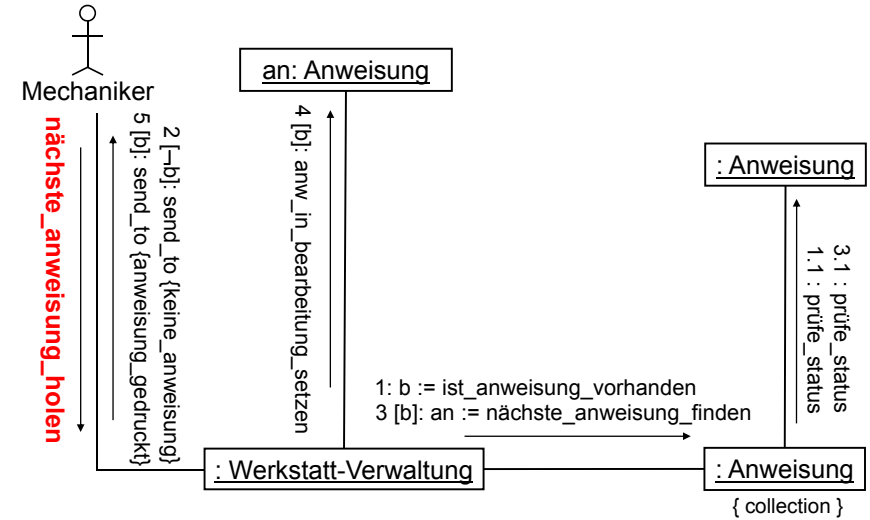
    bereite_anw == {a:Anweisung |
                    a.status = bereit_zur_bearbeitung}

    •
    (bereite_anw = ∅ ⇒
      is_sent {keine_anweisung_vorhanden} ∧
      no_effect ) ∧

    (bereite_anw ≠ ∅ ⇒
      an ∈ bereite_anw ∧
      an.status' = in_bearbeitung ∧
      no_effect ∧
      is_sent {anweisung_gedruckt})
  
```

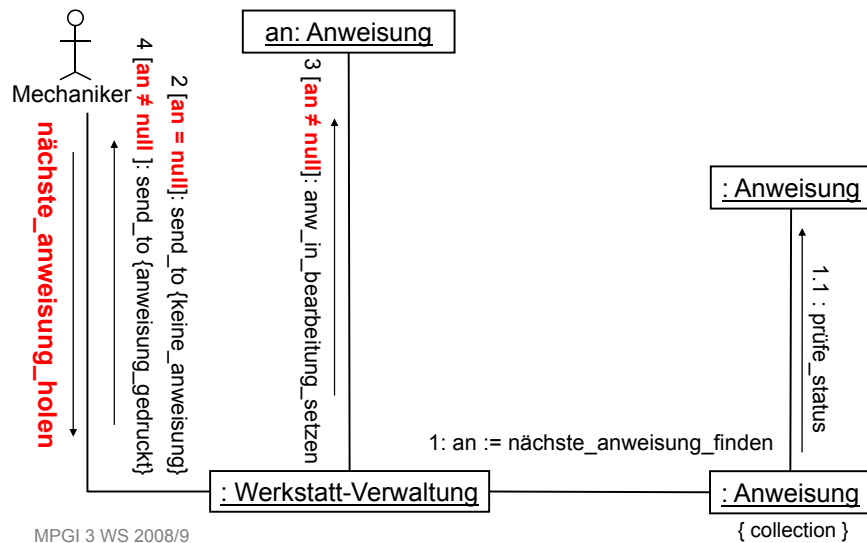
Umsetzung: **Dynamischer Nachrichtenfluss**

Beispiel: **nächste_anweisung_holen**



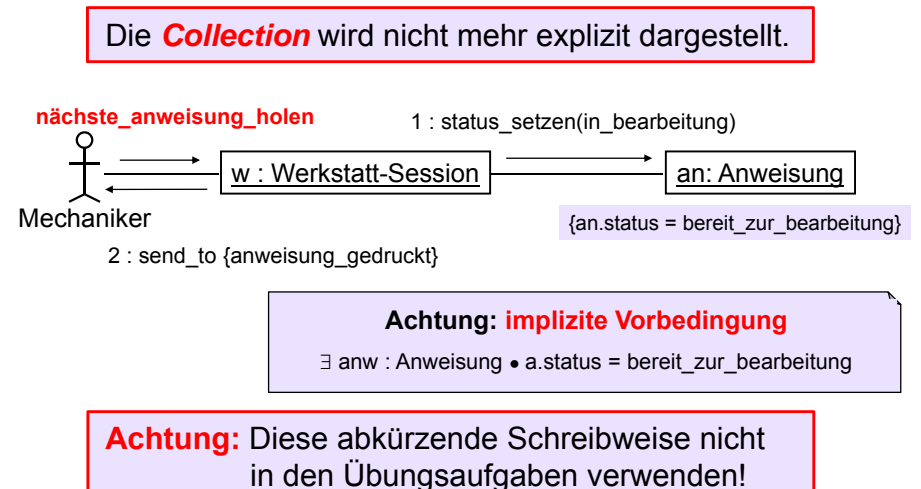
Alternative: **Suchen und Finden Zusammenfassen**

Beispiel: **nächste_anweisung_holen**



Alternative: **Abstrakte Darstellung**

Beispiel: **nächste_anweisung_holen**



Zurück zum Beispiel vom Beginn der Vorlesung

material_eintragen

Ausgangspunkt: Operationsschema

Beispiel: **material_eintragen**

Reads = w : Werkstatt-Verwaltung,
Lager-Posten, Verwaltet, Kennt,
Material-Art, Hat, Anweisung,
an : Anweisung **with**
an.id = anweisung_id \wedge (w,an) \in Verwaltet
mat : Material-Art **with**
mat.id = material_id \wedge (w,mat) \in Kennt

Changes = mp : Material-Posten **type**, Material-Posten
Braucht,
Besitzt

Ausgangspunkt: Operationsschema

Beispiel: **material_eintragen**

Pre = **implicit**

Post = **let**

verfügbar == { l : Lager-Posten | (l,m) \in Hat };

benötigt == { m : Material-Posten | (m,mat) \in Besitzt };

anzahl_{LP} == \sum_{LP} verfügbar;

anzahl_{MP} == \sum_{MP} benötigt

• (anzahl_{LP} < anzahl_{MP} + anzahl \Rightarrow
(**is_sent** { material_nicht_verfügbar } \wedge
is_sent { bestellung_erforderlich })) \wedge

mp **new** \wedge mp.anz' = anzahl \wedge

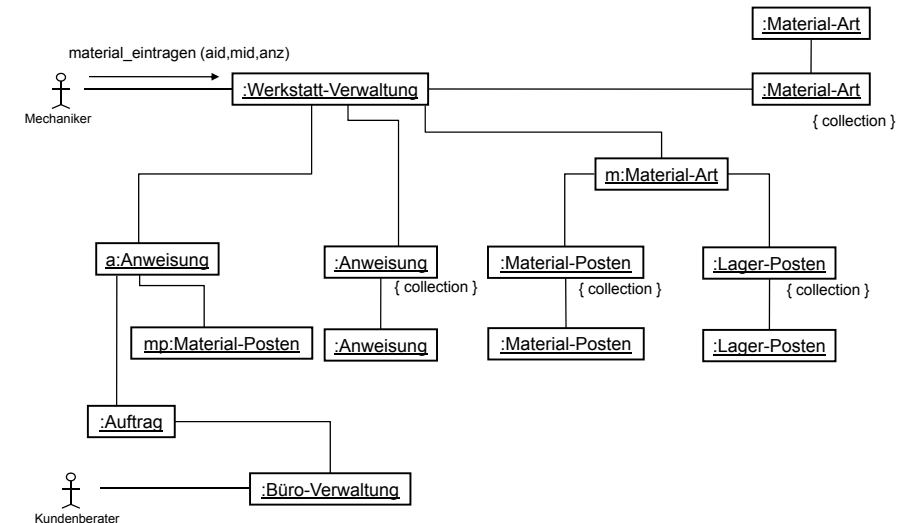
is_sent { material_vermerkt } \wedge

Braucht' = Braucht \cup { (an, mp) } \wedge

Besitzt' = Besitzt \cup { (mp, mat) }

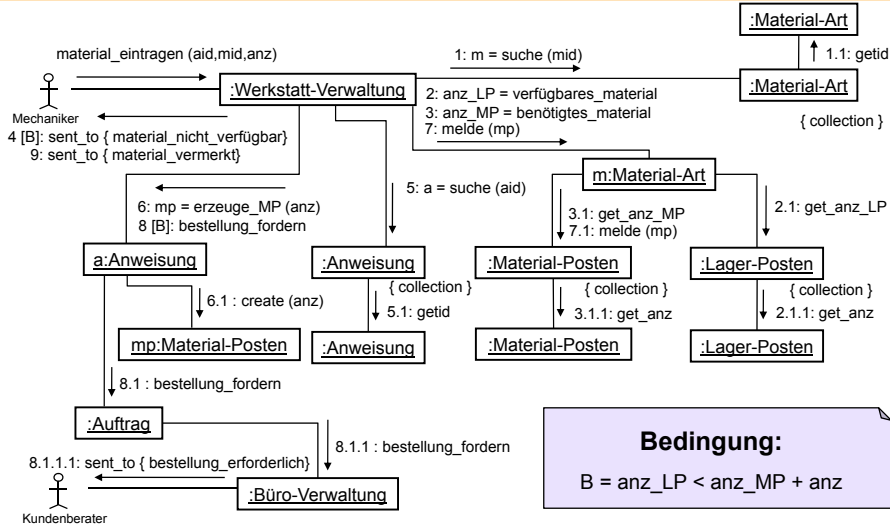
Umsetzung: Struktureller Aufbau

Beispiel: **material_eintragen**



Umsetzung: Dynamischer Ablauf

Beispiel: material_eintragen



Bedingung:
 $B = \text{anz_LP} < \text{anz_MP} + \text{anz}$

Was haben wir bis jetzt erreicht?

