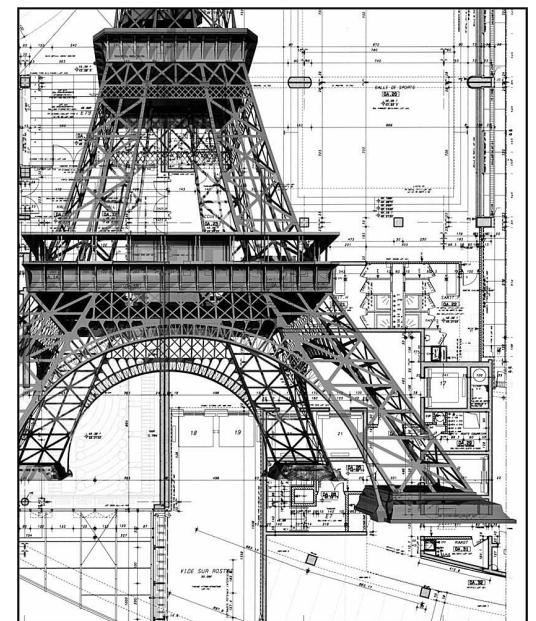


# Architektur von Informationssystemen

Hochschule für angewandte Wissenschaften Hamburg  
Fachbereich Informatik

Prof. Dr. Stefan Sarstedt  
(stefan.sarstedt@haw-hamburg.de)

Dokumentation und Visualisierung  
von Architekturen





# Dokumentation von Architekturen

- Wichtig, um
  - Getroffene Entwurfsentscheidungen später noch nachvollziehen zu können (vermeidet unnötige Diskussionen)
  - Schulung neuer Mitarbeiter
  - Als Basis für die technische Systemdokumentation
- Entscheidungen mit Begründungen dokumentieren!
- (UML-, XY-)Modelle mit verschiedenen Standpunkten einbeziehen
- Nicht zu grobgranular → wenig Aussagekraft
- Nicht zu feingranular → Implementierungsspezifika unnötig; häufige Überarbeitungen sind die Folge



# Dokumentation von Architekturen – Mögliche Struktur

<b>1</b>	<b>EINLEITUNG</b> .....
1.1	Ziele .....
1.2	Umsetzung der fachlichen Architektur .....

<b>2</b>	<b>GESAMTSYSTEM</b> .....
2.1	Architekturüberblick .....
2.2	Konfigurationsparameter .....
2.3	Benutzungsschnittstelle .....

[...]



# Dokumentation von Architekturen – Mögliche Struktur

## 3 SUBSYSTEME UND KOMPONENTEN.....

### 3.1 [REDACTED] .....

3.1.1 Aufgabe und Verantwortung des Subsystems.....

3.1.2 Entwurfsentscheidungen.....

3.1.3 Komponente [REDACTED] .....

    3.1.3.1 Aufgabe und Verantwortung der Komponente.....

    3.1.3.2 Entwurfsentscheidungen.....

    3.1.3.3 Außensicht.....

    3.1.3.4 Innensicht.....

    3.1.3.5 Persistenz.....

    3.1.3.6 Automatisierte Abläufe.....

    3.1.3.7 Konfigurationsparameter.....

    3.1.3.8 Benutzungsschnittstelle.....

    3.1.3.9 Datenauswertungen.....

    3.1.3.10 Prozesssicht.....

    3.1.3.11 Ressourcenverbrauch.....

    3.1.3.12 Anzupassende Komponenten.....

3.1.4 Komponente [REDACTED] .....

    3.1.4.1 Aufgabe und Verantwortung der Komponente.....

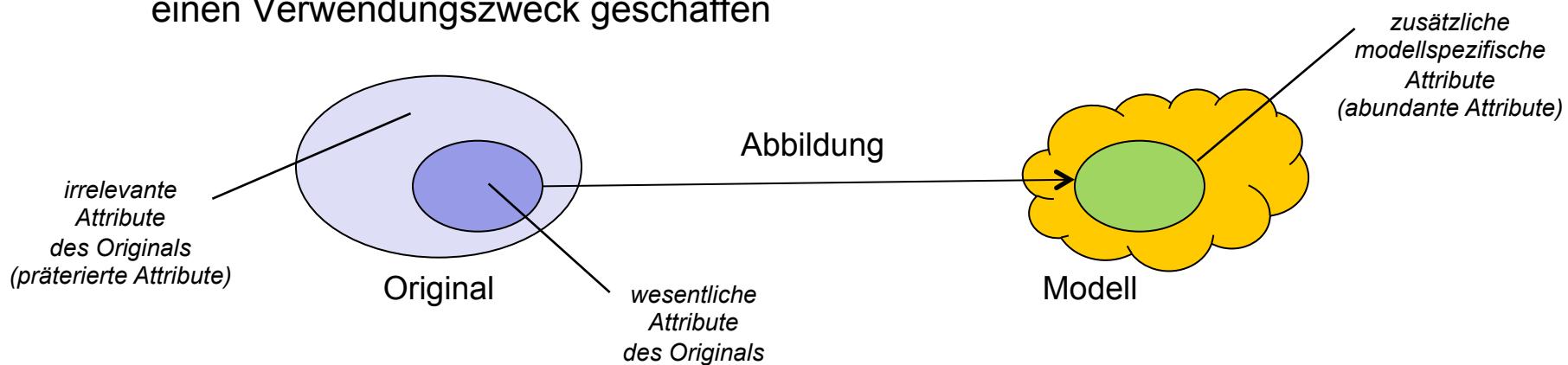


## Definition: Modell

- Abbildungen der realen Welt auf abstrakte Gebilde
  - Reduktion komplexer Sachverhalte auf das Wesentliche (Ausblenden unwichtiger Details)
  - **Abbilder oder Vorbilder**
- Ermöglichen einen Informationsgewinn
- Wichtig im Software-Engineering, da wir es mit immateriellen Produkten zu tun haben
  
- **Vorteile**
  - billiger als Prototypen
  - i.d.R. leicht verständlich
  - Dienen der Kommunikation und als Diskussionsgrundlage

# Eigenschaften von Modellen

- Charakteristische Merkmale von Modellen (nach Stachowiak, 1973)
  - Abbildungsmerkmal**  
Zu jedem Modell gibt es ein Original
  - Verkürzungsmerkmal**  
Nicht alle Attribute des Originals werden erfasst, neue (modellspezifische) Attribute kommen hinzu
  - Pragmatisches Merkmal**  
Modelle können unter bestimmten Bedingungen und bezüglich geeigneter Fragestellungen das Original ersetzen, d. h. Jedes Modell wird im Hinblick auf einen Verwendungszweck geschaffen





## Wozu modellieren?

*„The purpose of models is not to fit the data but to sharpen the questions“ (Karlin, 1983)*

*“It was originally Charles Box (and later Deming) who said:  
‘All Models Are Wrong, Some Are Useful.’  
We should learn to live with that reality.“*



# Standpunkte und Sichten der Architekturbeschreibung

- Alle Beschreibungsansätze erlauben Darstellung struktureller Dekomposition
- Statische Strukturen sind Grundlage für das Verständnis eines Systems, in vielen Fällen ist dies jedoch nicht ausreichend
- Für Systemanalysen sind bspw. auch die Darstellung von Verhaltensaspekten nötig
- ISO/IEC DIS 25961 definiert konzeptionellen Rahmen zur Beschreibung von Architekturen (Basis für das Folgende)



## Definitionen: Architekturbeschreibung, Standpunkt und Sicht

*Eine Architekturbeschreibung ist eine Menge von Modellen (also z. B. textuelle Spezifikationen oder grafische Diagramme wie die UML-Diagramme), die die Software-Architektur dokumentieren.*

*Ein Standpunkt (viewpoint) verkörpert miteinander verwandte Anliegen bzw. Aspekte (concerns), die durch die Interessen der Projektbeteiligten (stakeholder) bestimmt werden. Ein Standpunkt legt auch Verfahren zur Modellierung und zugehörige Notation fest. [...]*

*Eine Sicht (view) beschreibt ein (konkretes) System von einem bestimmten (abstrakten) Standpunkt aus. Die Sichten dienen dabei der Strukturierung der Architekturbeschreibung [...] Zwischen Sichten und Modellen besteht i. d. R. eine m-zu-n-Beziehung [...]*

[Reussner2009]



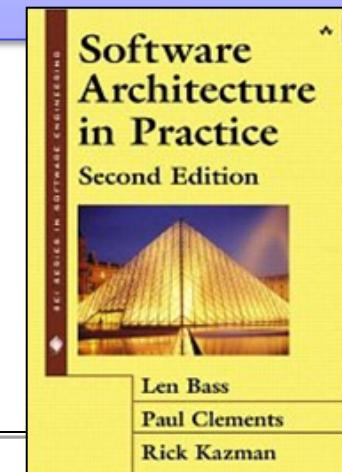
# Vergleich verschiedener Standpunktmengen

## *Die Standpunktmenge von Clements et al.*

**Modul-Standpunkt (module view type):** Sichten dieses Standpunkts beschreiben die strukturelle Zerlegung des Systems in Einheiten der Implementierung.

**Komponenten-und-Konnektoren-Standpunkt (components and connectors view type):** Sichten dieses Standpunkts beschreiben Komponenten und deren Interaktionen zur Laufzeit.

**Zuordnungs-Standpunkt (allocation view type):** Sichten dieses Standpunkts beschreiben den Zusammenhang zwischen der Software und ihrer Entwicklung- und Ausführungsumgebung





# Vergleich verschiedener Standpunktmengen

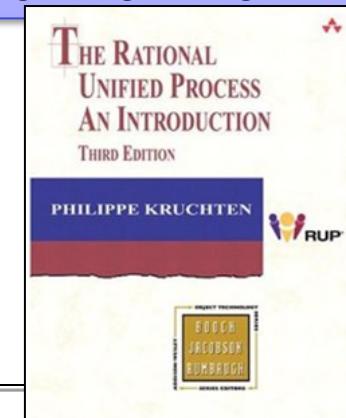
## *Die „4+1“-Standpunktmenge von Kruchten*

**Logischer Standpunkt:** Sichten dieses Standpunkts beschreiben die vorkommenden Objekte bzw. Komponenten.

**Prozess-Standpunkt:** Sichten dieses Standpunkts beschreiben den Programmablauf durch Prozesse, Threads und Tasks, und deren Kommunikation.

**Physischer Standpunkt:** Sichten dieses Standpunkts beschreiben das Deployment von Software-Komponenten auf Hardware-Einheiten.

**Entwicklungs-Standpunkt:** Sichten dieses Standpunkts beschreiben statische Organisation des Software-Systems in der verwendeten Entwicklungsumgebung.





# Vergleich verschiedener Standpunktmengen

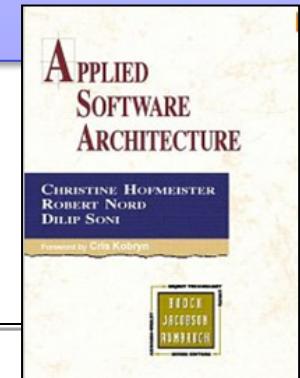
## *Die Standpunktmenge von Hofmeister, Nord und Soni*

**Konzeptioneller Standpunkt:** Dieser wird zur Beschreibung der funktionalen Aspekte des Systems und dessen Modifizierbarkeit verwendet.

**Modul-Standpunkt:** Dieser wird zur Beschreibung der Aufteilung der Software in Module und ihre Organisation in Schichten verwendet.

**Ausführungs-Standpunkt:** Dieser wird zur Beschreibung der Abbildung der Module auf ausführbare Einheiten, der Kommunikation zwischen ihnen und ihrer Zuordnung auf physikalische Einheiten verwendet.

**Quelltext-Standpunkt:** Dieser wird zur Beschreibung der Zuordnung von Modulen und Schnittstellen auf Quelldateien und von ausführbaren Einheiten auf ausführbare Dateien verwendet.





## Vergleich verschiedener Standpunktmengen – Fazit

**Statischer Standpunkt:** Sichten aus diesem Standpunkt beschreiben Zerlegungen des betrachteten Systems in Elemente und deren Abhängigkeiten, z. B. in Komponenten und Konnektoren. Ein System kann unterschiedliche Zerlegungen aufweisen.

**Dynamischer Standpunkt:** Sichten aus diesem Standpunkt beschreiben das Systemverhalten zu seiner Laufzeit. [...]

**Verteilungs-Standpunkt:** Sichten aus diesem Standpunkt beschreiben die Abbildung bestimmter Elemente aus der strukturellen Sicht einerseits auf Infrastruktur- bzw. Hardware-Einheiten (Prozessoren, Netzwerkressourcen, Drucker, etc.) – das Deployment – und andererseits auf organisatorische Einheiten (Entwickler, Teams oder externe Hersteller). [...]

[Reussner2009]





## Verteilungs-Standpunkt

- Eine Software kann ihren Zweck nicht erfüllen, bis sie installiert (“deployed”) wurde
  - Ausführbare Module werden der Hardware, auf der sie laufen sollen, zugeordnet
- Diese Perspektive einer Architektur kann **kritisch** in Bezug auf die Einschätzung sein, ob seine Anforderungen erfüllt werden können! 
- Mögliche Dimensionen der Einschätzung
  - zur Verfügung stehender Speicher
  - Energieverbrauch
  - benötigte Netzwerkanbindung

Client Computer

Webbrowser

HTTPS

Transportplanungssystem Application Server

Windows Server 2008 R2  
IP: 141.2.10.201

Java RMI

JDBC

Failover Service

SMTP

Legende



= ausführbare Softwareeinheit

— = Kommunikation

Windows Server 2008 R2  
IP: 141.2.11.168



JMS

Apache ActiveMQ 5.3 Server

Queue

SAP/R3

XML

## Verteilungs-Standpunkt – Beispiel

*Operator*



# Architekturbeschreibung mit der UML

- **Unified Modeling Language** bekannt aus SE1
- Populär zur Beschreibung von Architektsichten und Entwürfen

## Die UML ist

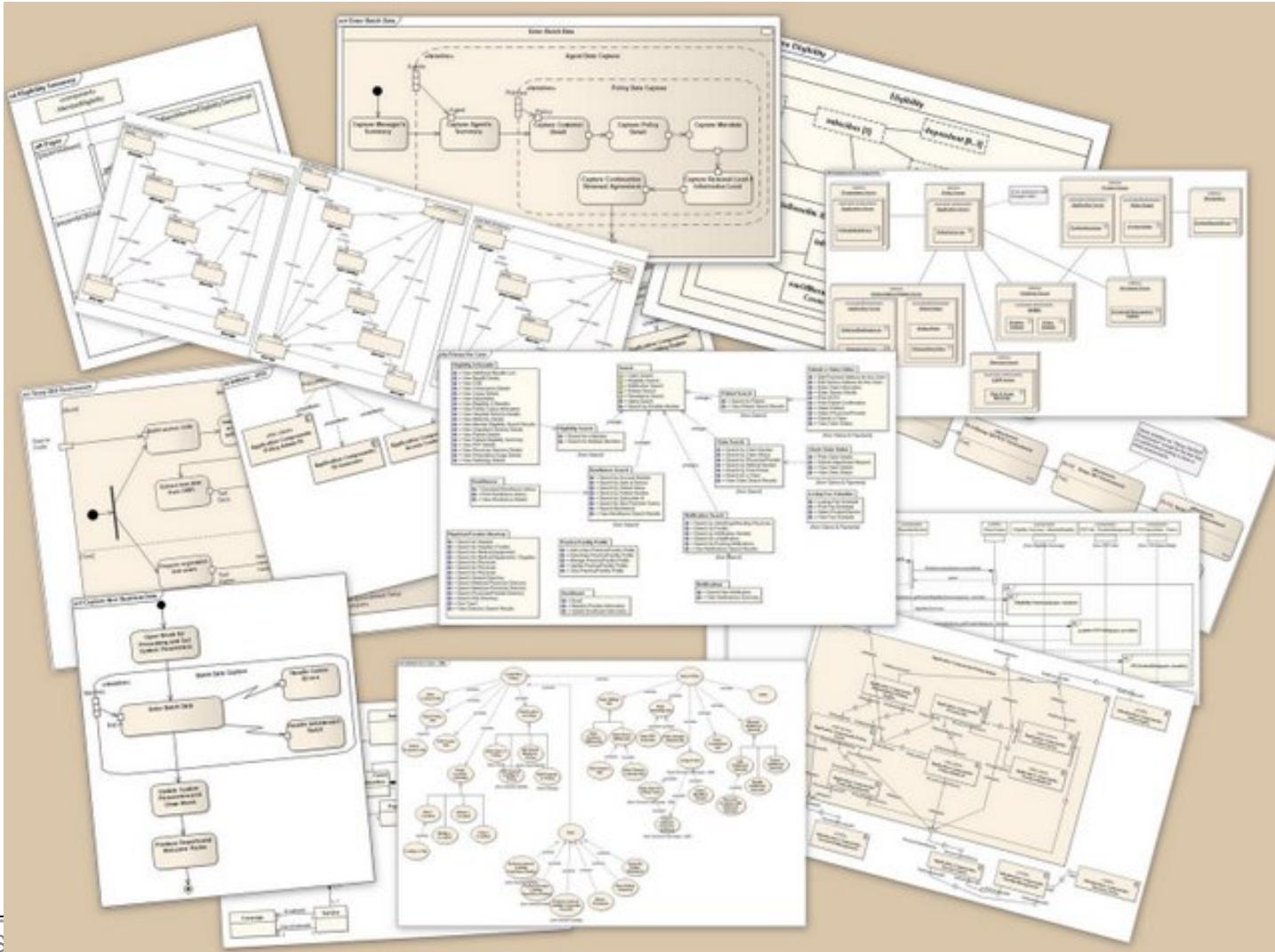
- eine Modellierungssprache
- eine standardisierte Notation
- ein Kommunikationsmittel
- technisch und fachlich neutral
- flexibel und erweiterbar

## Die UML ist nicht

- ein Prozessmodell oder eine Methode
- eine Programmiersprache
- ein Ersatz für geschriebenen Text
- formal beschrieben



# Architekturbeschreibung mit der UML



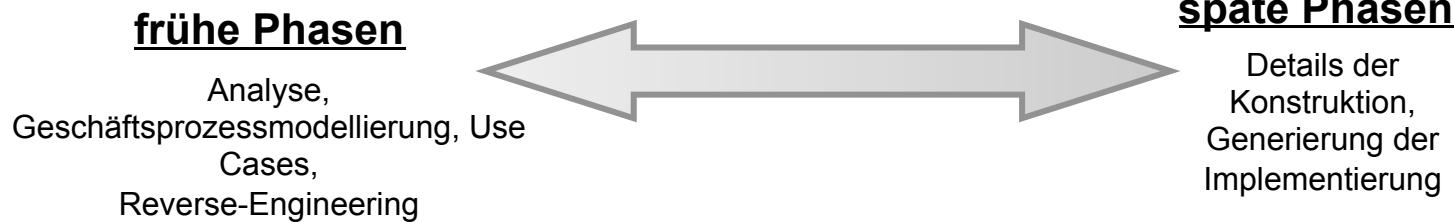
Wikipedia

# UML – Einsatzszenarien

Nach Formalitätsgrad:

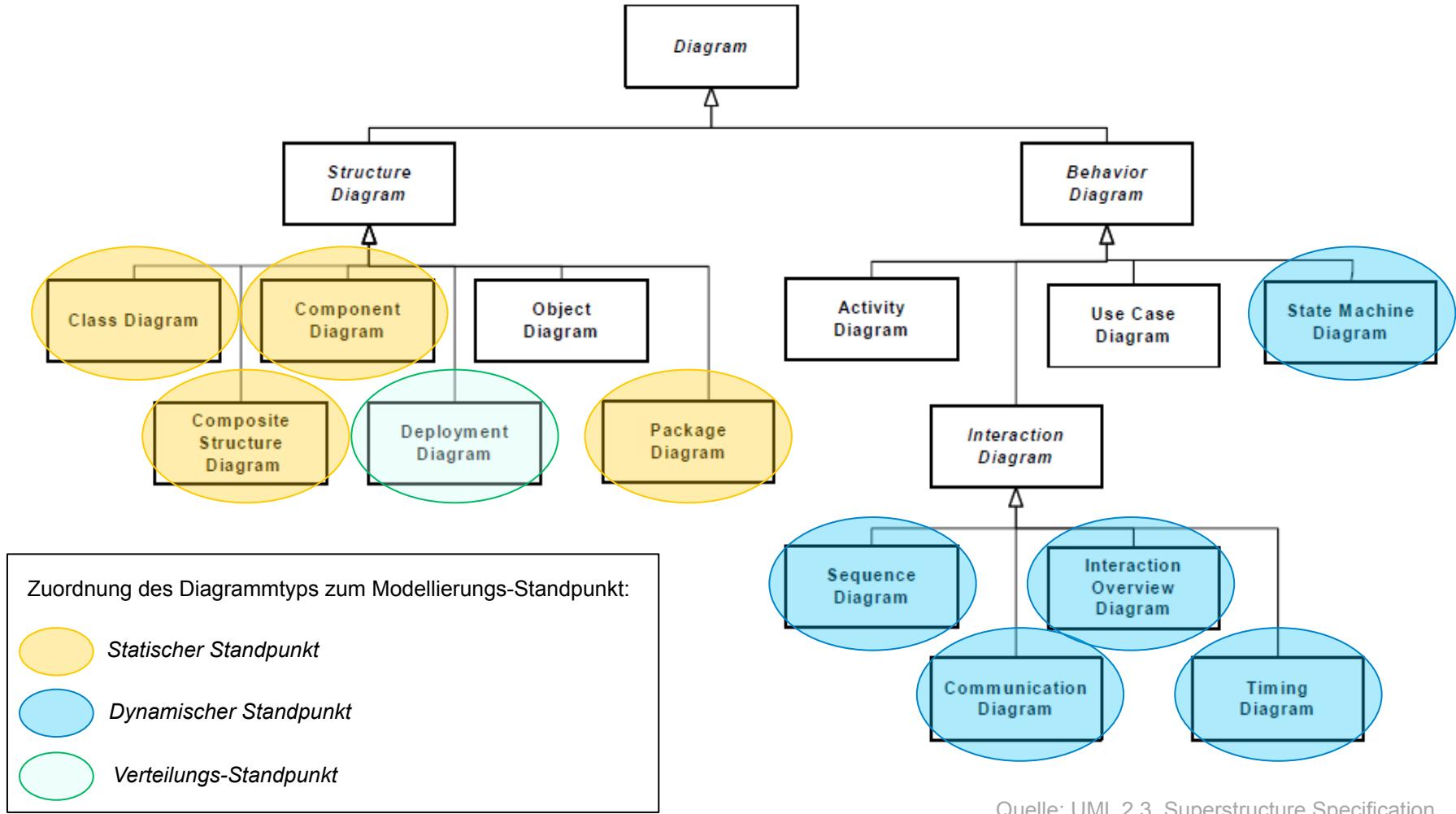


Bezogen auf die Projektphasen:





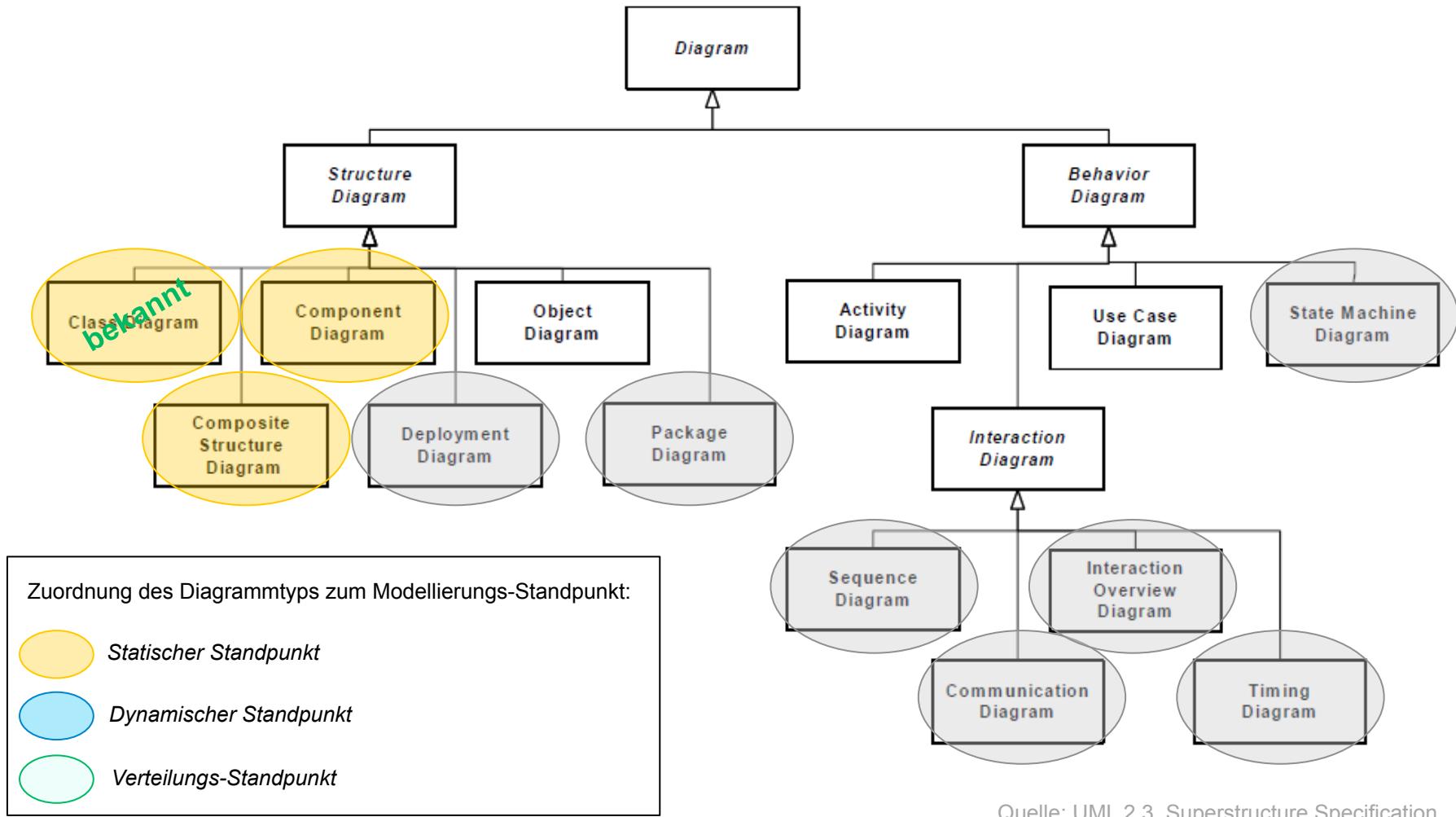
# Architekturbeschreibungs-Standpunkte und die UML



Quelle: UML 2.3. Superstructure Specification



# Architekturbeschreibungs-Standpunkte und die UML





## Architekturen modellieren mit der UML – Grundbegriff „Classifier“ / „Classifier“

[...] Ein **Classifier** innerhalb von UML ist [...] ein abstraktes gedankliches Konstrukt, als Modellierer nie direkt einen Classifier in eines der Diagramme der UML einfügen. [...]

Innerhalb der UML-Spezifikation nimmt der Classifier jedoch eine **zentrale Rolle** ein, weil er die gemeinsamen Eigenschaften von Modellelementen der UML umfasst, die auch Anwender der UML häufig einsetzen. So sind zum Beispiel die Modellelemente **Klasse**, **Schnittstelle**, **Komponente**, **Verhalten**, **Aktivität**, **Interaktion** oder **Zustandsautomat** Spezialisierungen des Classifiers - ein **Classifier ist also eine Generalisierung dieser Modellelemente**. [...]

[http://de.wikipedia.org/w/index.php?title=Classifier\\_\(UML\)&oldid=43913066](http://de.wikipedia.org/w/index.php?title=Classifier_(UML)&oldid=43913066)



# Architekturen modellieren mit der UML

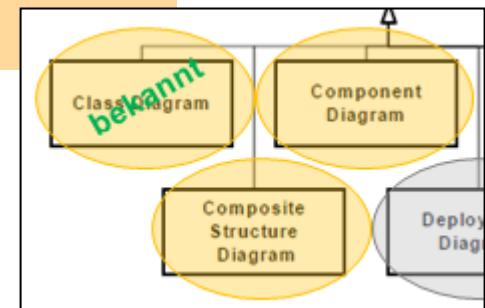
Das **Komponentendiagramm** [...] zeigt eine bestimmte Sicht auf die **Struktur** des modellierten Systems. Die Darstellung umfasst dabei typischerweise **Komponenten** mit deren **Schnittstellen** bzw. **Ports**. Es zeigt auch, wie Komponenten über **Abhängigkeitsbeziehungen** und Konnektoren miteinander verbunden sind.

Um das **Innere** einer Komponente darzustellen, zeigt ein Komponentendiagramm oft Notationselemente, die sonst vor allem in Klassen- oder Kompositionsstrukturdiagrammen angezeigt werden, zum Beispiel Klassen oder Parts.

<http://de.wikipedia.org/w/index.php?title=Kompositionssstrukturdiagramm&oldid=77541163>

Das **Kompositionssstrukturdiagramm** zeigt einheitlich **das Innere** eines Klassifizierers und dessen **Wechselwirkung** mit seiner Umgebung.

<http://de.wikipedia.org/w/index.php?title=Komponentendiagramm&oldid=77541167>





## Architekturen modellieren mit der UML

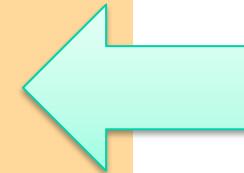
Das **Komponentendiagramm** [...] zeigt eine bestimmte Sicht auf die **Struktur** des modellierten Systems. Die Darstellung umfasst dabei typischerweise **Komponenten** mit deren **Schnittstellen** bzw. **Ports**. Es zeigt auch, wie Komponenten über **Abhängigkeitsbeziehungen** und Konnektoren miteinander verbunden sind.

Um das **Innere** einer Komponente darzustellen, zeigt ein Komponentendiagramm oft Notationselemente, die sonst vor allem in Klassen- oder Kompositionsstrukturdiagrammen angezeigt werden, zum Beispiel **Klassen** oder **Parts**.

<http://de.wikipedia.org/w/index.php?title=Kompositionssstrukturdiagramm&oldid=77541163>

Das Kompositionssstrukturdiagramm zeigt einheitlich das **Innere** eines **Klassifizierers** und dessen **Wechselwirkung** mit seiner Umgebung.

<http://de.wikipedia.org/w/index.php?title=Komponentendiagramm&oldid=77541167>





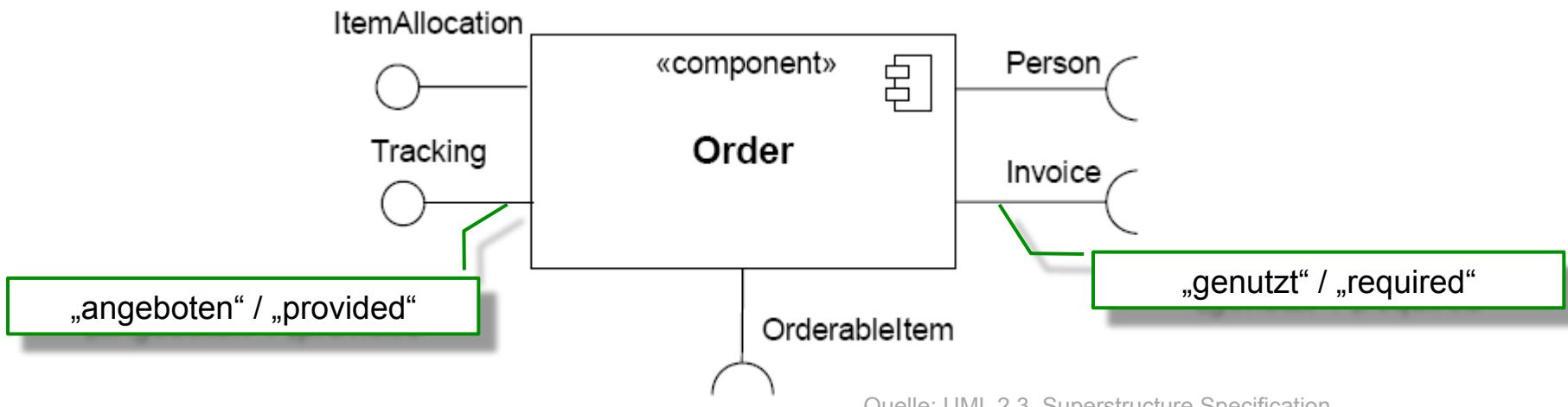
## UML Komponentendiagramme – Basis

- Unterstützt die Modellierung von
  - **logischen/fachlichen** Komponenten
  - **technischen** Komponenten (EJB, .NET, CORBA, ...)
- ...zusammen mit den Artefakten, die diese Komponenten implementieren (z.B. „Auftragsmanagement.jar“) und dem Knoten, auf denen sie installiert werden
- Der Stereotyp „«component»“ ist optional
- Andere Stereotype sind möglich, z. B.
  - „«subsystem»“ für „grobgranularere“ Komponenten
  - „«process»“ für Prozesskomponenten
  - „«service»“ für zustandslose funktionale Komponenten
  - u.a. (**Stereotype sparsam und mit Bedacht einsetzen!**)



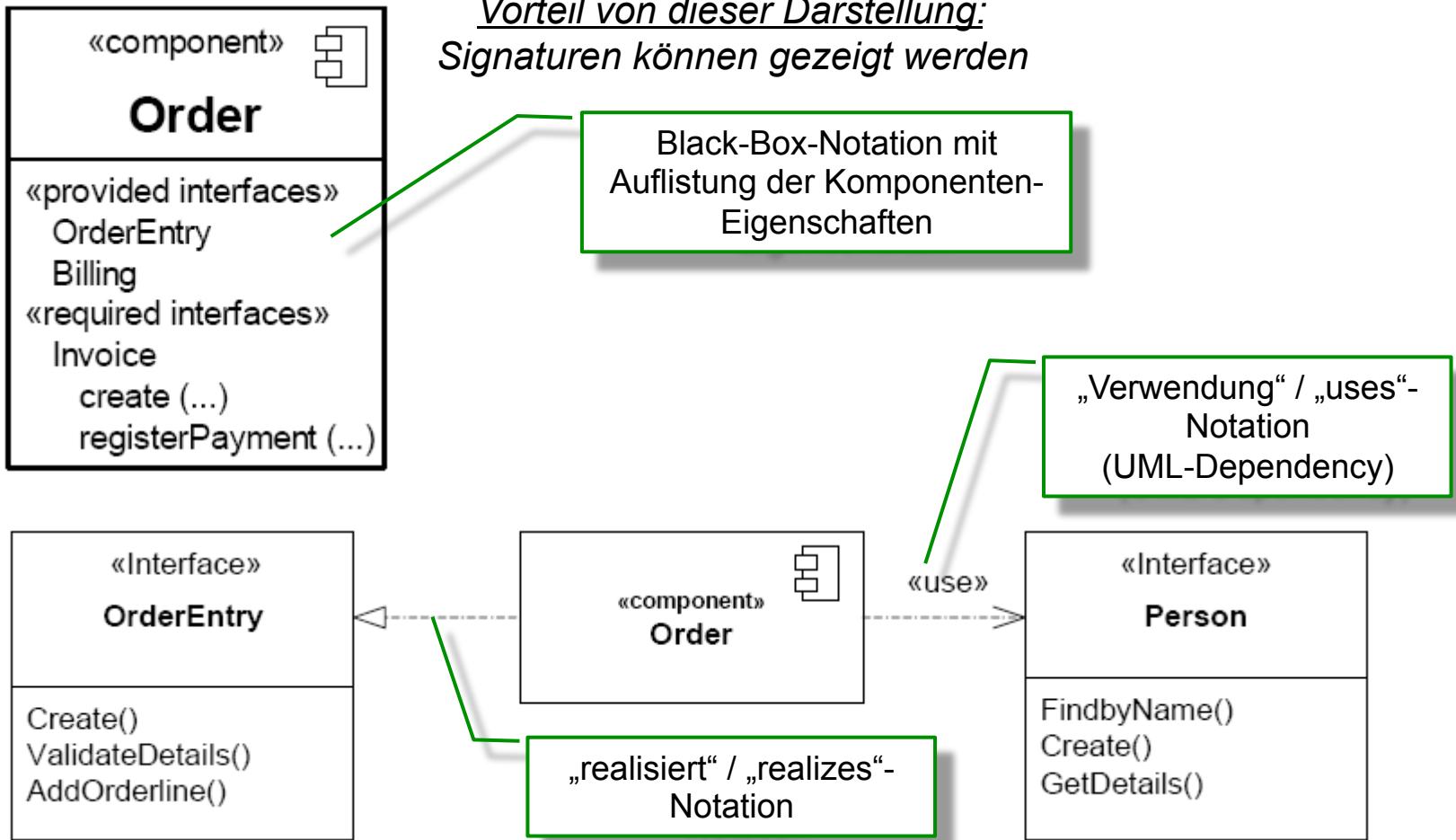
# UML Komponentendiagramme – Schnittstellen

- Eine Komponente hat eine Menge von **angebotenen** („provided“) und **genutzten** („required“) Schnittstellen/Interfaces
  - Angebotene Schnittstellen werden entweder direkt durch die Komponente realisiert, durch eine seiner „realizingClassifier“, oder durch einen Port (s.u.)
  - Genutzte Schnittstellen zeichnen sich durch eine Nutzungs-Beziehung aus; von der Komponente, eines „realizingClassifier“, oder durch einen Port





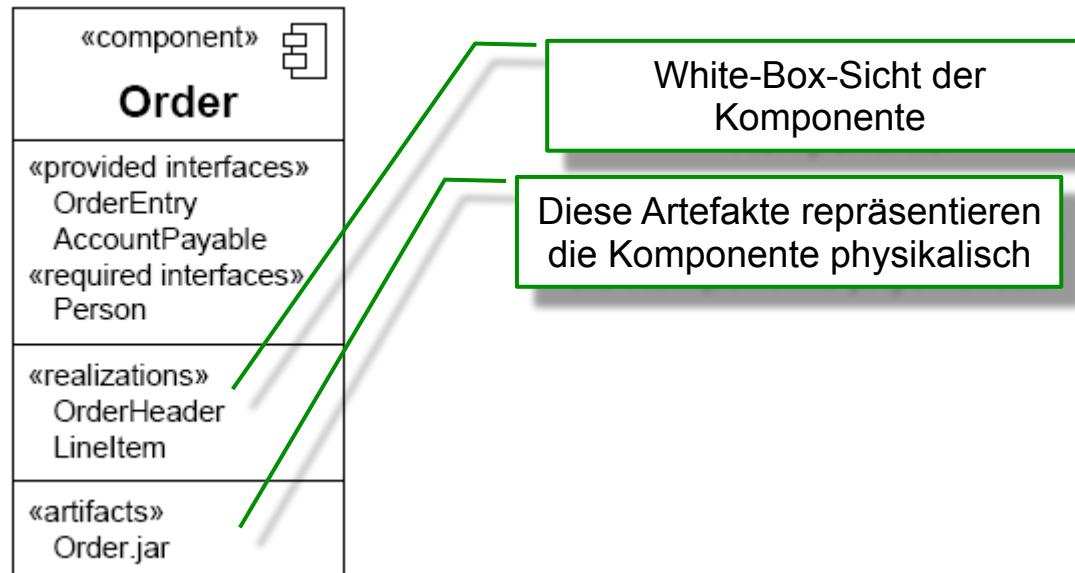
# UML Komponentendiagramme – Schnittstellen



Quelle: UML 2.3. Superstructure Specification

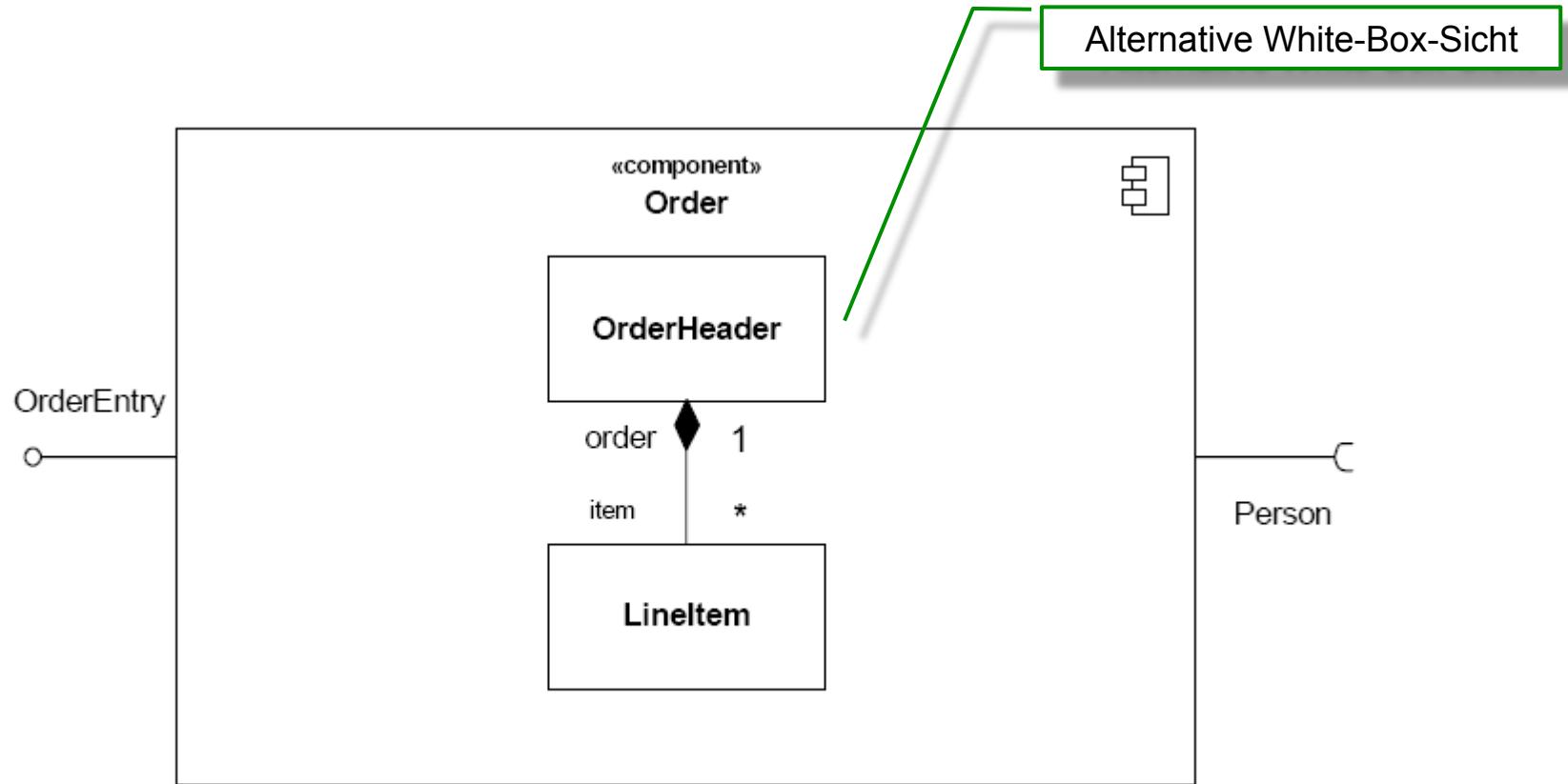


# UML Komponentendiagramme – White-Box-Sicht



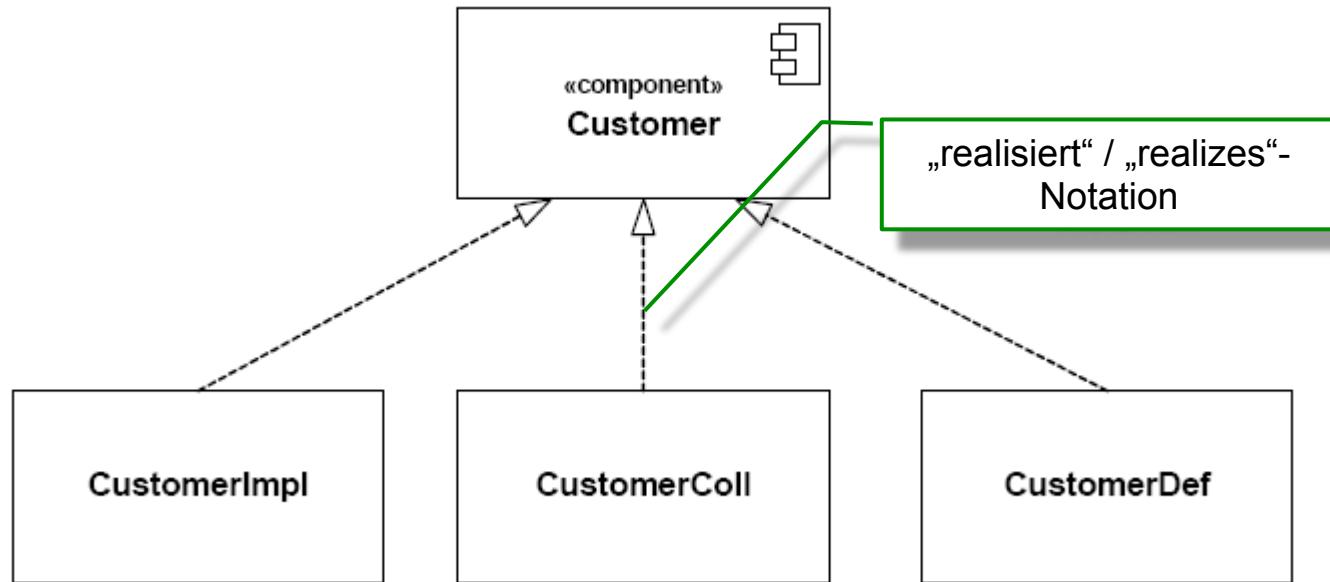
Quelle: UML 2.3. Superstructure Specification

# UML Komponentendiagramme – White-Box-Sicht



Quelle: UML 2.3. Superstructure Specification

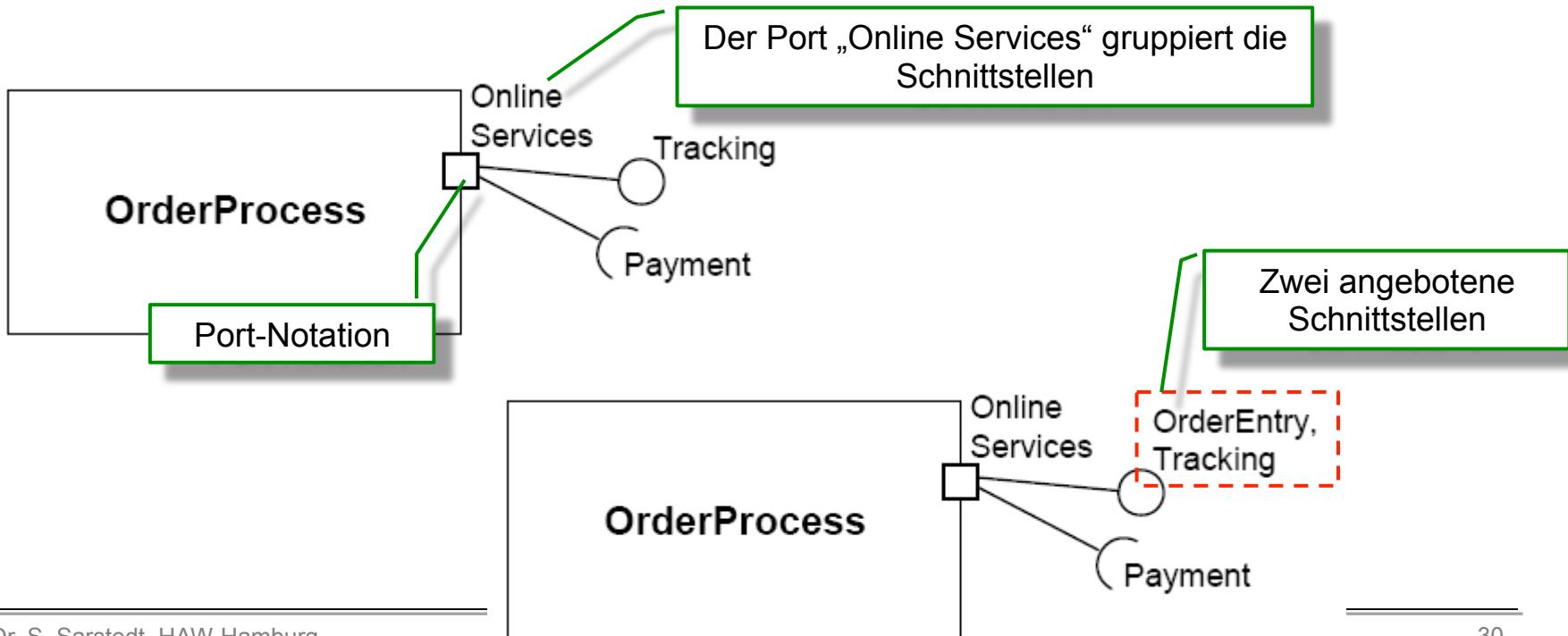
# UML Komponentendiagramme – Realisierungen



Quelle: UML 2.3. Superstructure Specification

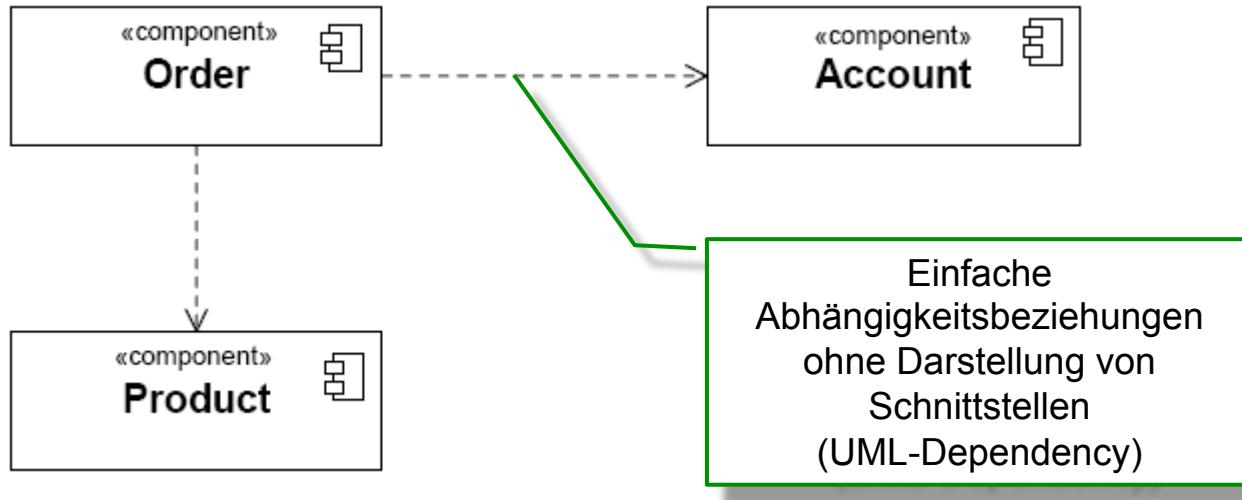
# UML Komponentendiagramme – Ports

- Eine UML-Komponente kann optional eine Menge sog. **Ports** besitzen, die ihre Interaktionspunkte formalisieren
  - wichtig in Kompositionssstrukturdiagrammen (s.u.) um die Verbindung von Schnittstellen zu den internen Bestandteilen zu visualisieren
  - Ports können Schnittstellen außerdem gruppieren





# UML Komponentendiagramme – Abhängigkeiten



- ausreichend zu Beginn der Architekturentwicklungsphase, wenn wir noch keine Schnittstellen haben
- Später zu wenig aussagekräftig!

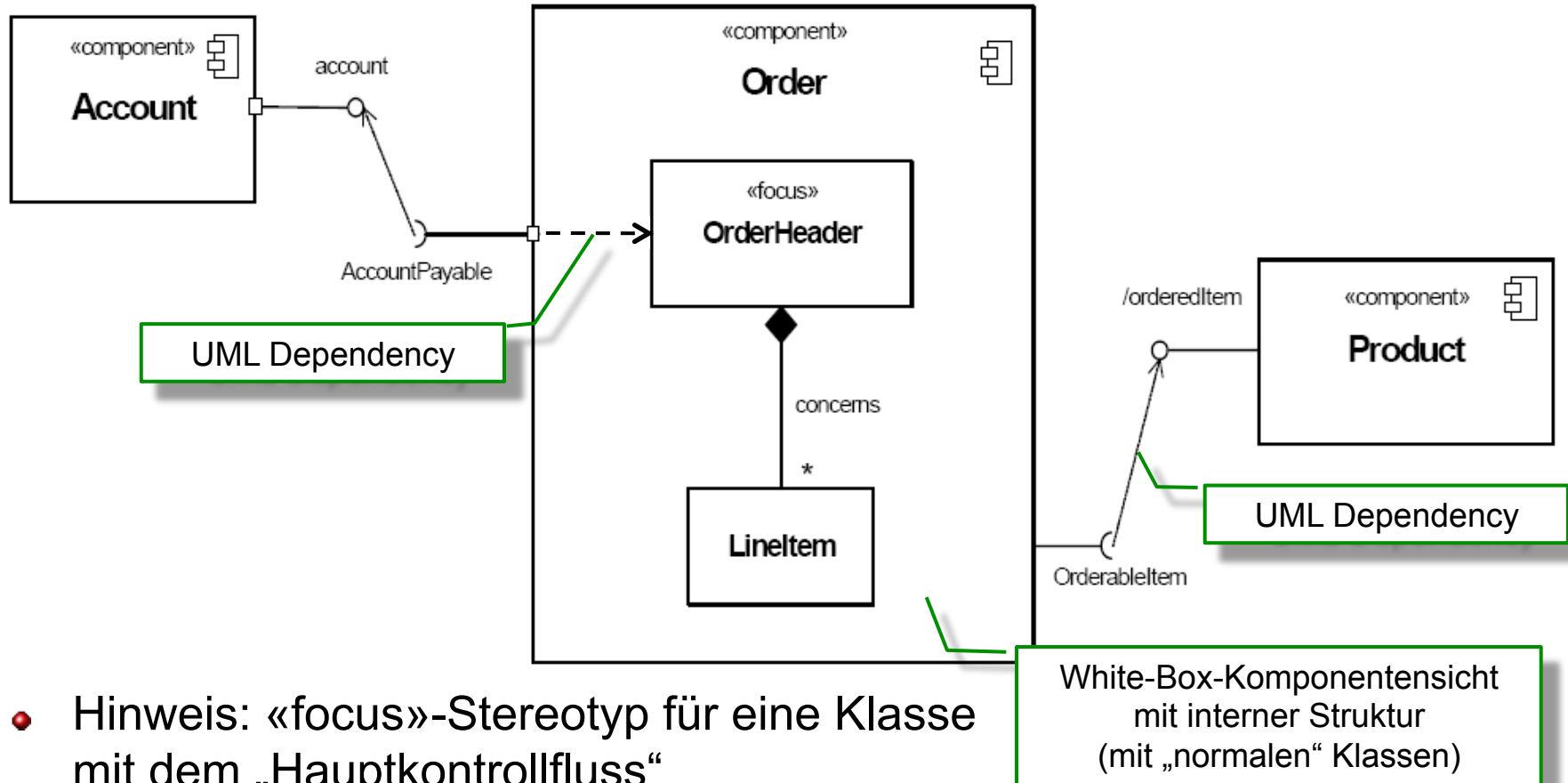


## UML Komponentendiagramme – Abhängigkeiten

- Die **strukturelle Verknüpfung zwischen Komponenten** erfolgt durch Abhängigkeiten zwischen “Simple Ports” (Ports mit einer einzigen Schnittstelle) oder zwischen Schnittstellen-Realisierungen und – Verwendungen (“Lollipops” und “Sockets”)
- Zusätzliche Angaben können nötig sein!  
→ Performanz- oder andere nichtfunktionale Anforderungen



# UML Komponentendiagramme – Abhängigkeiten



- Hinweis: «focus»-Stereotyp für eine Klasse mit dem „Hauptkontrollfluss“  
(lieber weglassen, versteht niemand)



## Architekturen modellieren mit der UML

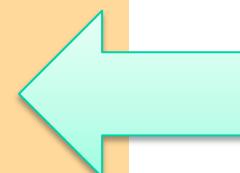
Das Komponentendiagramm [...] zeigt eine bestimmte Sicht auf die **Struktur** des modellierten Systems. Die Darstellung umfasst dabei typischerweise **Komponenten** mit deren **Schnittstellen** bzw. **Ports**. Es zeigt auch, wie Komponenten über **Abhängigkeitsbeziehungen** und Konnektoren miteinander verbunden sind.

Um das **Innere** einer Komponente darzustellen, zeigt ein Komponentendiagramm oft Notationselemente, die sonst vor allem in Klassen- oder Kompositionsstrukturdiagrammen angezeigt werden, zum Beispiel **Klassen** oder **Parts**.

<http://de.wikipedia.org/w/index.php?title=Kompositionsstrukturdiagramm&oldid=77541163>

Das **Kompositionsstrukturdiagramm** zeigt einheitlich **das Innere eines Klassifizierers und dessen Wechselwirkung mit seiner Umgebung**.

<http://de.wikipedia.org/w/index.php?title=Komponentendiagramm&oldid=77541167>





## UML Kompositionsstrukturdiagramm – Parts

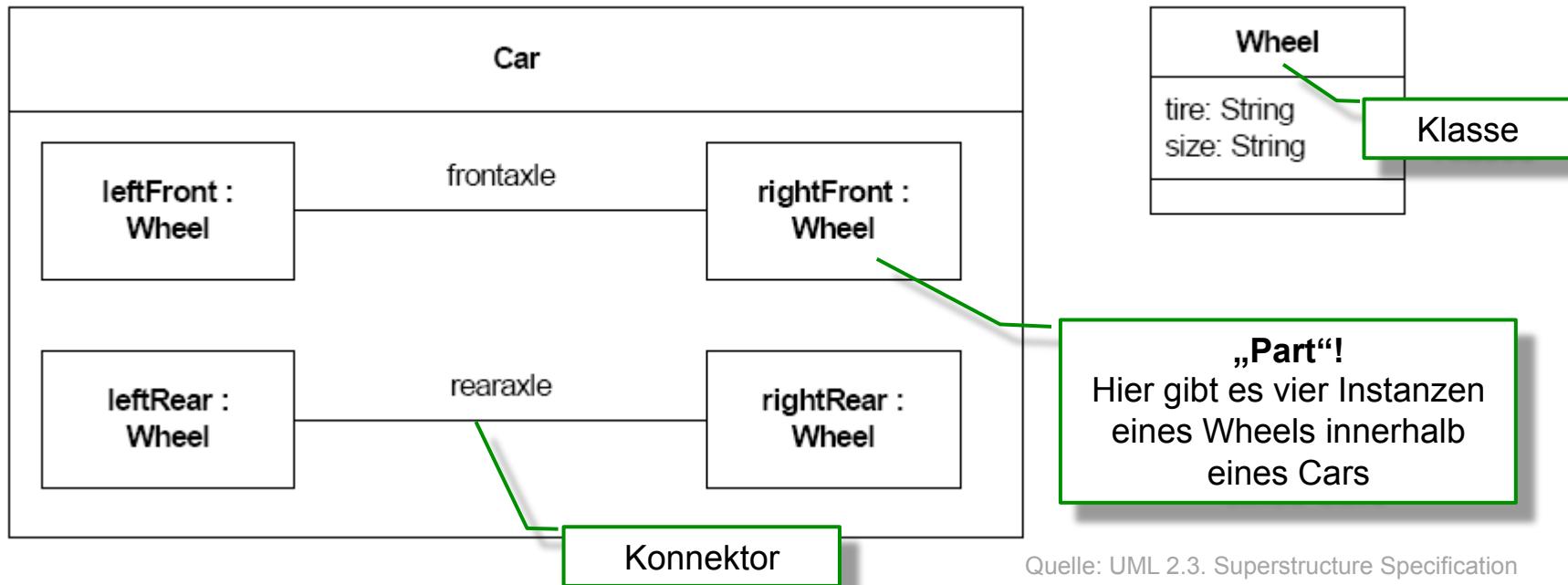
- Falls Details auf Instanz-Ebene für Interna benötigt werden, sind sog. **Parts** einzusetzen
  - z. B. ist eine Klasse der Typ („Classifier“); Parts stellen Instanzen dar

**Parts** sind **Bestandteile** des Ganzen, die durch eine Kompositionsbeziehung zum Ganzen gehören bzw. vom Ganzen komponiert werden. Ein Part wird mithilfe eines Rechtecks dargestellt, der eine eigene Multiplizität besitzen kann.

Sie werden vom Ganzen mit einem großen Rahmen umschlossen. Im Inneren des Rahmens zeigen die Parts ihre Beziehungen mithilfe von **Konnektoren** untereinander auf, nach außen hin werden über **Schnittstellen** (Interfaces) die angebotenen und bereitgestellten Merkmale (Features) dargestellt.

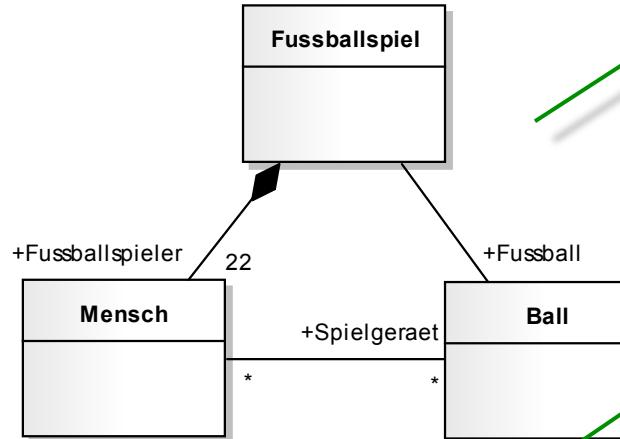
<http://de.wikipedia.org/w/index.php?title=Kompositionsstrukturdiagramm&oldid=77541163>

# UML Kompositionssstrukturdiagramm – Beispiel 1



# UML Kompositionssstrukturdiagramm – Beispiel 2

class FußballspielKlassenDiagramm



Klassendiagramm zeigt nicht Zusammenhang zur Laufzeit

Kompositionssstrukturdiagramm stellt dagegen die Instanzen dar

composite structure FußballspielAusParts



Multiplizität

Ball gehört nicht „eng“ zu Fußballspiel (keine Komposition), daher gestrichelt



# Verbinden von Parts / Ports / Schnittstellen

Ein **Konnektor** spezifiziert eine Verbindung (“link”) zur Kommunikation zwischen zwei oder mehreren Instanzen.

Ein **Delegationskonnektor** (“delegation connector”) ist ein Konnektor, der die externen Schnittstellen einer Komponente (die durch Ports spezifiziert sind) mit der Realisierung dieses Verhaltens verbindet. Es repräsentiert die Weiterleitung von Operationen und Ereignissen: ein Signal, dass an einem Port ankommt wird durch den Delegationskonnektor an ein oder mehrere Parts weitergeleitet.

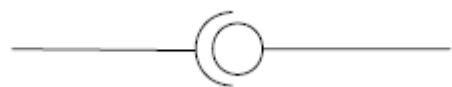
Ein **Kompositionskonnektor** (“assembly connector”) zwischen zwei oder mehreren Parts oder Ports auf Parts definiert, dass ein oder mehrere Parts Services für andere Parts bereitstellen.

UML 2.3. Superstructure Specification

- Schlecht: sehen aus wie eine normale Assoziation! → Vorsicht

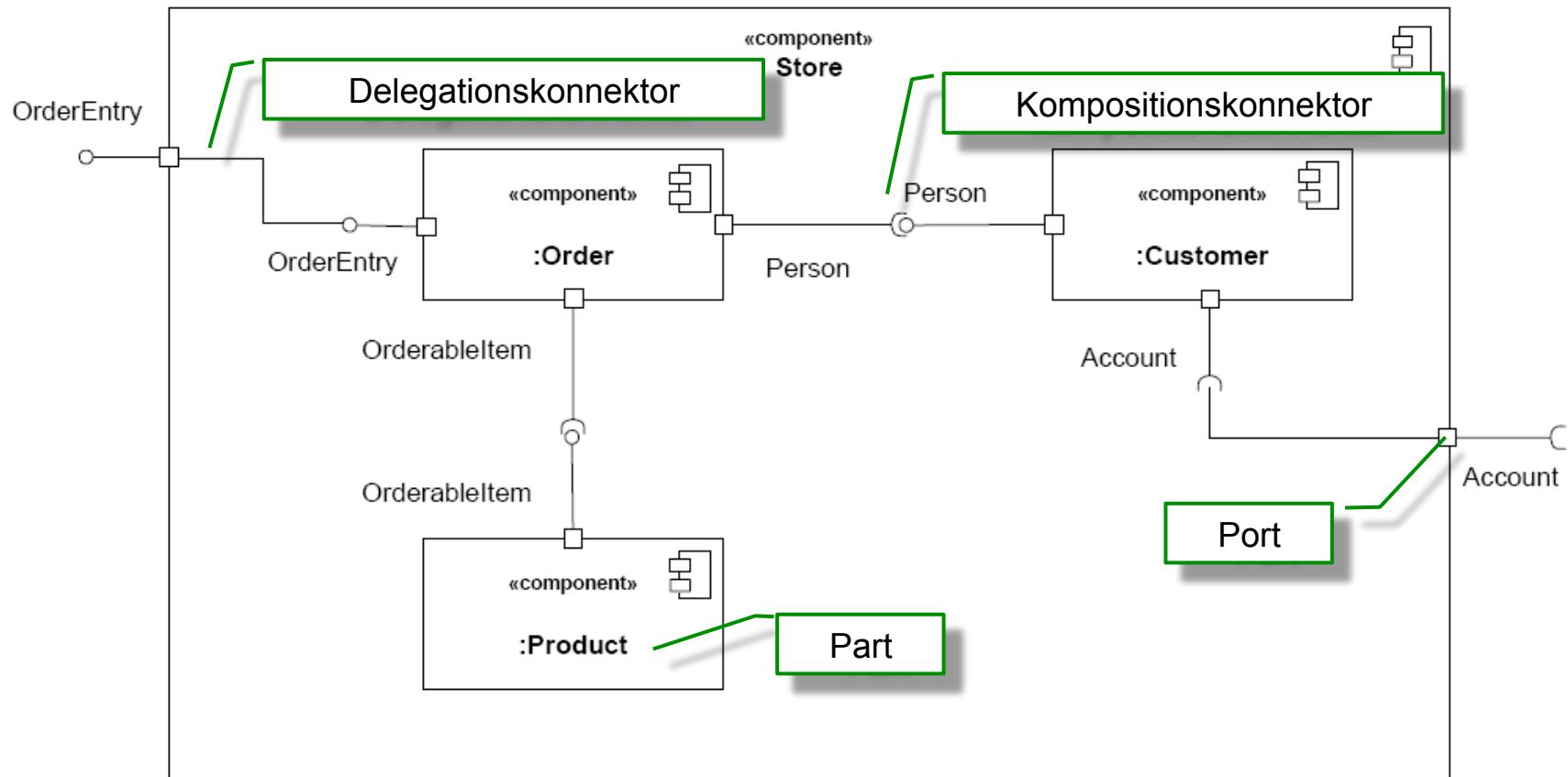


Konnektor allgemein



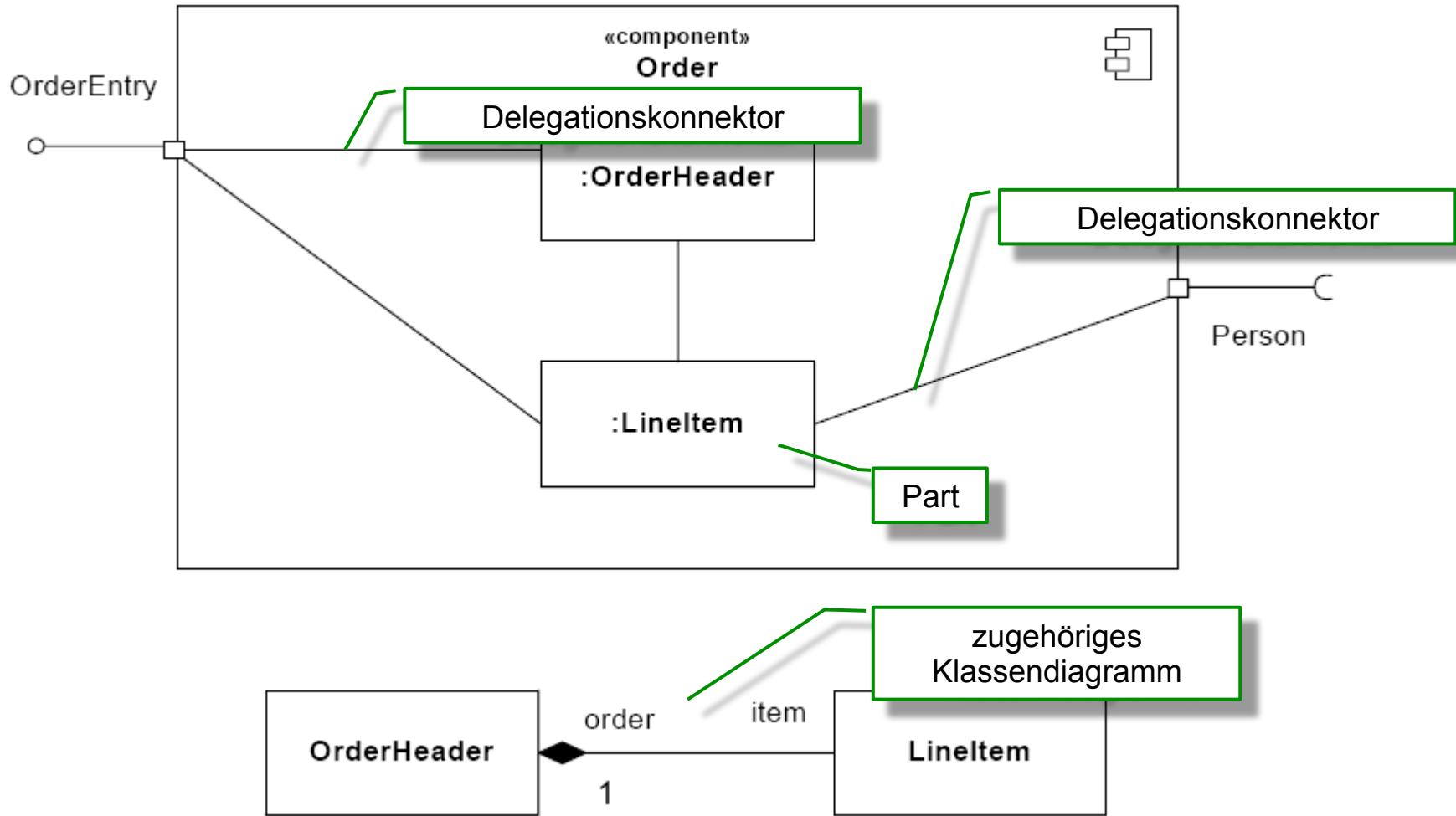
Kompositionskonnektor Alternative

## UML Kompositionssstrukturdiagramm – Verknüpfung von Parts



Quelle: UML 2.3. Superstructure Specification

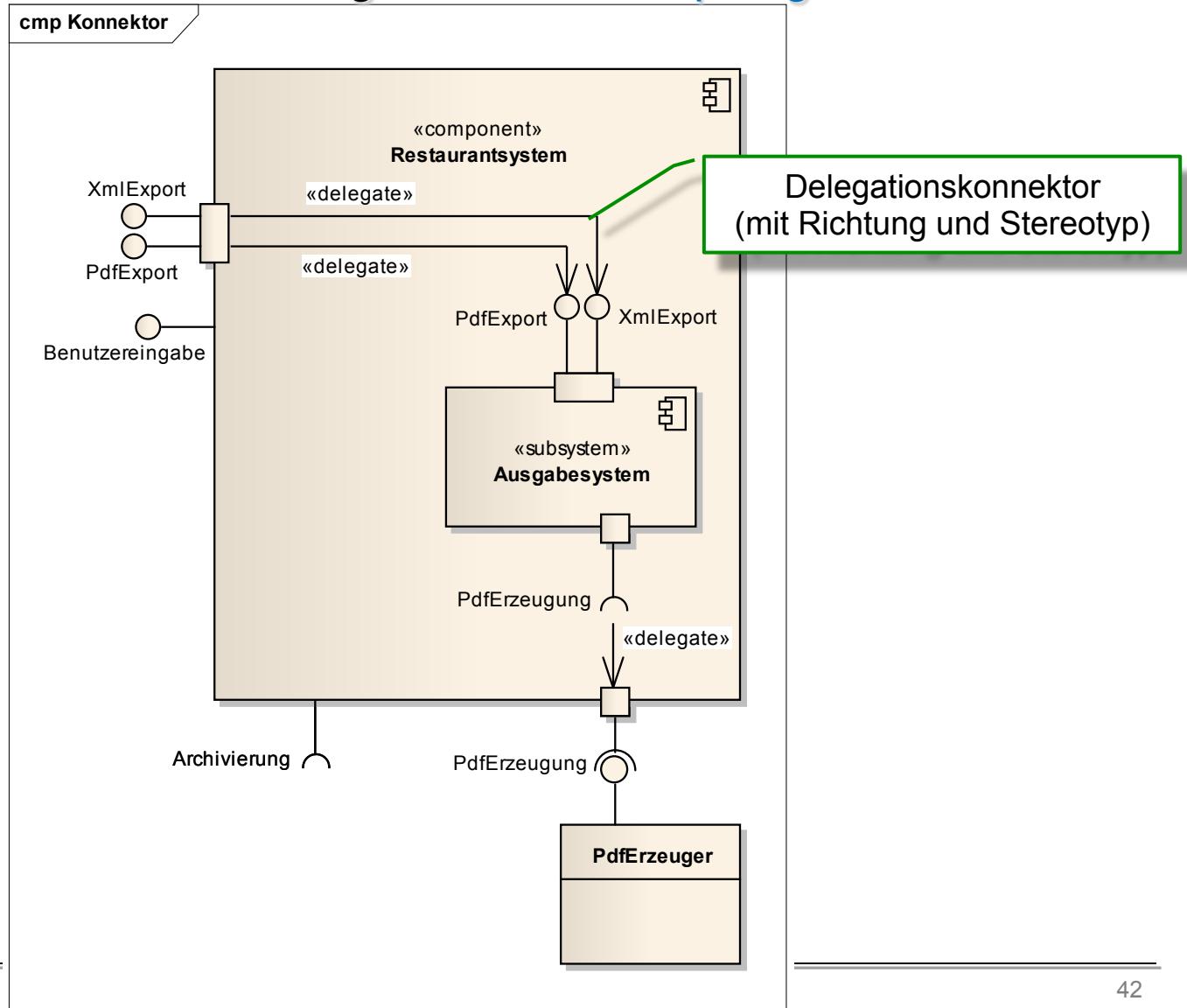
## UML Kompositionssstrukturdiagramm – Verknüpfung von Parts



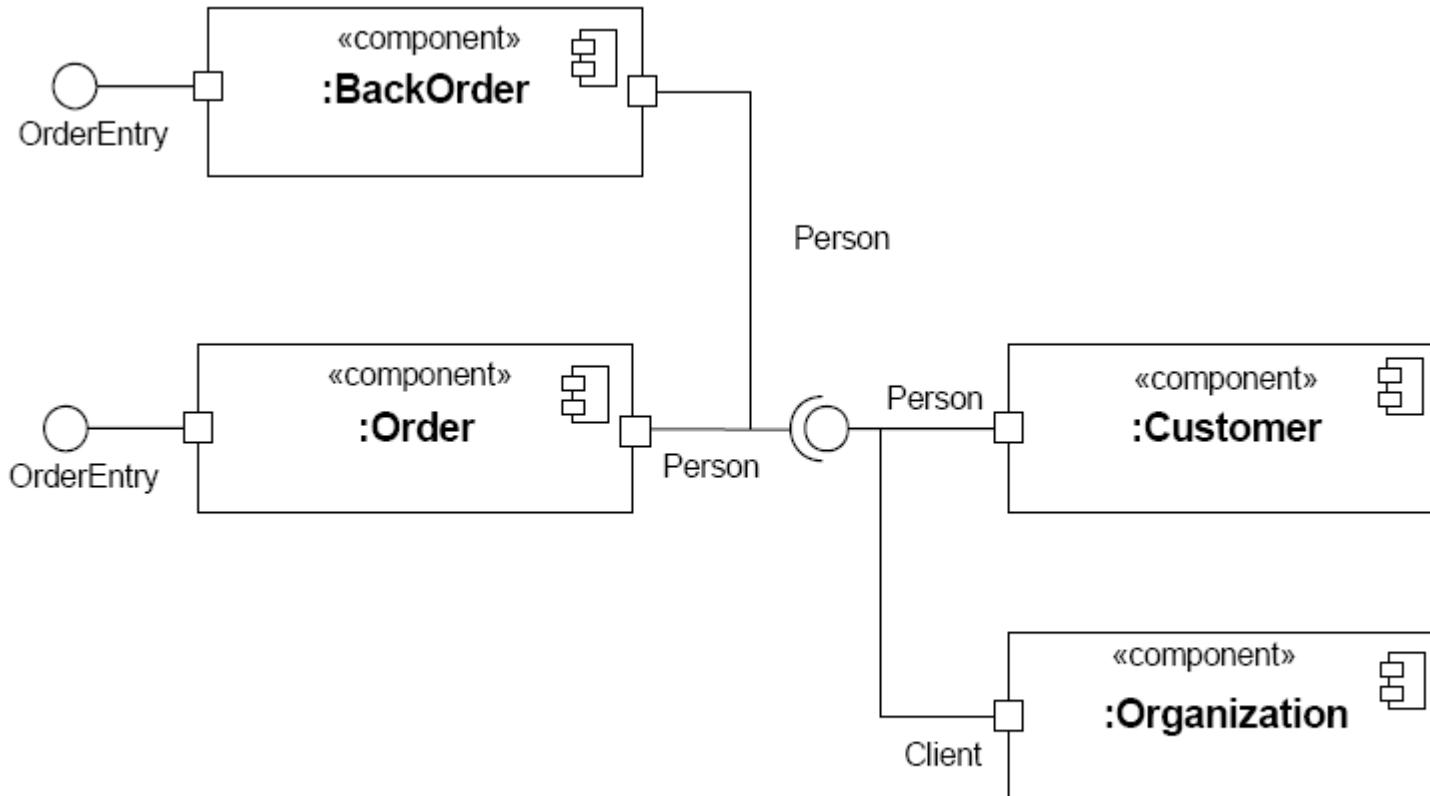
Quelle: UML 2.3. Superstructure Specification



## UML Kompositionsstrukturdiagramm – Verknüpfung von Parts



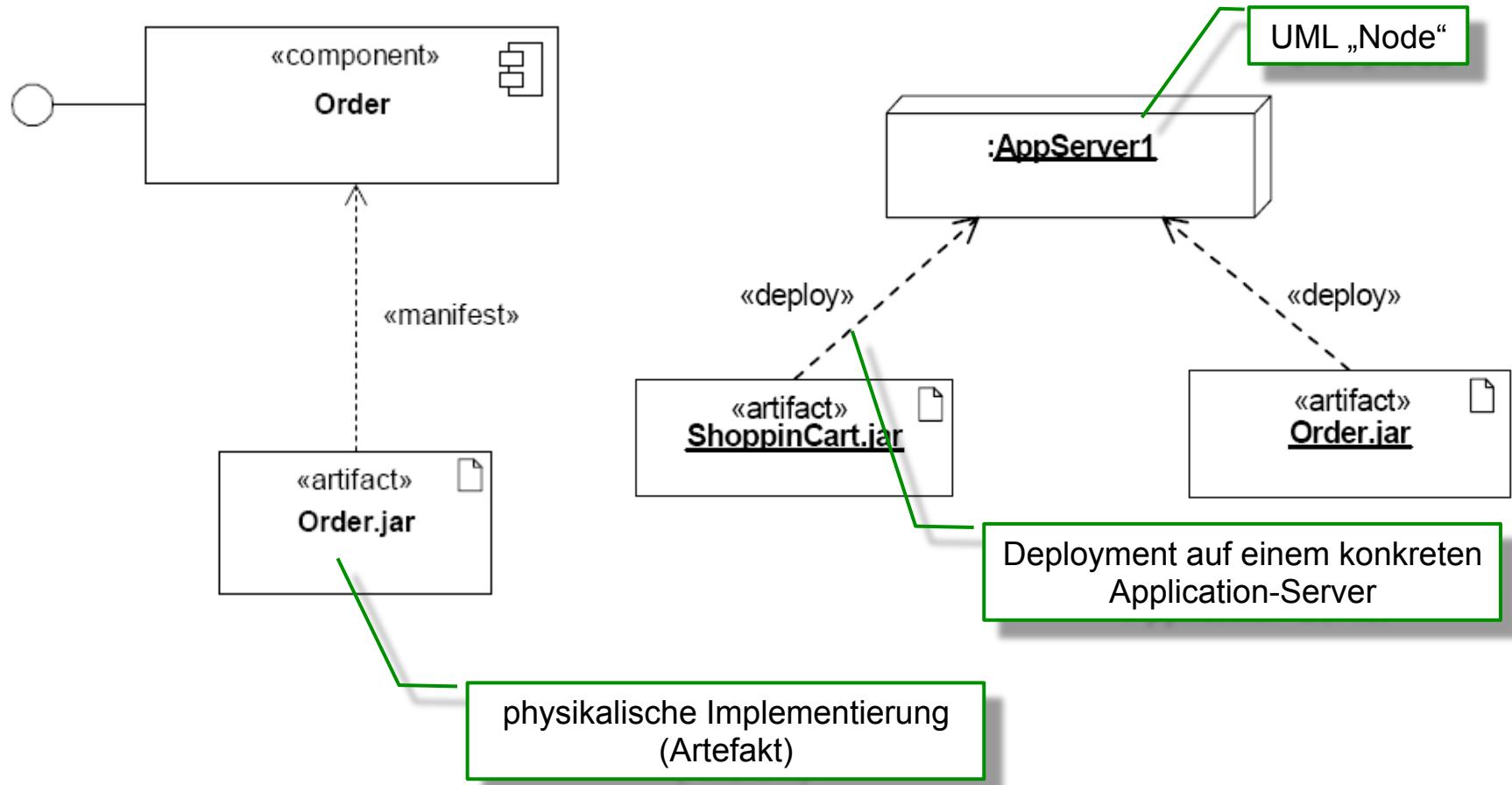
## UML Kompositionssstrukturdiagramm – Verknüpfung von Parts



Note: Client interface is a subtype of Person interface

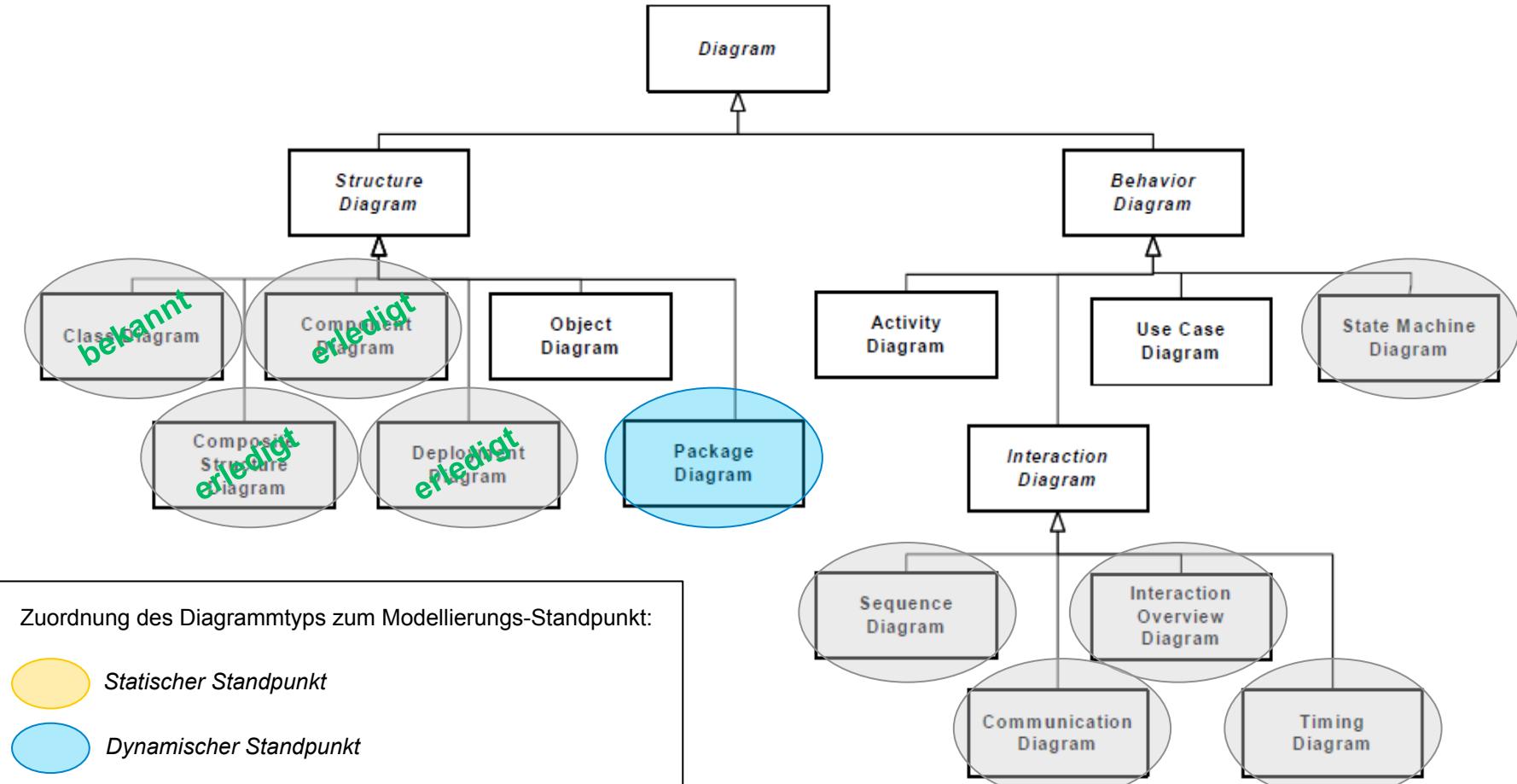
Quelle: UML 2.3. Superstructure Specification

# UML Komponenten- und Deploymentdiagramme – Implementierungs-Artefakte und Deployment





# Architekturbeschreibungs-Standpunkte und die UML



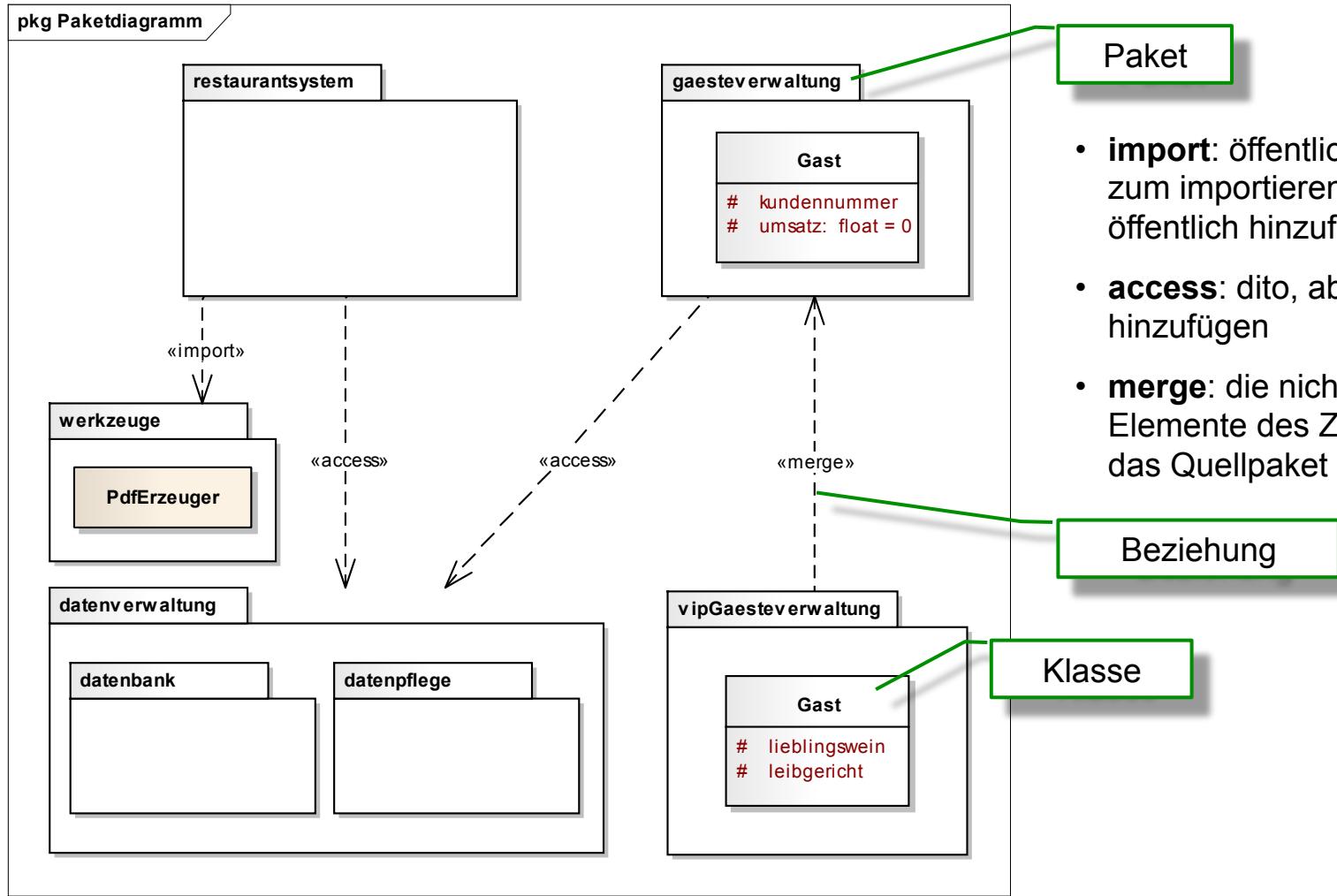
Quelle: UML 2.3. Superstructure Specification



# UML Paketdiagramme

- Paketdiagramme gruppieren UML-Elemente und definieren Namensräume
- Sie eignen sich zur horizontalen oder vertikalen Strukturierung
  - z. B. in Form von Schichten (GUI, Geschäftslogik, Zugriffsschicht, ...)
- Modularisieren das System und gestalten das Modell somit überschaubarer

# UML Paketdiagramme – Übersicht



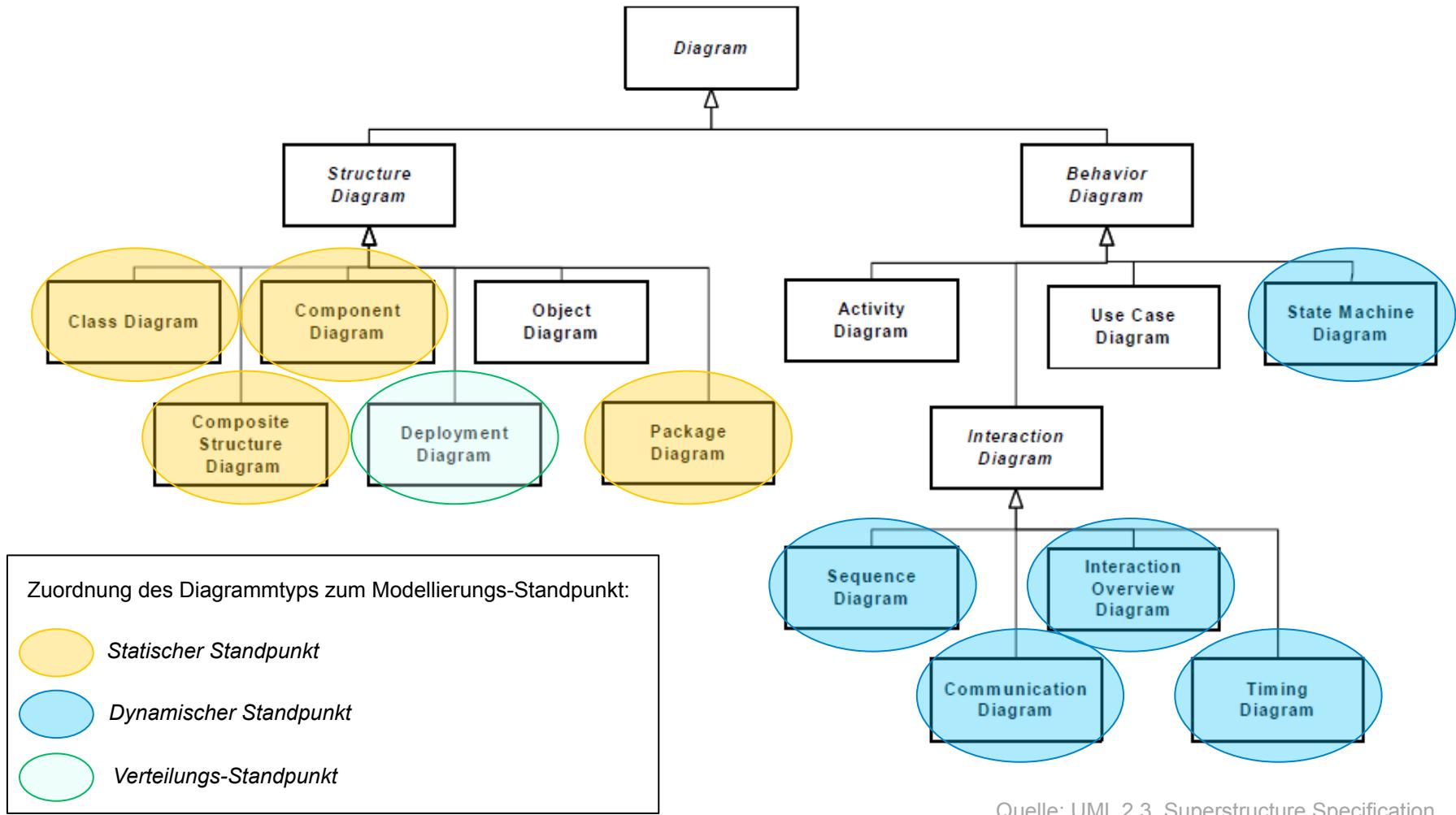


## UML Strukturdiagramme – Fazit

- Geeignet und sinnvoll, um den **Statischen Standpunkt** von Architekturen zu modellieren
- **Komponentendiagramme** zeigen die Aufteilung des Systems in Komponenten/Subsysteme, deren Ports und Schnittstellen, sowie die Abhängigkeiten zwischen den Komponenten
- **Kompositionsstrukturdiagramme** veranschaulichen den inneren Aufbau von Komponenten oder Klassen in Form von Bausteinen („Parts“), sowie die Zusammenhänge zwischen diesen Bausteinen; außerdem werden die Verbindungen zwischen den Parts und die Verknüpfung mit den Schnittstellen der Komponente dargestellt
- **Paketdiagramme** dienen der horizontalen oder vertikalen Strukturierung eines Modells



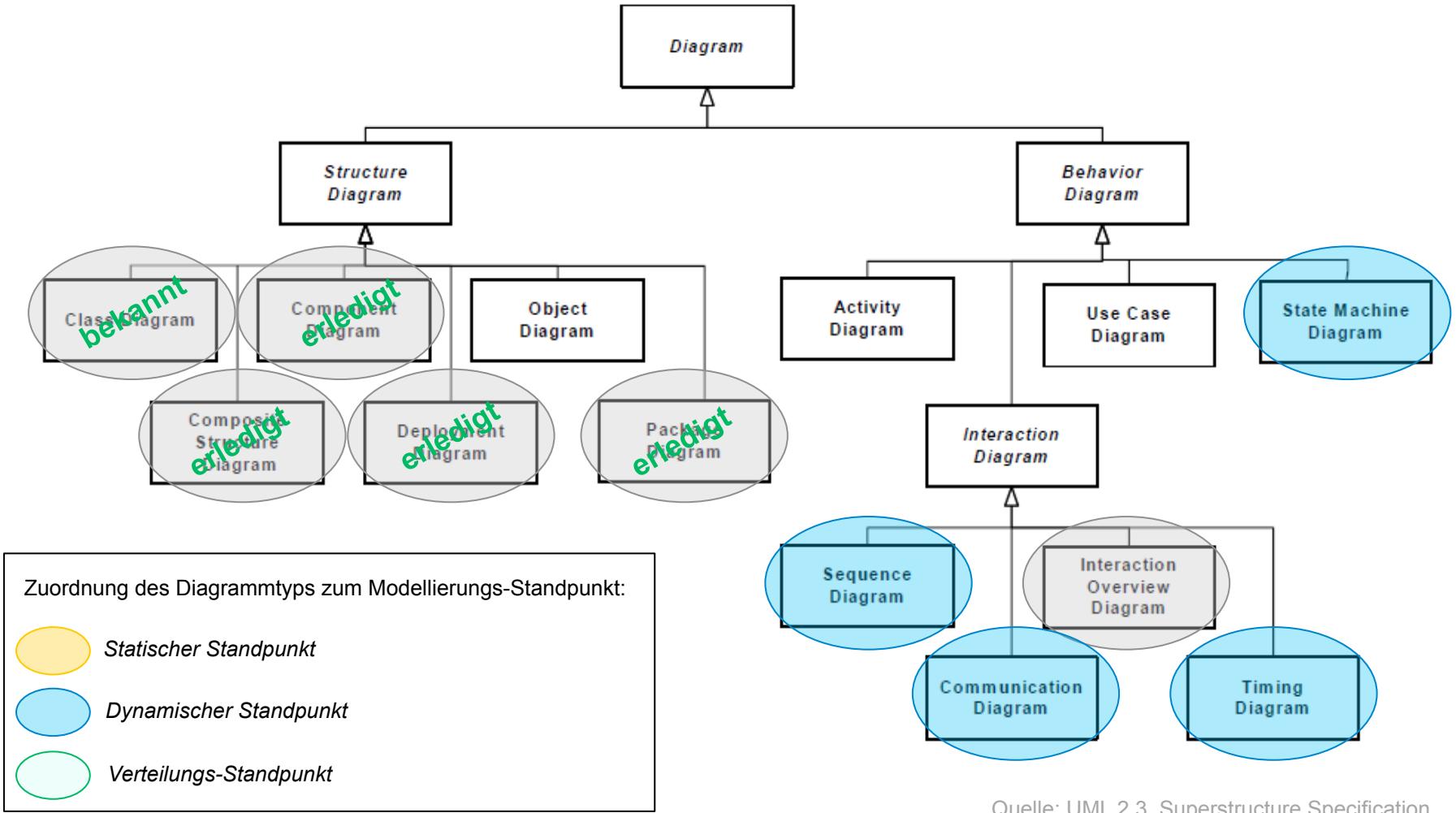
# Architekturbeschreibungs-Standpunkte und die UML



Quelle: UML 2.3. Superstructure Specification



# Architekturbeschreibungs-Standpunkte und die UML





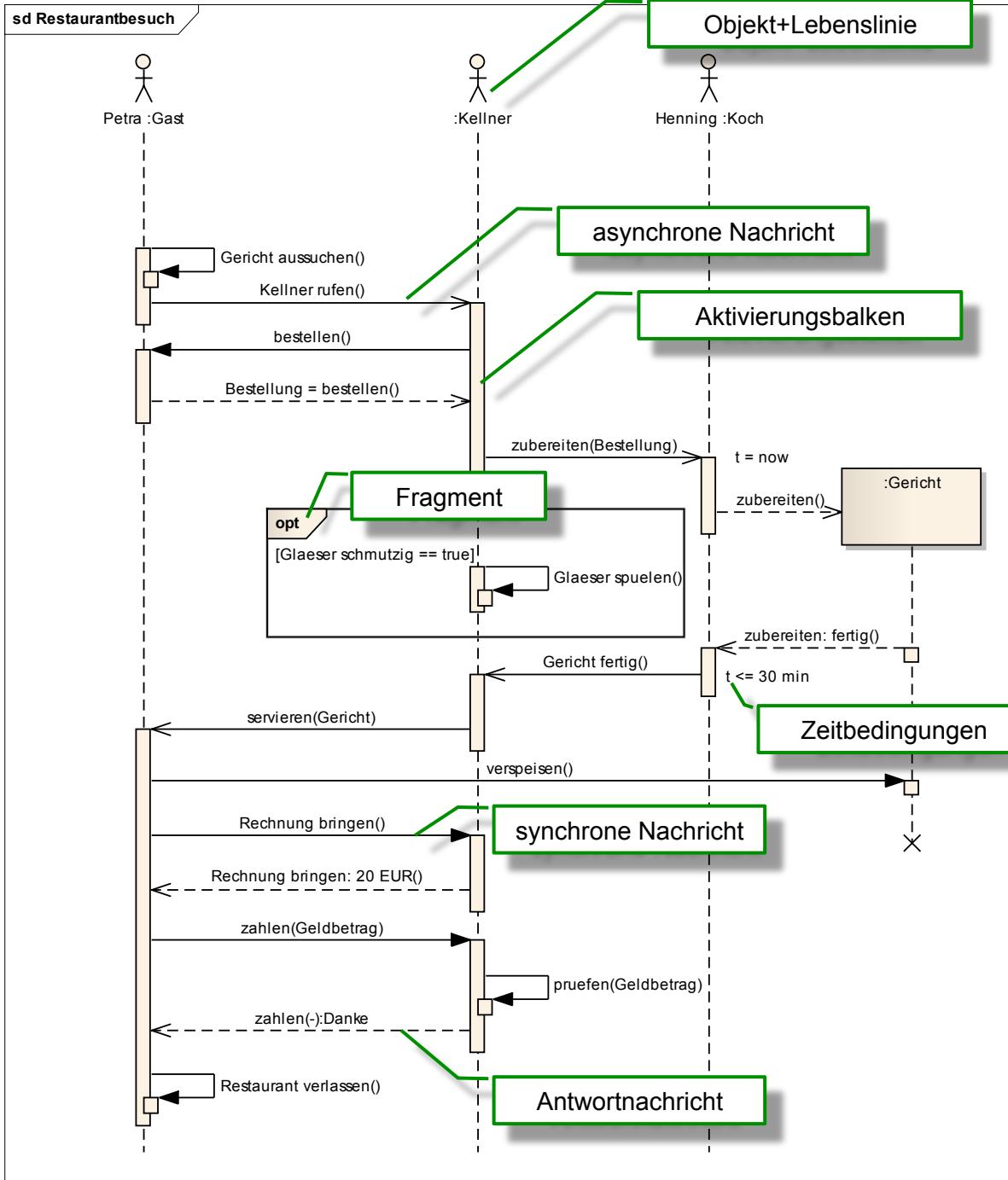
# UML Sequenzdiagramme

- Sequenzdiagramme modellieren Interaktionen zwischen Objekten
  - nicht nur Objekte von Klassen; dies können auch Komponenten-Objekte sein!
- Sie konzentrieren sich auf den Nachrichtenaustausch und nicht auf die Darstellung aller möglichen Ablaufpfade (im Ggs. zu Aktivitäts- oder Zustandsdiagrammen)
  - hier werden also nur bestimmte **Szenarien** modelliert
    - z. B. erfolgreiche Erteilung eines Auftrags
- In der Architektur sinnvoll zur Modellierung von Interaktionen zwischen Komponenten/Subsystemen zur Laufzeit
  - betreffen also den **Dynamischen Standpunkt**

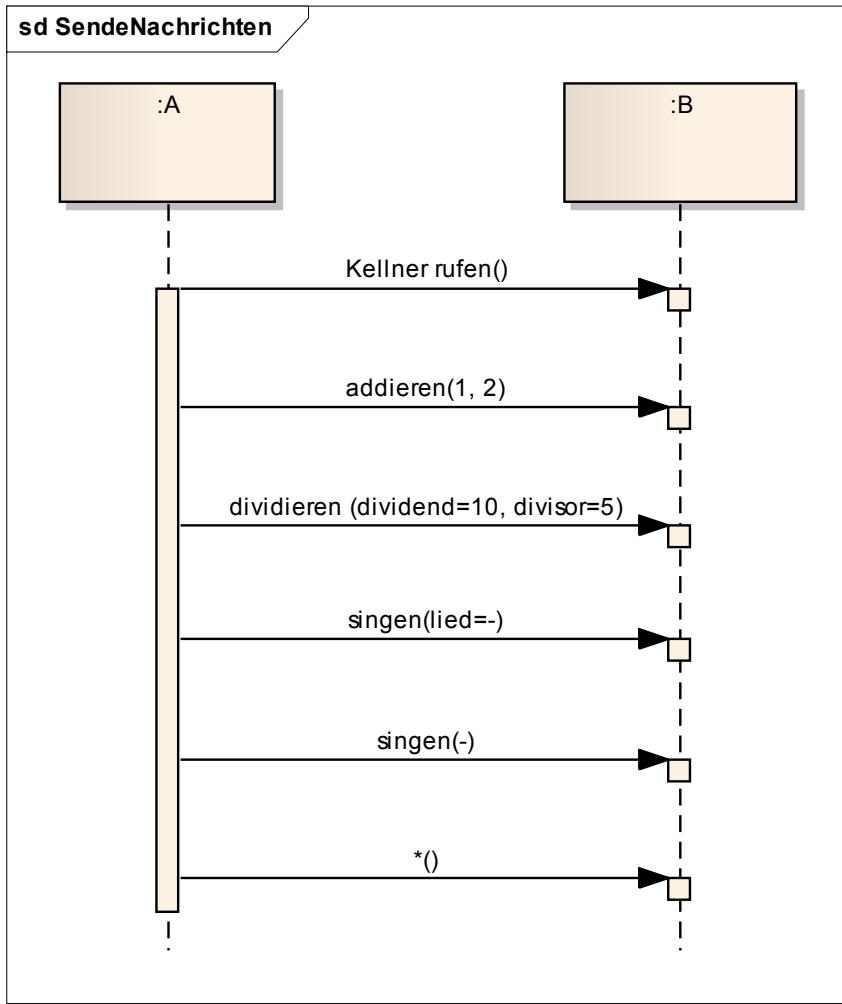


# UML Sequenzdiagramme – Übersicht

Quelle: [Kecher2009]



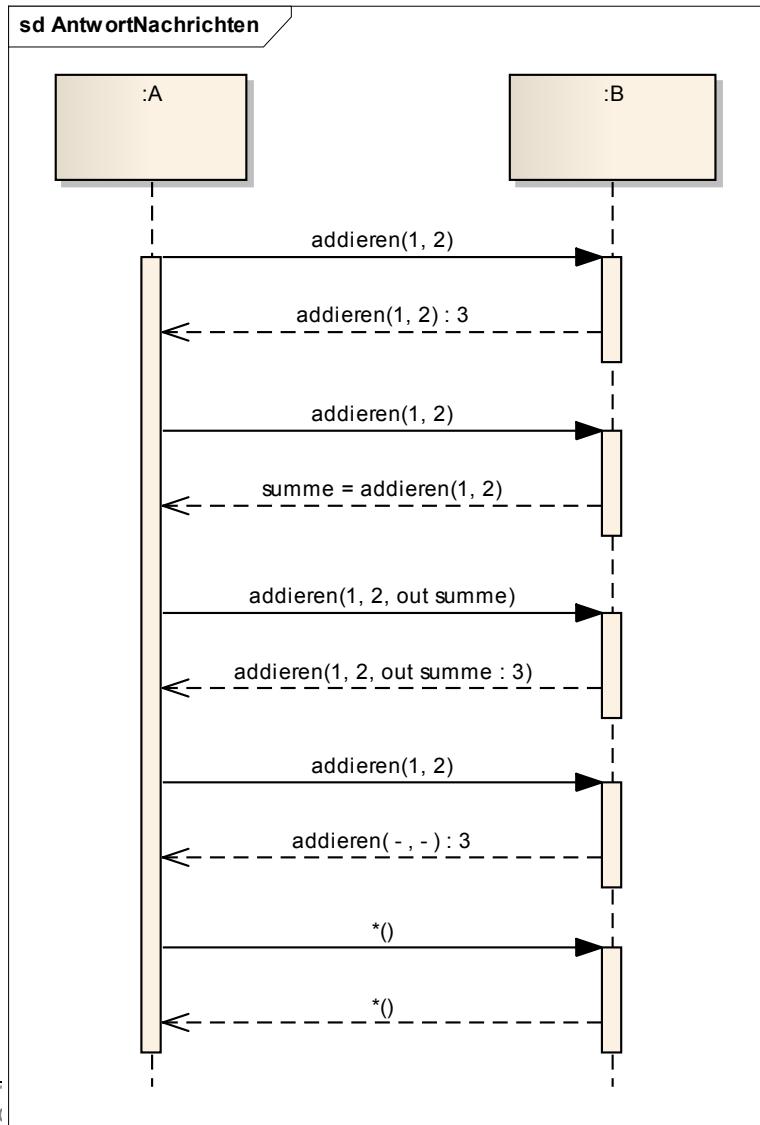
# UML Sequenzdiagramme – Nachrichtenaustausch 1



- ohne Argumente
- mit Argumenten
- mit expliziter Zuweisung
- unspezifiziertes Argument
- unspezifizierte Argumente
- unspezifizierte Nachricht

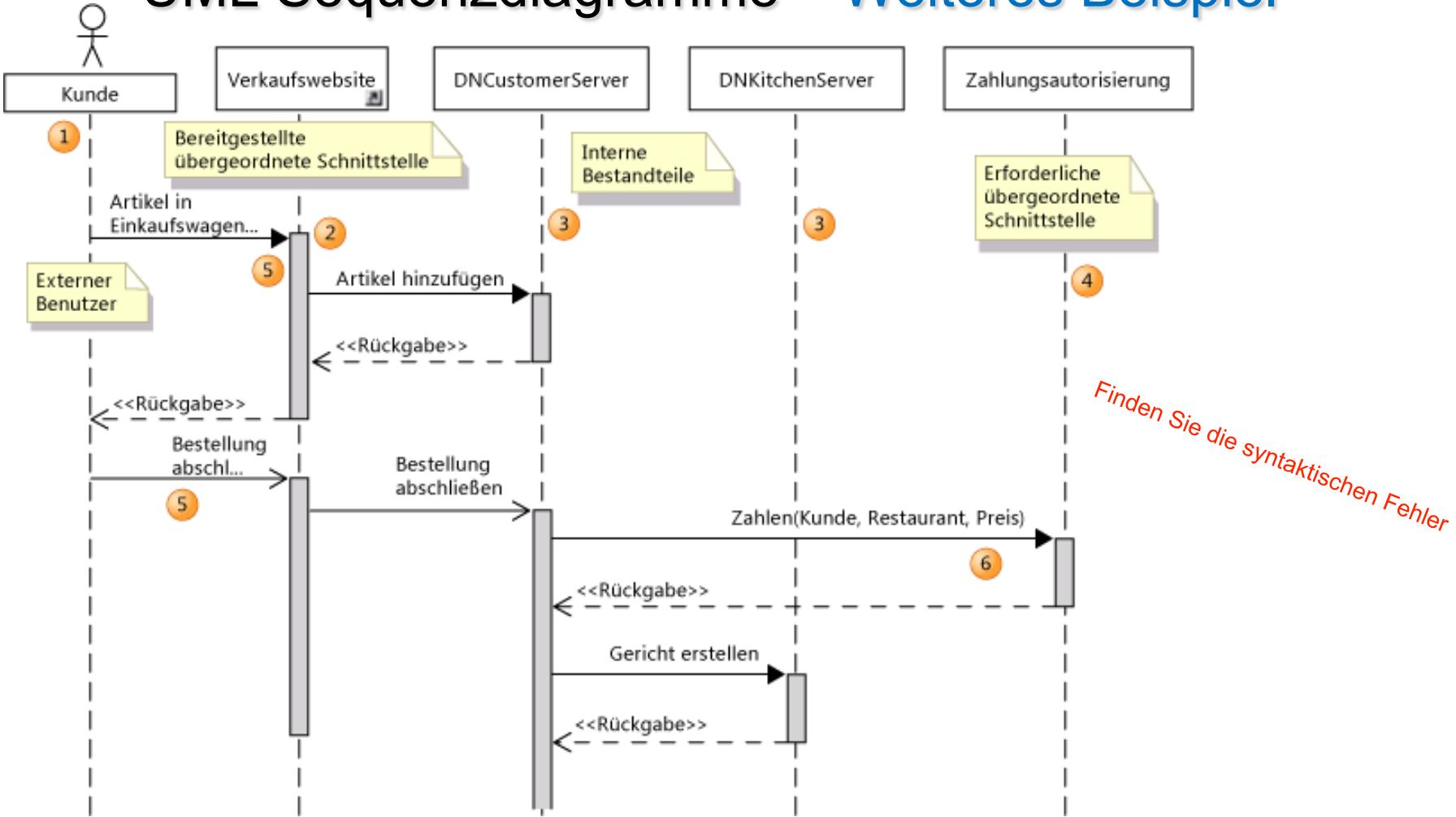
Quelle: [Kecher2009]

# UML Sequenzdiagramme – Nachrichtenaustausch 2



- mit Rückgabewert
- mit Attributzuweisung
- mit out-Parameter
- unspezifizierte Parameter
- unspezifizierte Antwort-Nachricht

# UML Sequenzdiagramme – Weiteres Beispiel



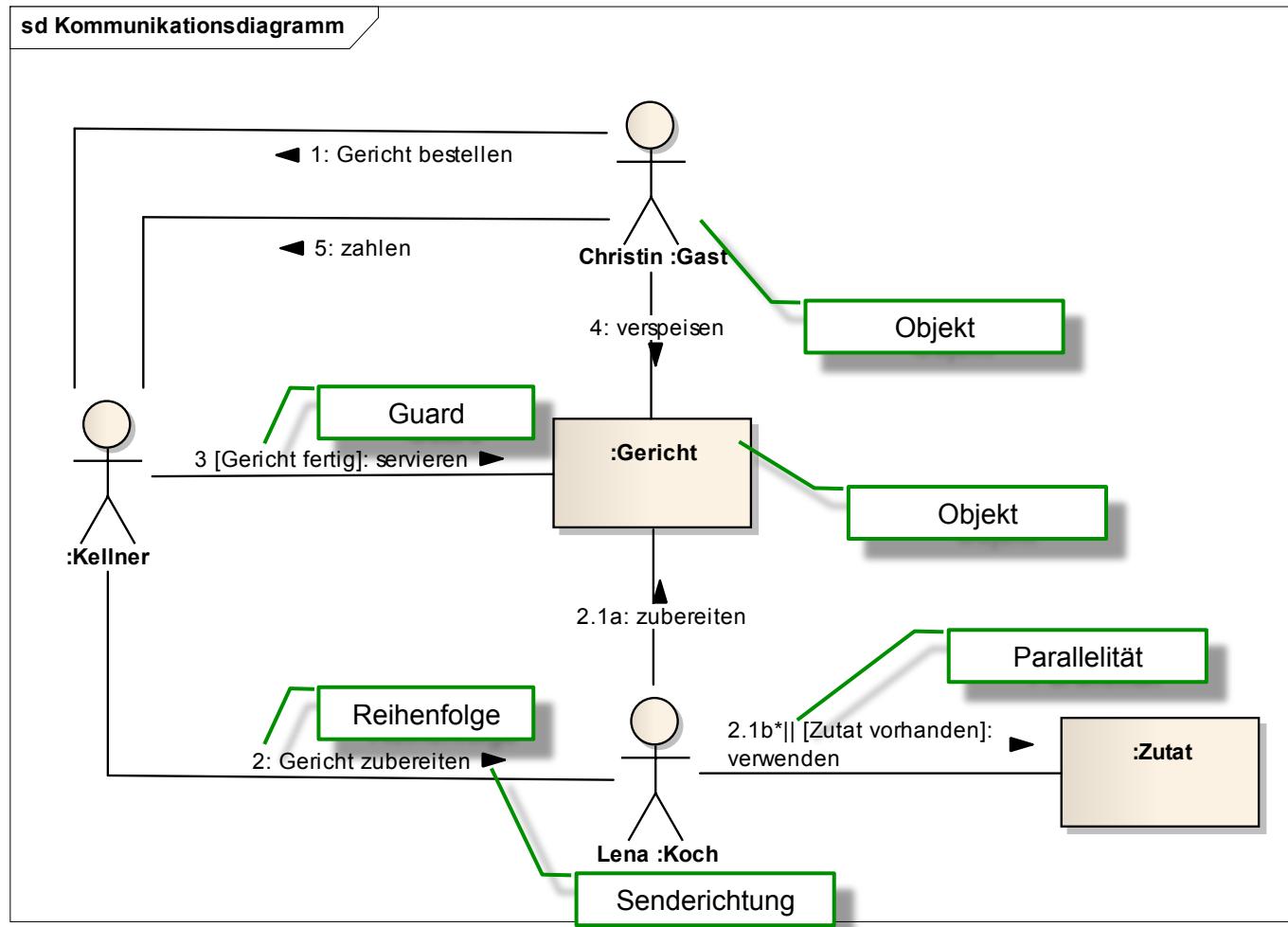
Quelle: <http://msdn.microsoft.com/de-de/library/dd409393.aspx>



## UML Kommunikationsdiagramme

- Kommunikationsdiagramme modellieren ebenfalls Interaktionen von Objekten und beschränken sich auf die Darstellung von **Szenarien**
- Kommunikationsdiagramme heben im Ggs. zu Sequenzdiagramme (die den zeitlichen Ablauf in den Vordergrund stellen) die **Beziehungen der Interaktionsteilnehmer** hervor
- In der Architektur sinnvoll zur Modellierung von Interaktionen zwischen Komponenten/Subsystemen zur Laufzeit
  - betreffen also den **Dynamischen Standpunkt**

# UML Kommunikationsdiagramme – Übersicht



Quelle: [Kecher2009]