University of Applied Sciences and Arts Dortmund

Softwaretechnik 2

Architekturmodellierung

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Struktur

Begriffe

Wie wird technisch gebaut?

„[When talking architecture] … we're talking something that's important" [Fowler 2003]

Bedeutung der Architekturmodellierung

- Wird ein Projekt basierend auf einer **unzureichenden Planung/Modellierung** durchgeführt, führt es nicht selten zu einem **nicht zufriedenstellenden Produkt**.
- Ziel ist es ein System in **funktional getrennte Blöcke** zu teilen. ⍰ **Modularisierung**
- Die Modularisierung ist notwendig, um ein System entwerfen zu können, das die **Grundanforderungen an eine Software** – Skalierbarkeit, Wartbarkeit – erfüllen kann.

Begriffe

Definition

- Eine Architektur ist eine **Beschreibung von System-Strukturen** (Datenströme, Modulare Strukturen, Prozess Strukturen usw.)
- Eine Architektur ist das **erste Artefakt** zur Analyse der Einhaltung von **Qualitätseigenschaften**.
- Eine Architektur ist eine Beschreibung der **Beziehung von Komponenten und Verbindungen**. [Bass et al. (1998)]

Definition

- Eine Architektur ist die **fundamentale Organisation eines Systems**, verkörpert durch seine **Komponenten**, deren **Beziehungen** und deren **Umwelt** .... [IEEE Standard 1471 (2000)]

Begriffe

Begriffsverwirrung

- Architektur kann eine sehr **grobe Form der Darstellung** der Systemteile bedeuten, die dazu dient einen raschen Überblick zu erhalten.
- Architektur kann aber auch den **technisch detaillierten Aufbau** eines konkreten Systemteils bedeuten.

Verwendung der Begriffe „Architektur“ und „Entwurf/Design“ für diese Veranstaltung

- **(Software)Architektur** als Architektur im „Großen“
- **Entwurf/Design** als Architektur im „Kleinen“

Begriffe

Strategischer Entwurf Architektur

- Design im „Großen“
- Entwurf auf Systemebene
- Entscheidungen über Technologie, Gesamtaufbau des Systems, etc.

Taktischer Entwurf Entwurf/Design

- Design im „Kleinen“
- Entwurf im klassischen Software Engineering Sinne
  (auf Bausteinebene, nicht auf Systemebene)
- Entscheidungen über Aufbau der einzelnen Bausteine
  (z.B. Entwurf der Klassen, usw.)

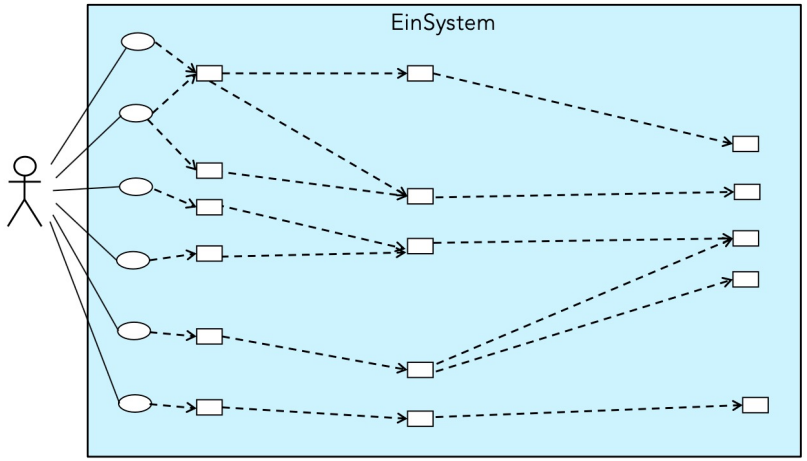
Ziele und Aufgabe der Architekturmodellierung

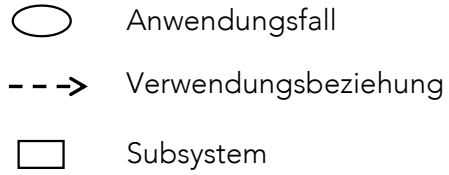
- Beschreibung eines Systems, das die **Anforderungen** des Kunden berücksichtigt
- Beschreibung des Systems auf **technischer Ebene**
- **Ausgangspunkt für Implementierung**
- **Kommunikationsmittel** zwischen den Projekt-Stakeholdern
- Ausdruck des **Architekturstils**

Struktur

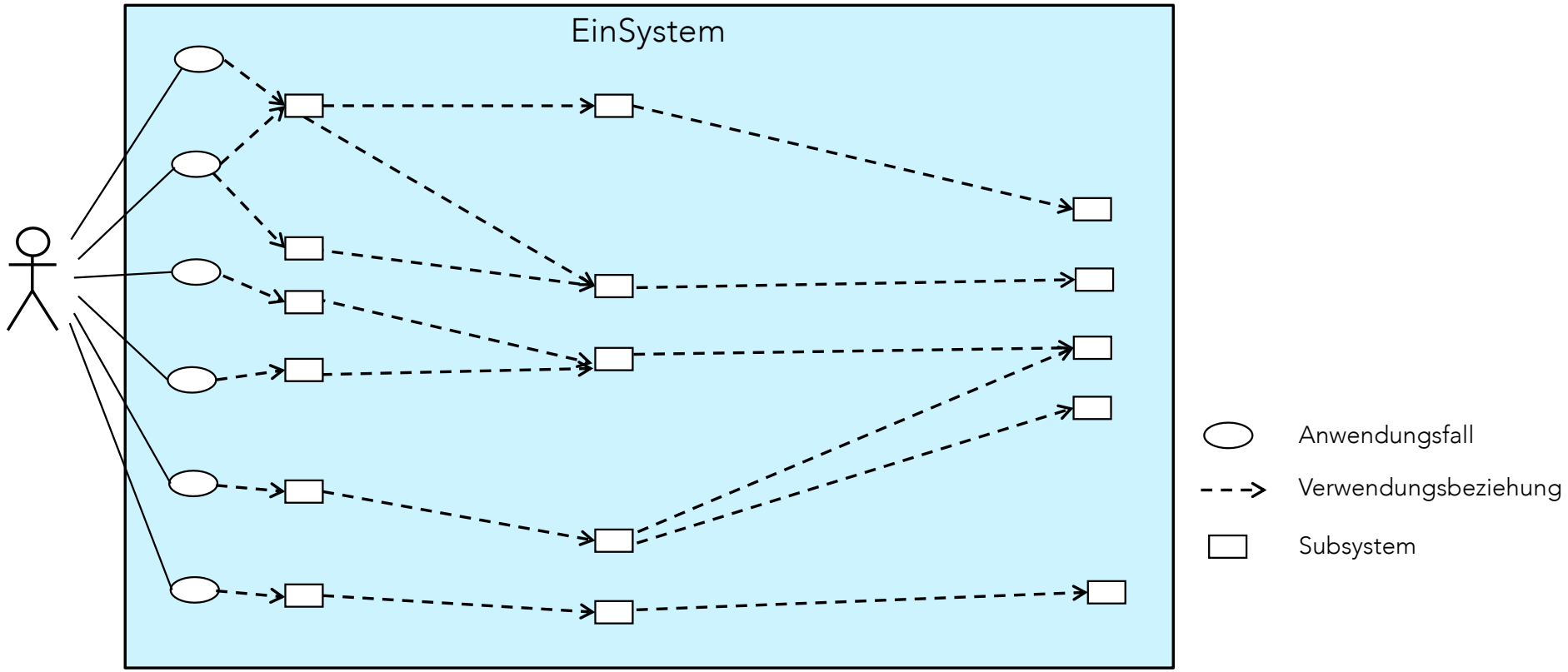
Beispiel: Vom Analysemodell zur Systemdekomposition

- Gruppieren nach ähnlichen Funktionalitäten (Komponenten bilden)
- Angebotene und genutzte Dienste identifizieren (Schnittstellen)

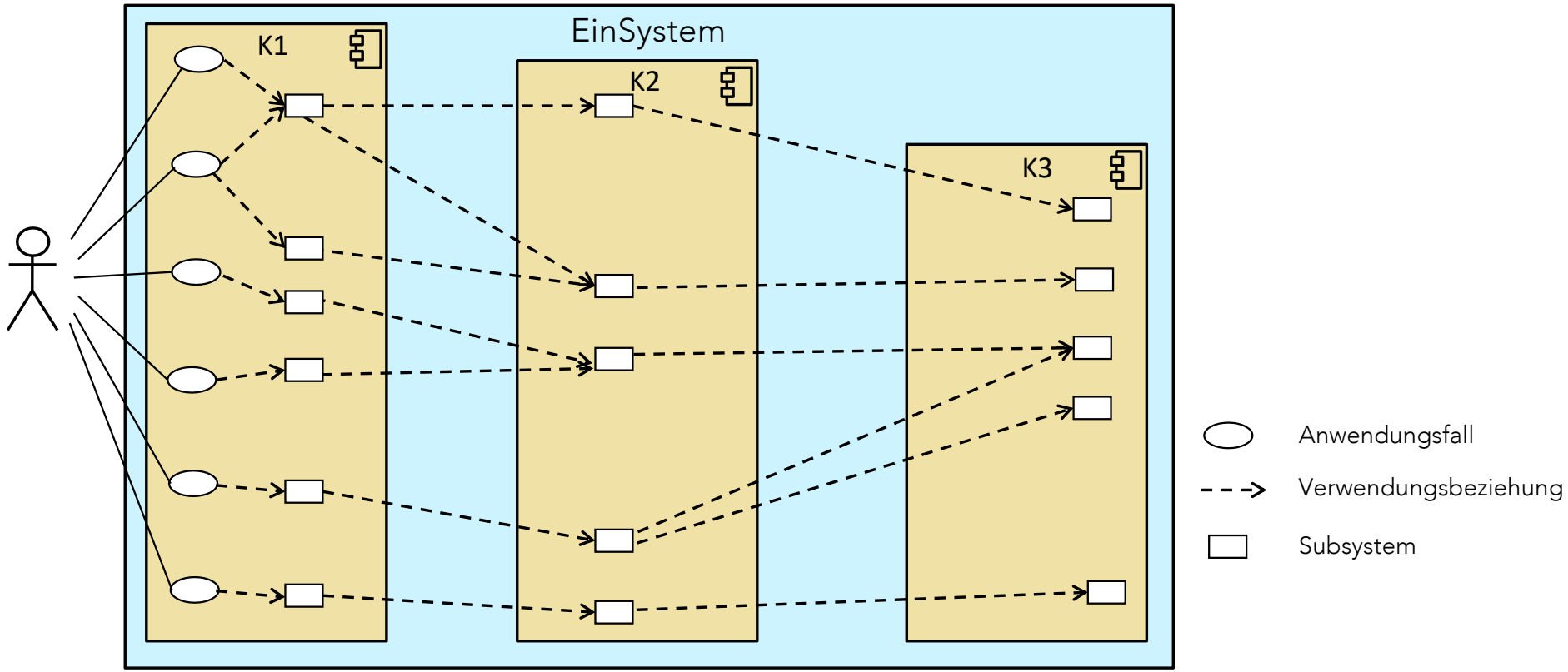




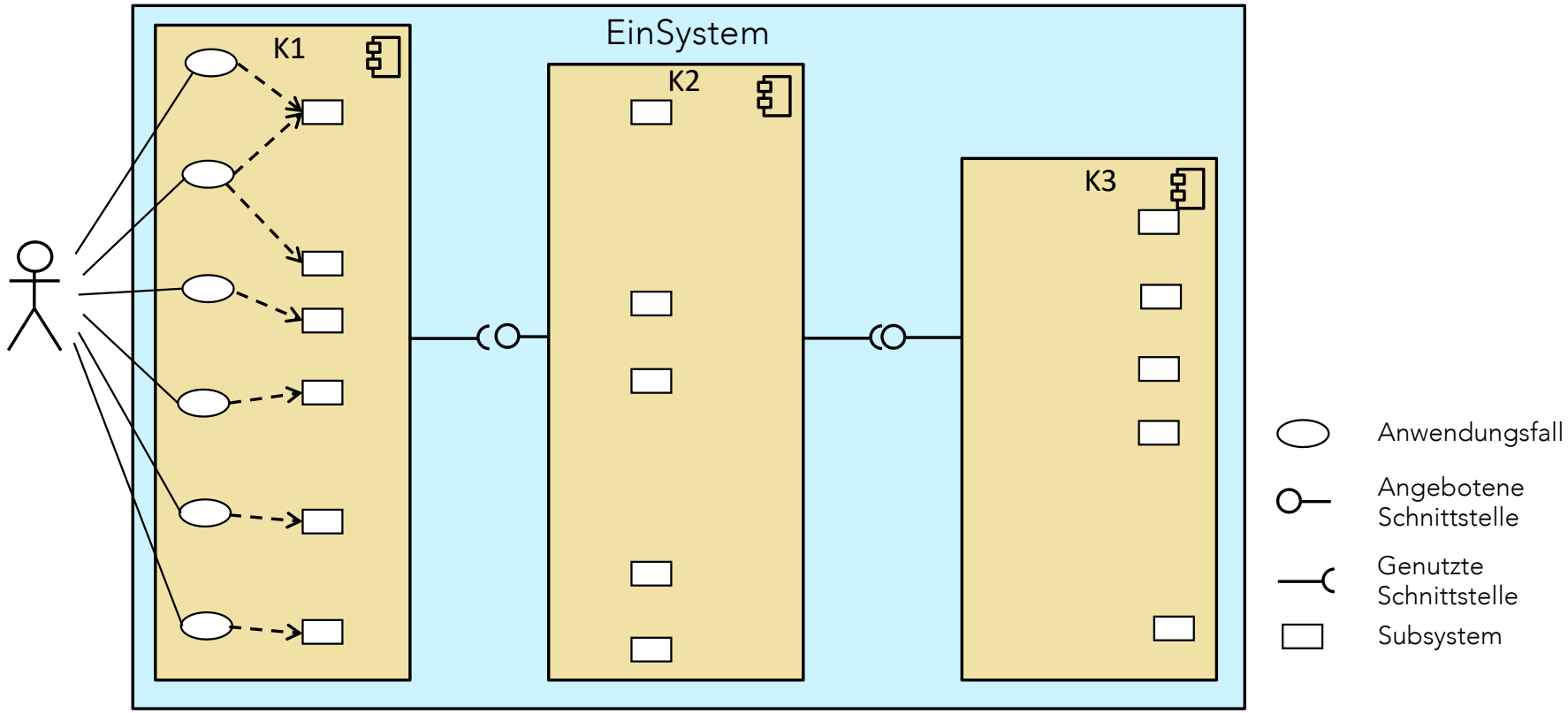
Architekturmodellierung



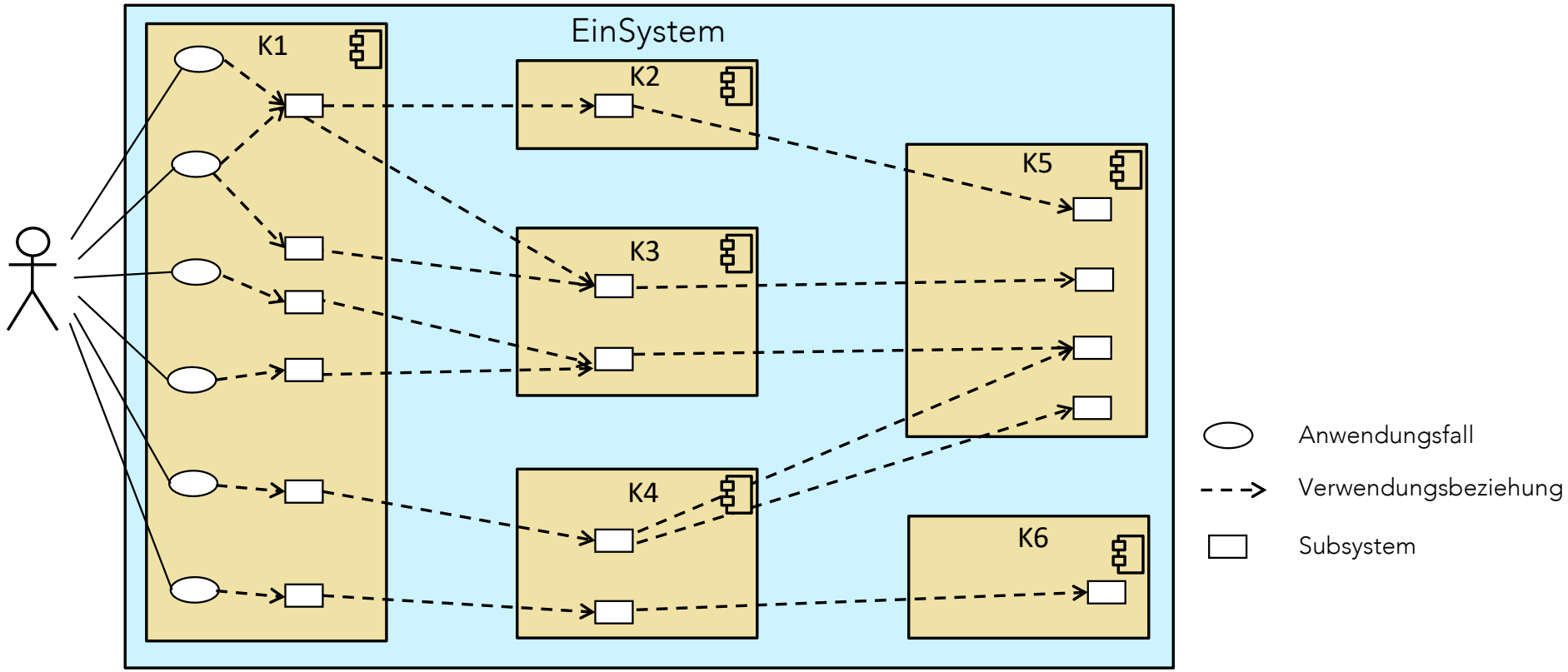
Architekturmodellierung



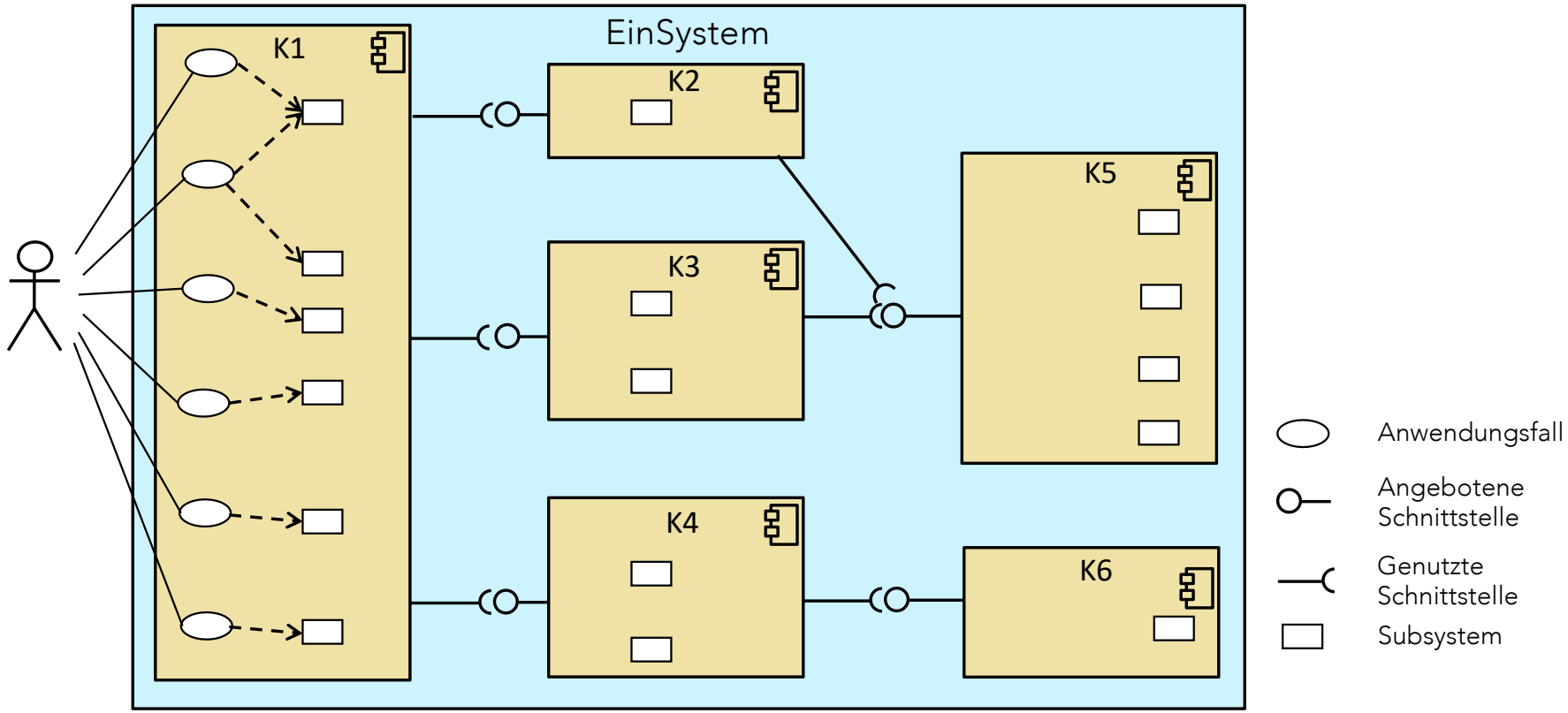
Architekturmodellierung



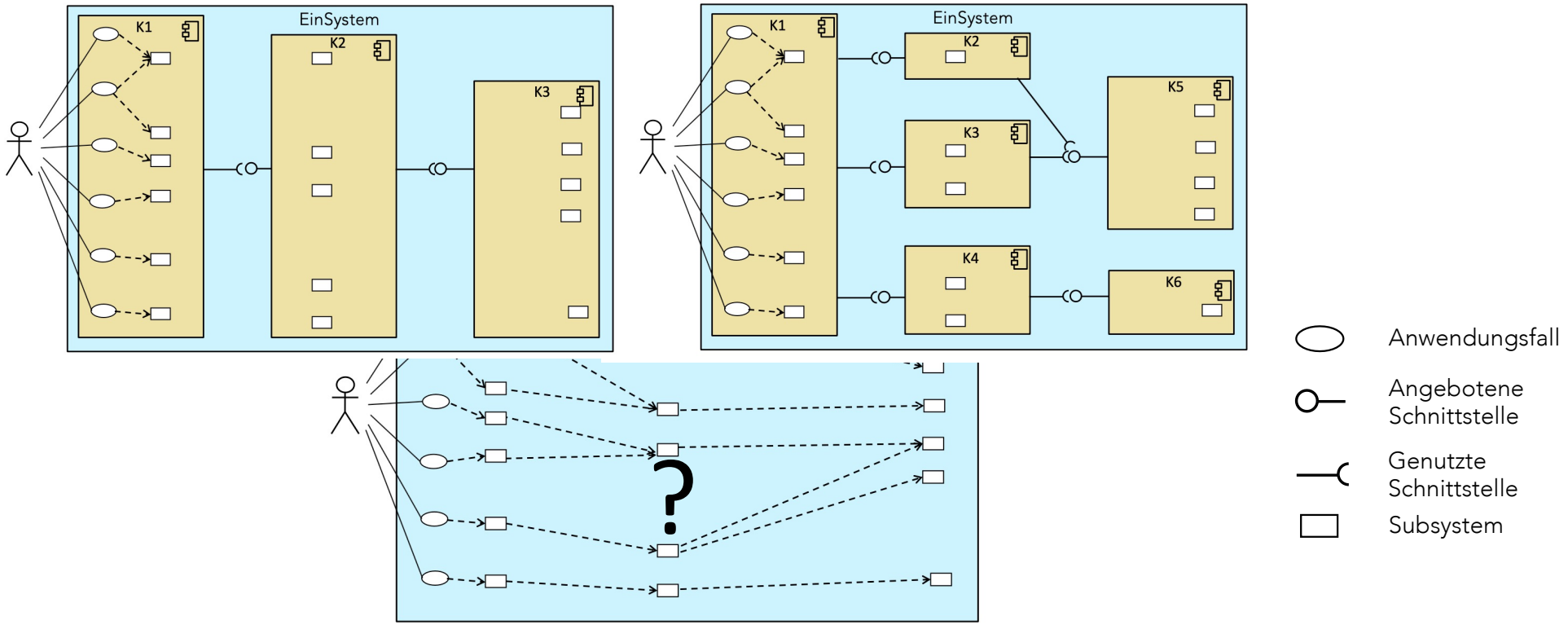
Architekturmodellierung



Architekturmodellierung



Architekturmodellierung



Struktur

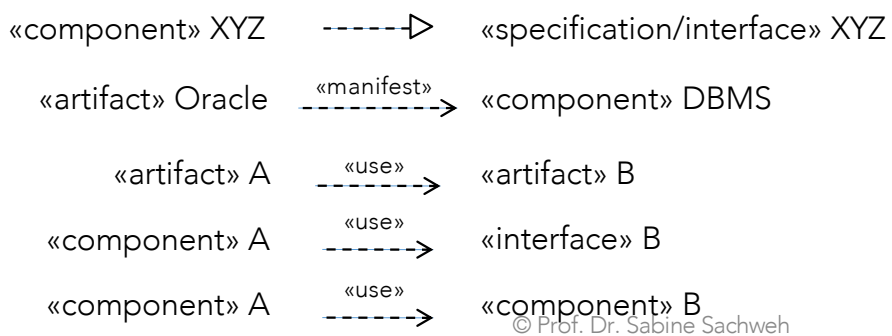
Komponentendiagramm

- Liefert **Antwort auf die Fragen**:
  a) „Wie ist das Softwaresystem strukturiert und
  b) wie werden diese Strukturen erzeugt?“

⇒

- Komponentendiagramm zeigt
  - **Definition von Komponenten**
  - **Abhängigkeiten** dazwischen

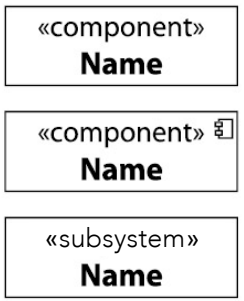
- **Wesentliche Modellierungselemente**
  - Komponente
  - Schnittstelle
  - Klasse
  - Artefakt
  - Port
  - Beziehungen
    - Realisierungsbeziehung
    - Implementierungsbeziehung
    - Verwendungsbeziehung



Komponente

- Eigenschaften einer Komponente in UML2
  - **modularer Teil** eines Systems
  - **Abstraktion** und **Kapselung** beliebig komplexer Struktur
  - verfügt über **wohldefinierte Schnittstellen** (angebotene und benötigte, Gruppierung durch Ports) ⇒ **einfache Verknüpfung** von Komponenten
  - im **gesamten Softwareentwicklungszyklus** verwendbar
  - kann sowohl **logische** (z.B. Anwendungsfälle) als auch **physische Modellelemente** (z.B. Artefakte) umfassen

- Notationsvarianten



- Komponenten können auch in anderen Strukturdiagrammen, z.B. im **Klassendiagramm**, verwendet werden

Artefakte

Artefakte

- physische Informationseinheit
  wie z.B. Modelle, Quellcode, Skripte, ausführbare Binärdateien, Tabellen (innerhalb einer rel. DB), Dokumente, E-Mails, etc.
- sie werden im Entwicklungsprozess oder zur Laufzeit produziert oder konsumiert
- sie können durch Stereotypen konkretisiert werden
  folgende Stereotypen sind in der UML vorgesehen:
  - «file» physische Datei innerhalb des Systems
  - «document» hat keinen Quell- oder ausführbaren Code
  - «executable» enthält Anweisungen für direkte Ausführung
  - «source» enthält übersetz- oder interpretierbaren Code
  - «library» enthält statisch oder dynamisch verwendbare Bibliothek

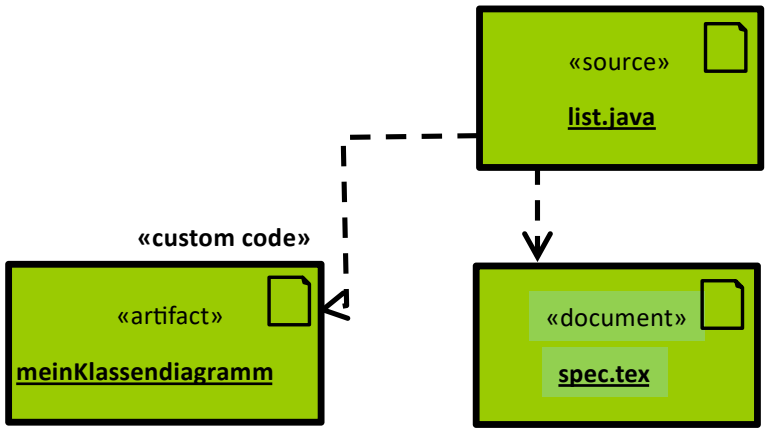
Durch Profile können **weitere Stereotype** für verschiedenste Plattformen definiert werden
z.B.: EJB Profil definiert «jar» als Unterklasse von «executeable»!

Komponentendiagramm

Beispiel: Warenkorb

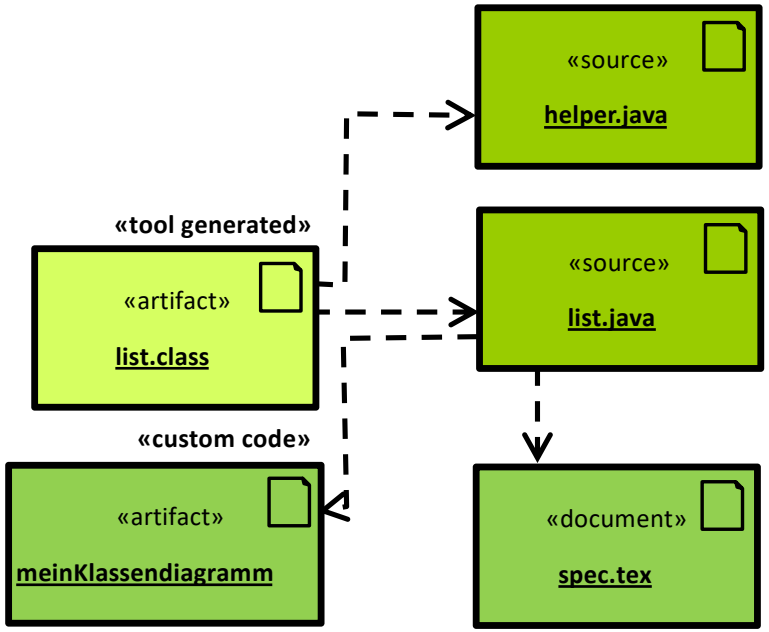
Komponentendiagramm

Beispiel: Warenkorb



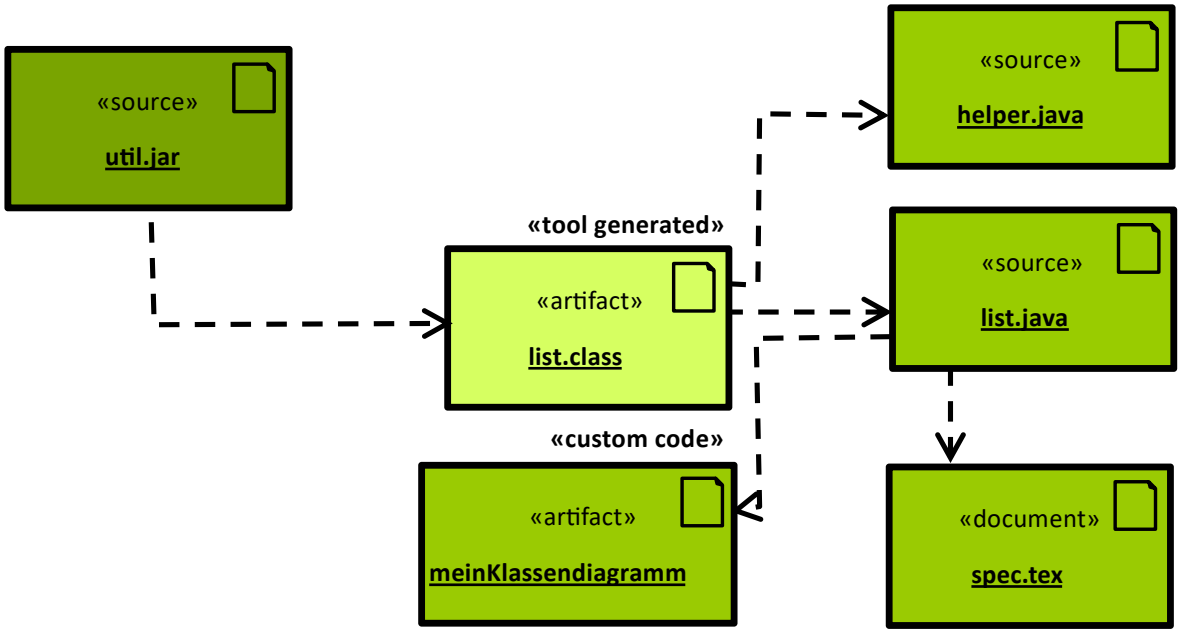
Komponentendiagramm

Beispiel: Warenkorb



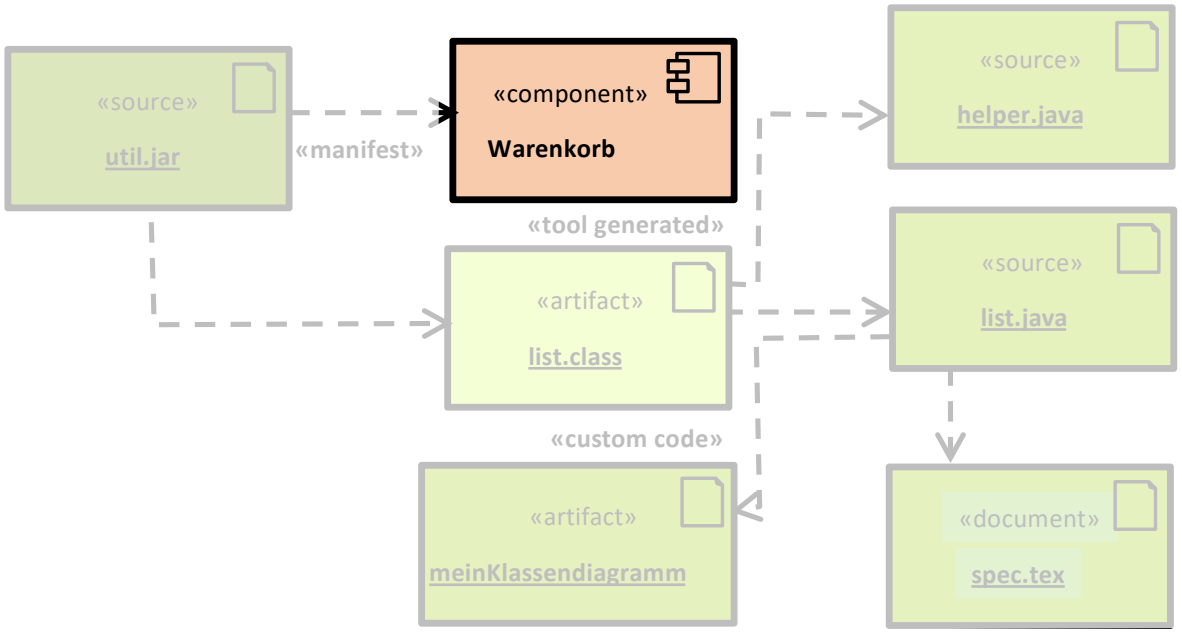
Komponentendiagramm

Beispiel: Warenkorb



Komponentendiagramm

Beispiel: Warenkorb

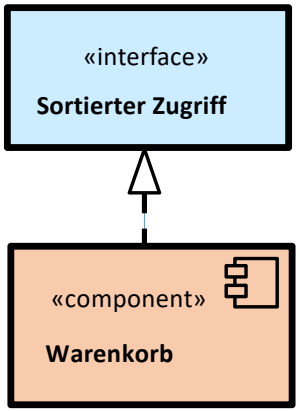


Komponentendiagramm

Beispiel: Warenkorb

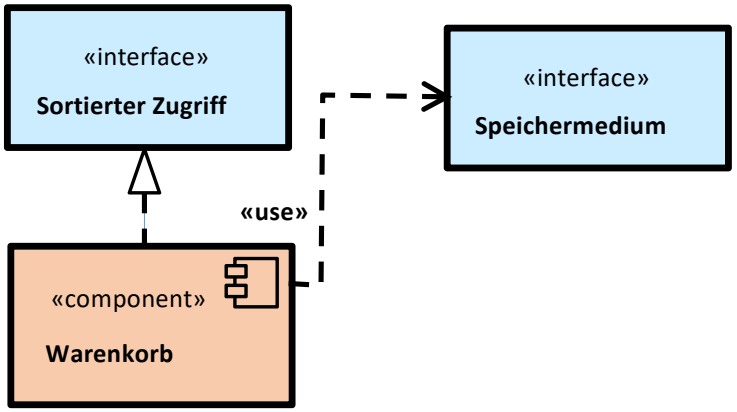
Komponentendiagramm

Beispiel: Warenkorb



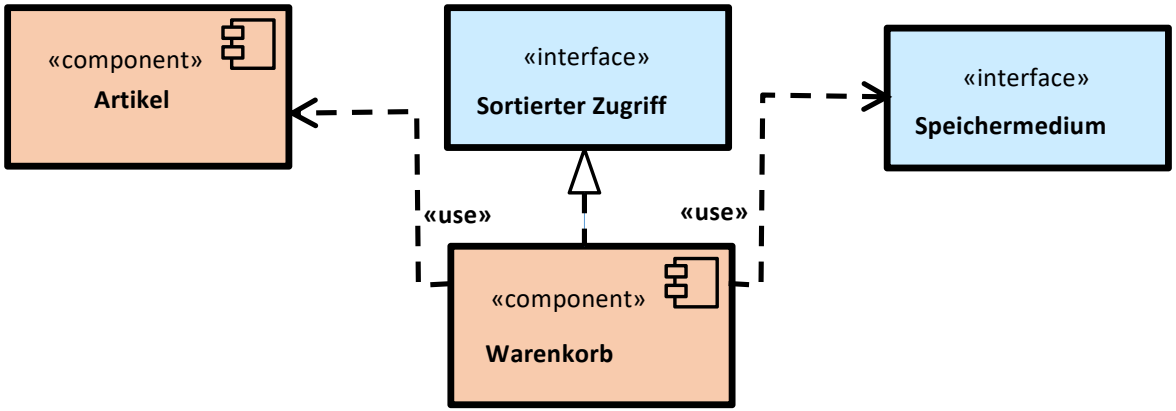
Komponentendiagramm

Beispiel: Warenkorb



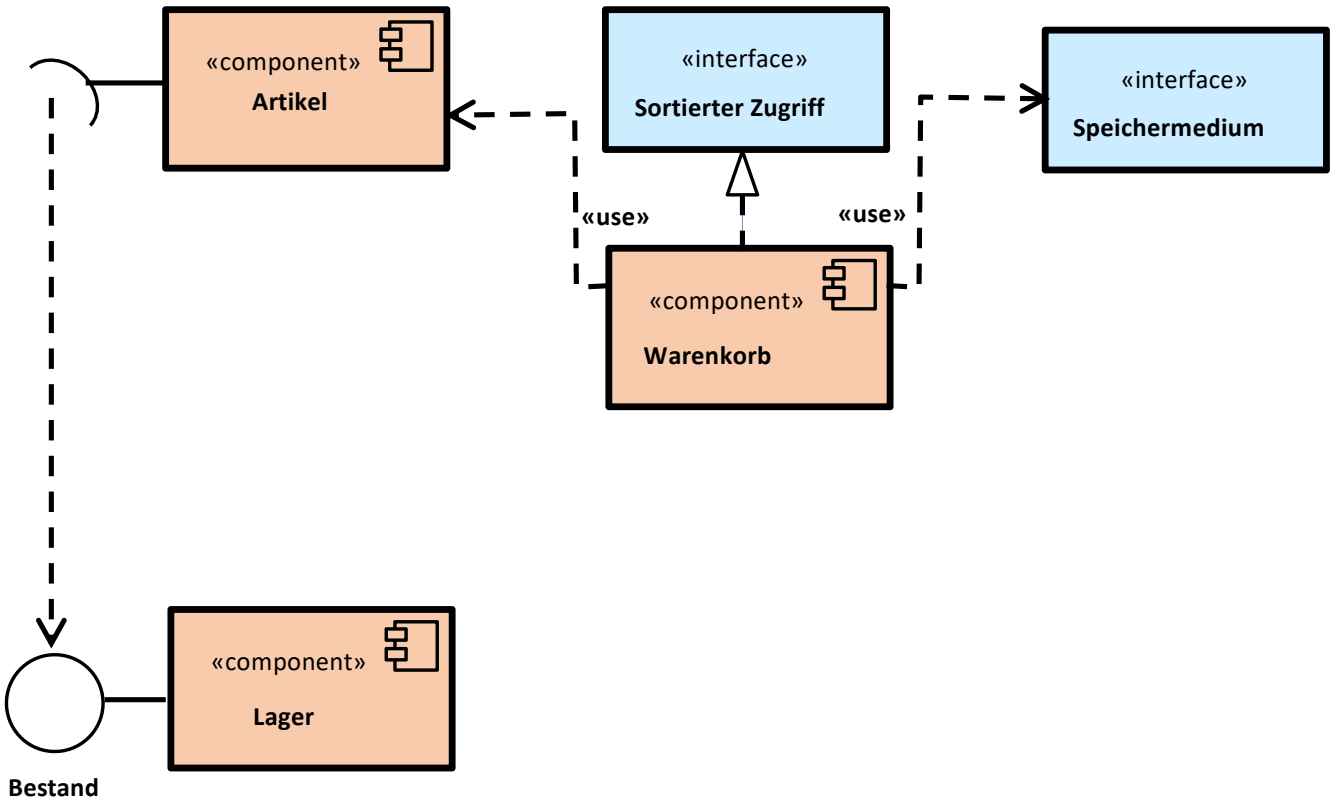
Komponentendiagramm

Beispiel: Warenkorb



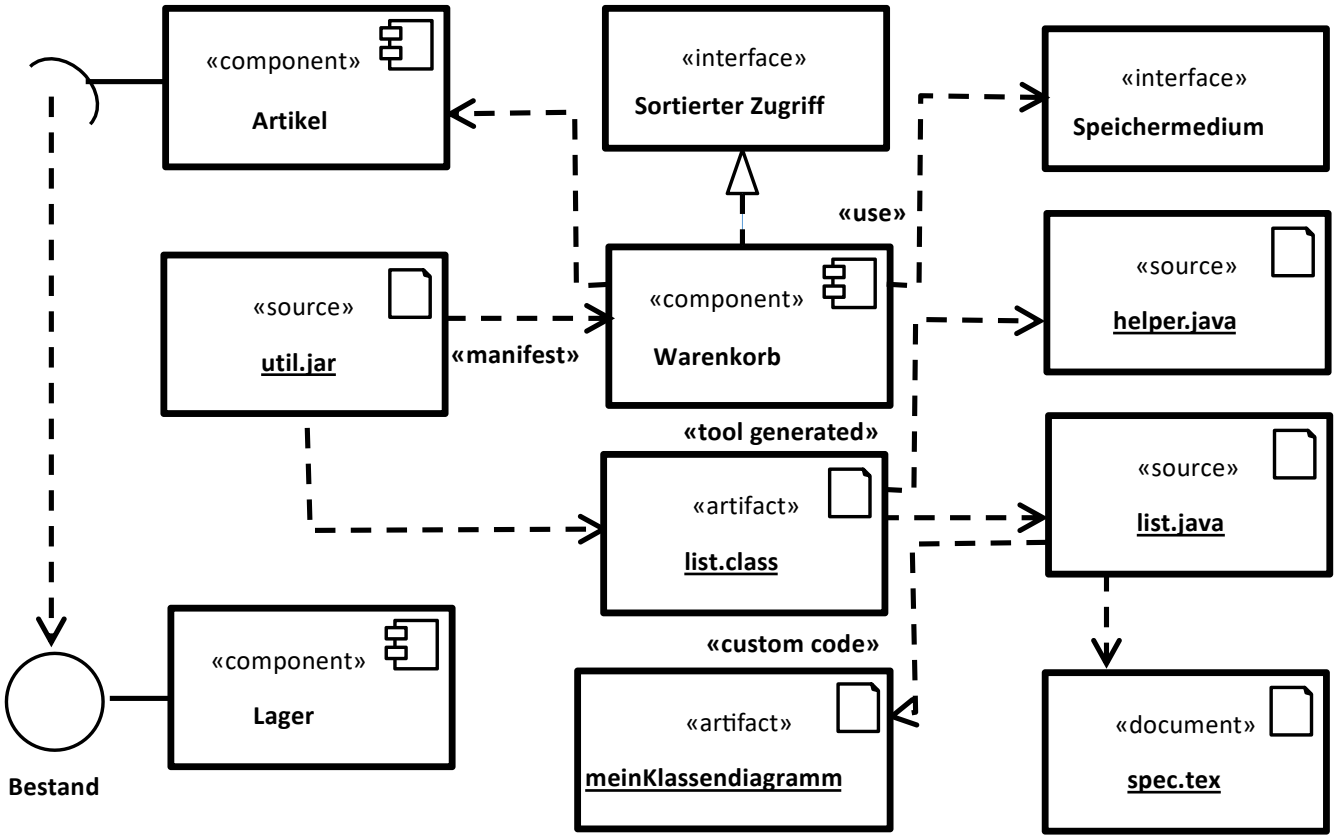
Komponentendiagramm

Beispiel: Warenkorb



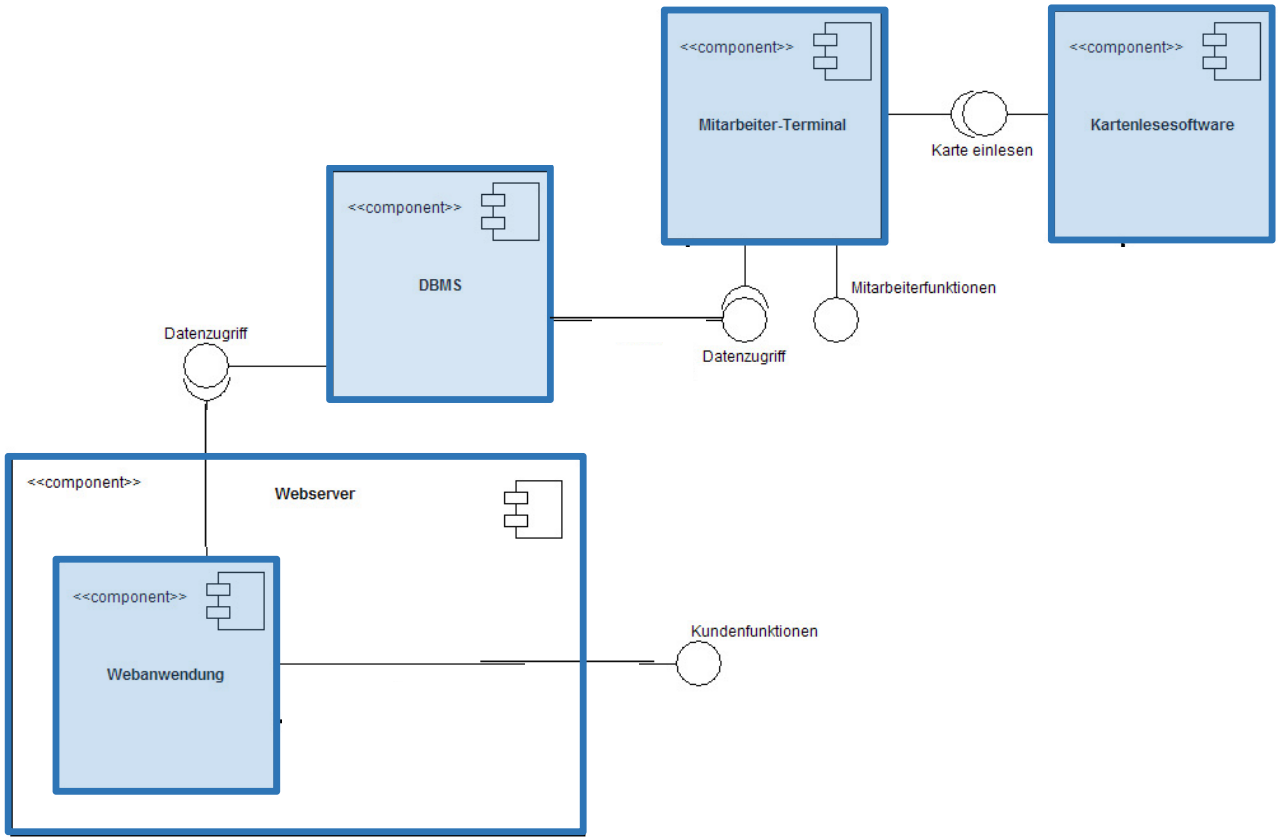
Komponentendiagramm

Beispiel: Warenkorb



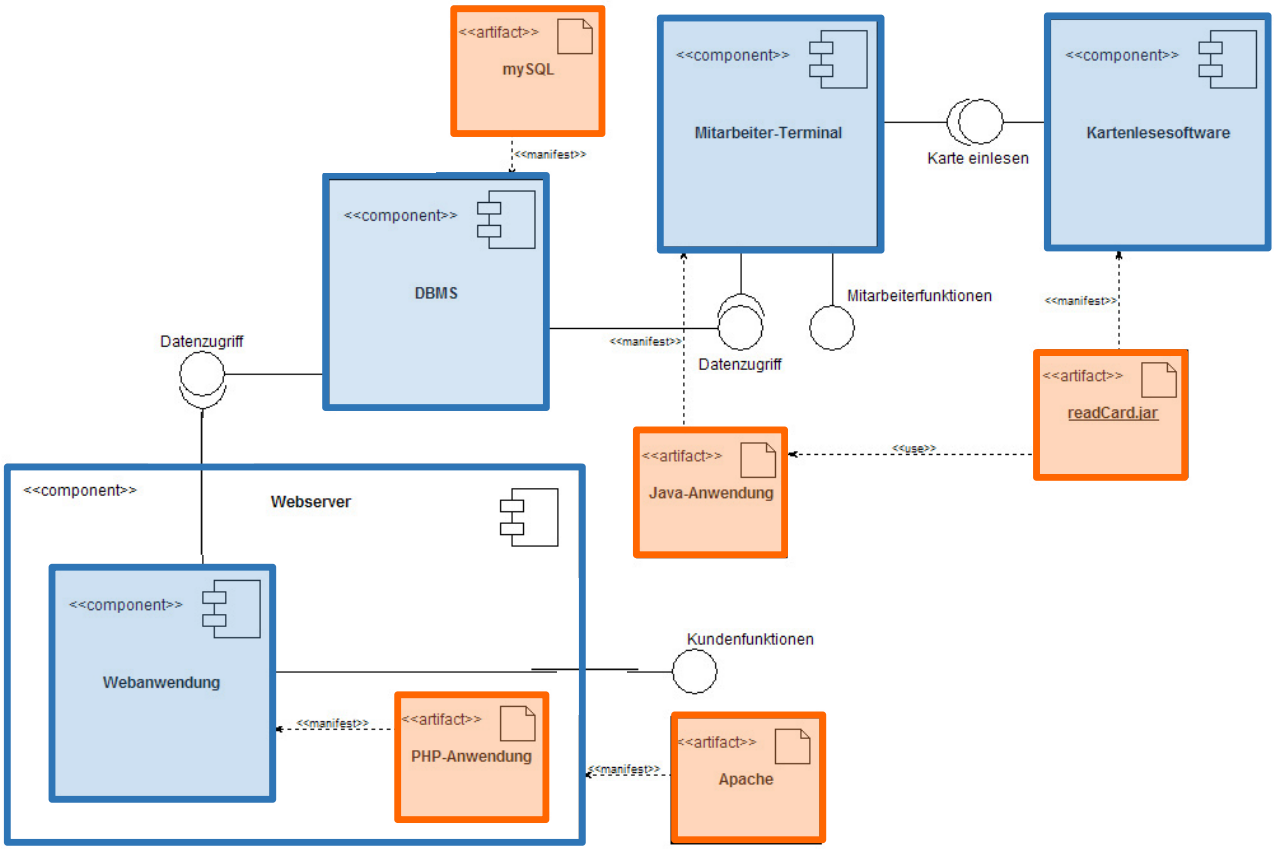
Komponentendiagramm

Beispiel: Realisierungsabhängigkeiten



Komponentendiagramm

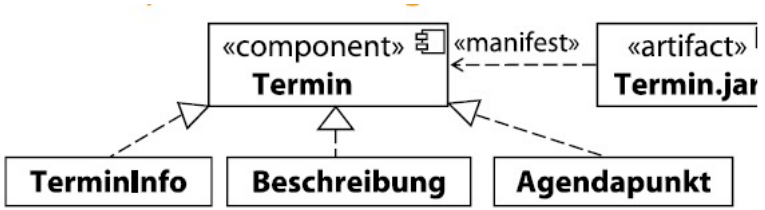
Beispiel: Realisierungsabhängigkeiten

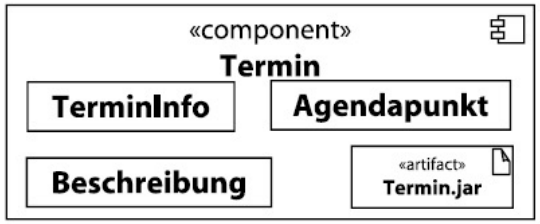


Notationsvarianten für Komponenten

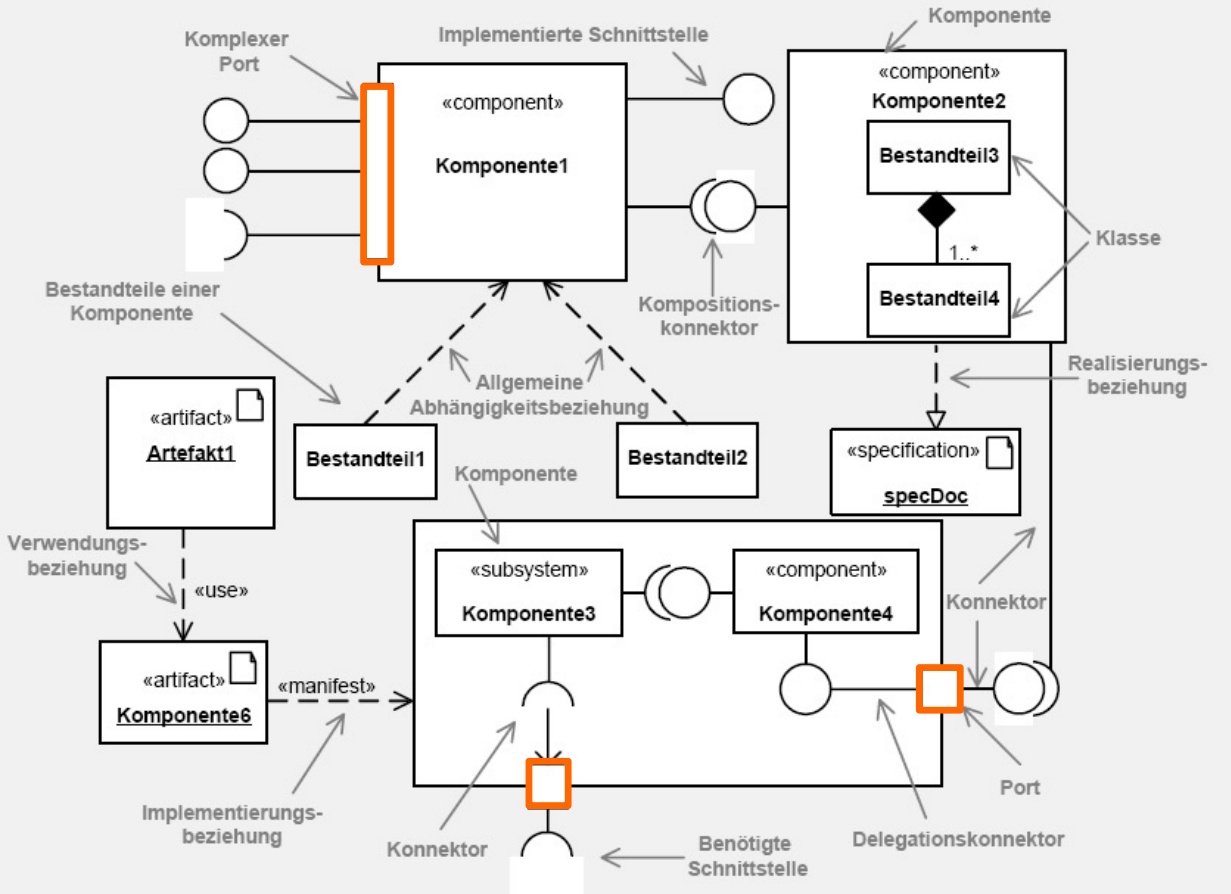
Notationsvarianten

- explizite Realisierungsabhängigkeiten
- direkte Schachtelung in einer Komponente
- eigene Abschnitte für Realisierung und Artefakte



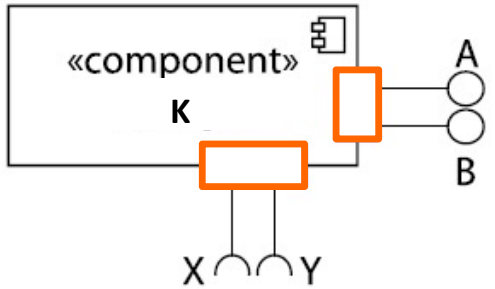


Elemente des Komponentendiagramms

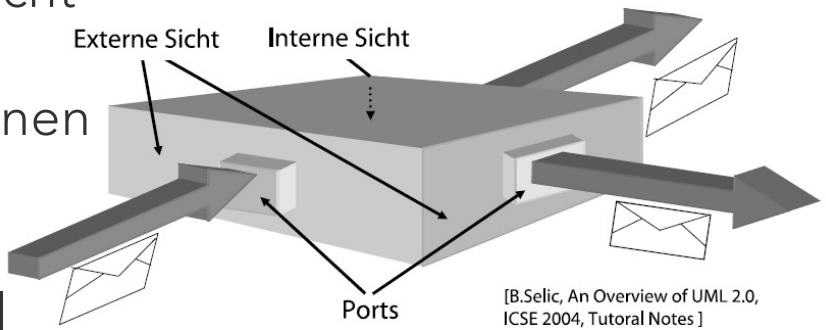


Port im Komponentendiagramm

- **gruppiert Schnittstellen** zu Diensten
  - meist aufgrund funktionaler Kohäsion

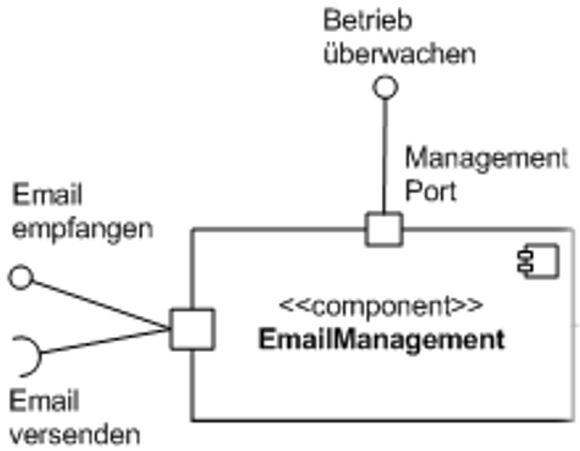


- **entkoppelt** die interne Sicht von der externen Sicht
  Plug & Play, d,h, eine Komponente
  - braucht keine anderen Komponenten zu kennen
  - muss keine Informationen über ihre Internas zur Verfügung stellen
  - kennt nur ihre Ports als strukturelles Merkmal



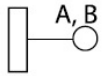
Port im Komponentendiagramm

- spezifiziert einen **expliziten Interaktionspunkt** einer Komponente
  - grobe Granularität erleichtert Verknüpfung und Austauschbarkeit
  - Prüfung von Typkonformität auf Basis von Ports
- ermöglicht die Definition von Protokollen zur Festlegung Interface-übergreifender Interaktionen
  - Einsatz von Protokollzustandsautomaten
- agiert als »**Verteilerzentrum**« für ein- und ausgehende Nachrichten
  - Verhaltensausführung in Abhängigkeit vom benutzten Port



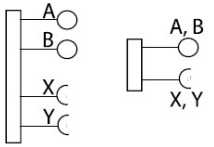
Port im Komponentendiagramm

- Notation
  - Port mit ausschließlich
    - angebotenen Interfaces



    - benötigten Interfaces

  - kombinierter Port



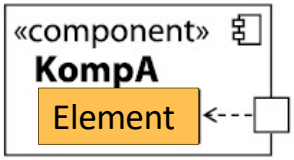
Port im Komponentendiagramm

- Port-Syntax

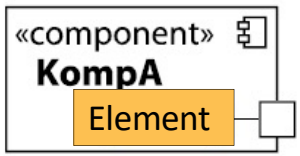
```
Portname:Typ
[Multiplizität]
{Eigenschaft}
```

```
audioOut: Klinke
[2]
{analog}
```

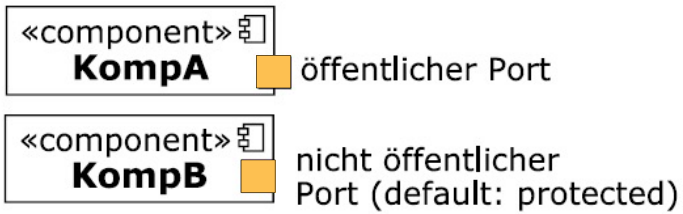
- Delegationsport
  - angefordertes Verhalten wird an realisierendes Element delegiert



- Verhaltensport
  - angefordertes Verhalten wird direkt von einer Instanz der Komponente ausgeführt



- Sichtbarkeit

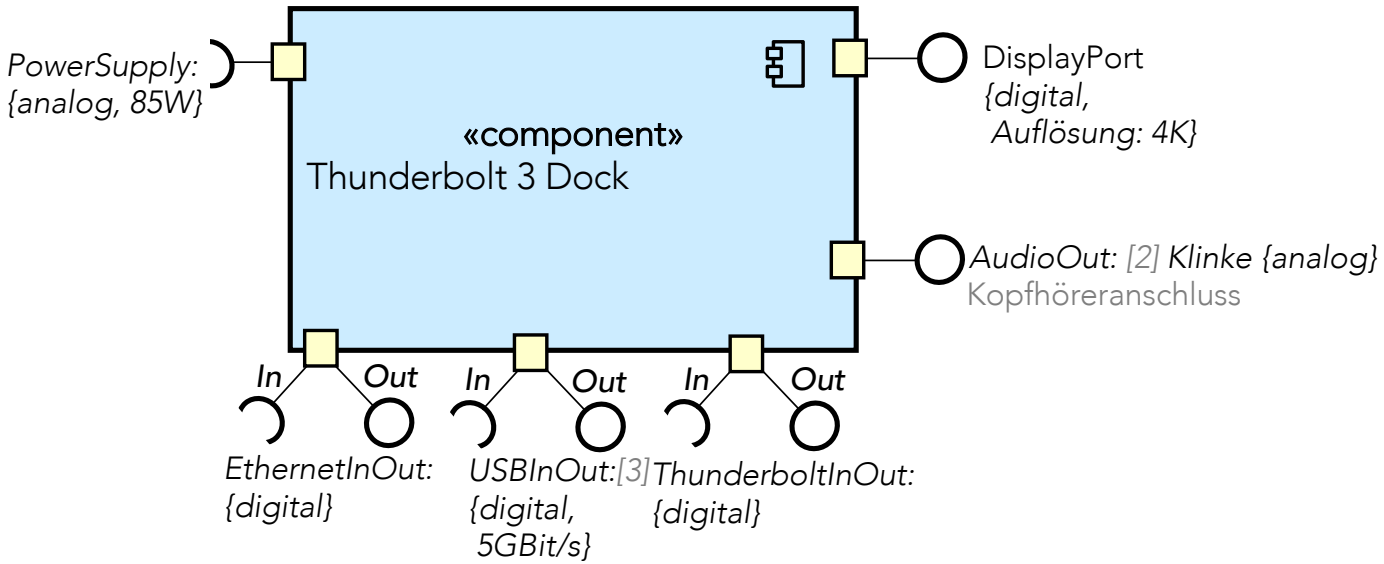


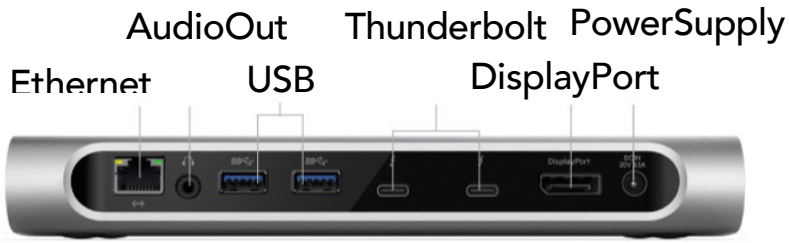
Port im Komponentendiagramm

- Implementierungsoptionen
  - Einfache Umsetzung, falls die Implementierungssprache das **Port-Konzept** unterstützt, wie bspw. bei WSDL
  - Realisierung als **eigene Klassen** - »first-class-objects«
  - **Keine Implementierung** - rein logisches Konstrukt

Beispiel:
Thunderbolt 3 Dock

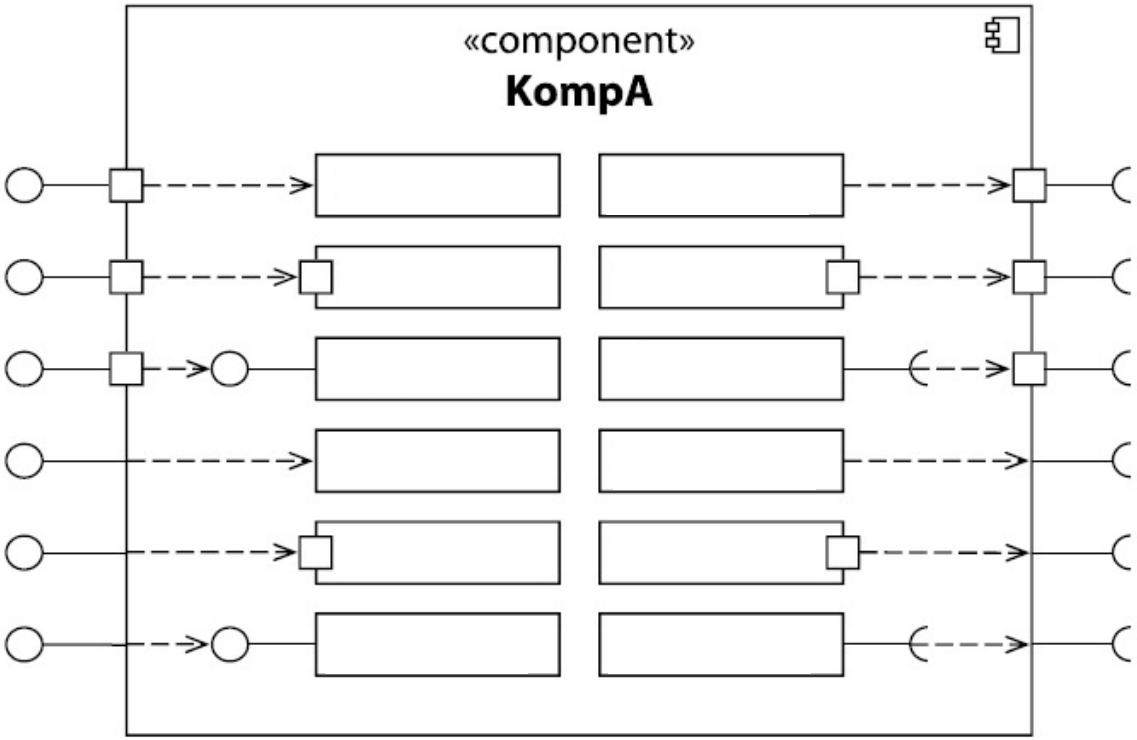
Port im Komponentendiagramm





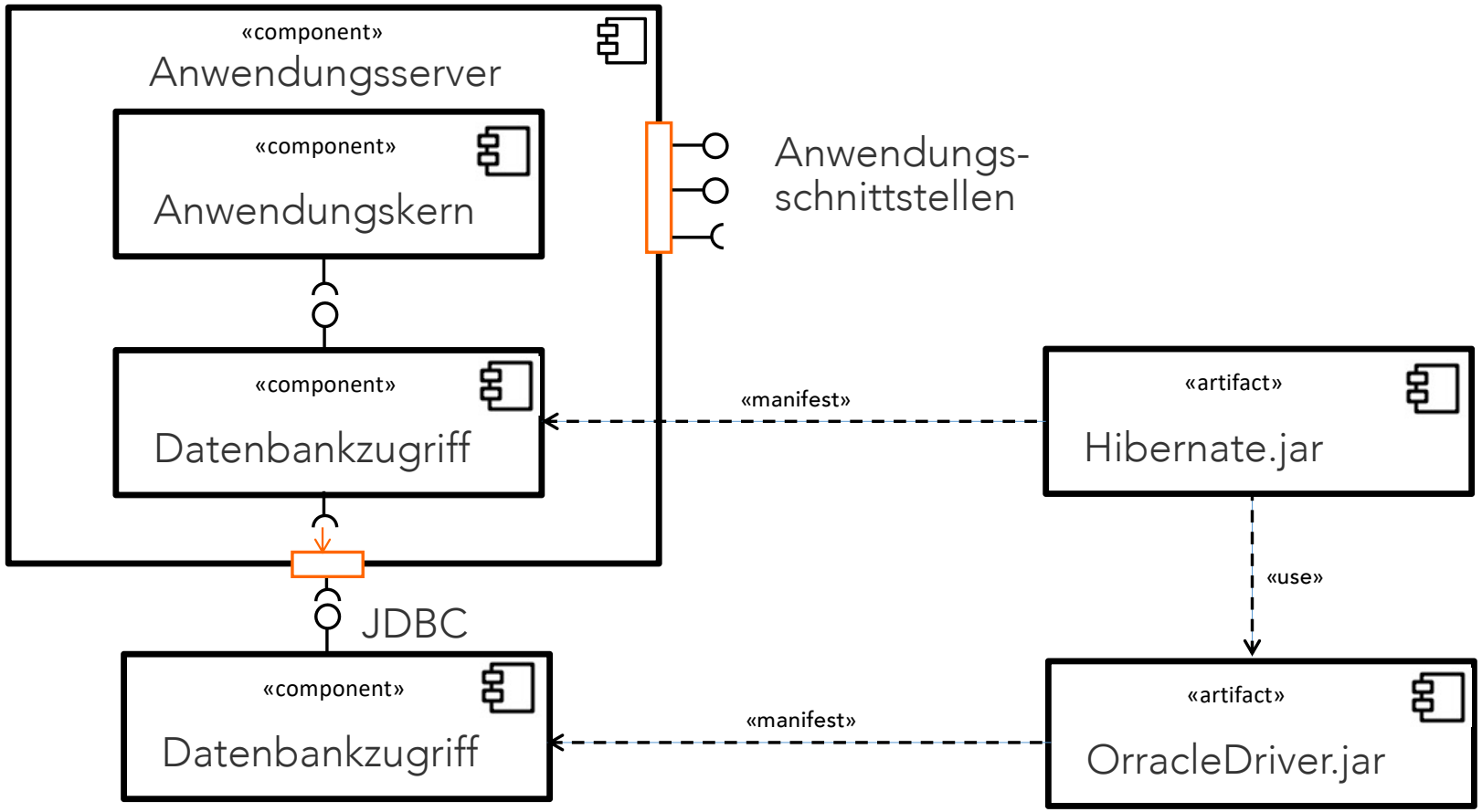
Port im Komponentendiagramm

Schnittstellenmodellierung von Komponenten



Komponentendiagramm

Beispiel: Anwendungsserver

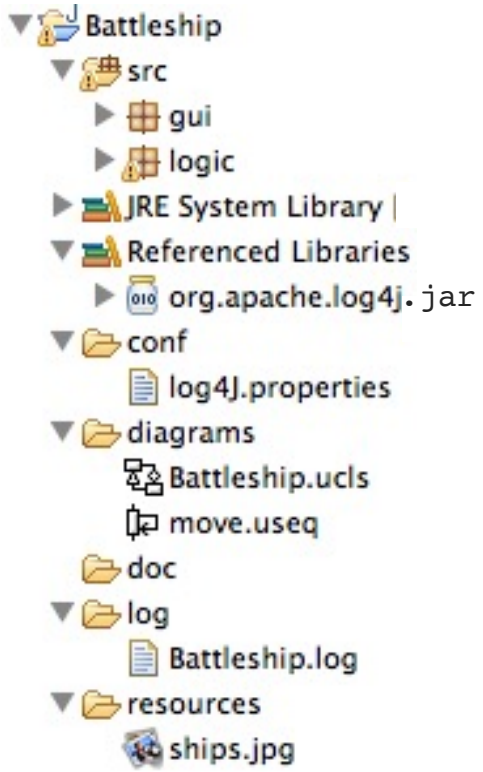


VL-Aufgabe: Komponentendiagramm

Sie analysieren das Eclipse-Projekt der Anwendung *SchiffeVersenken* (bzw. engl. *Battleship*).
Der Ordner *diagrams* enthält automatisch aus dem SourceCode generierte UML-Diagramme: ein Klassendiagramm *Battleships.ucls* und ein Sequenzdiagramm *move.useq*.

Wenn Sie sich das Projekt auf Filessystemebene anschauen, befindet sich dort zusätzlich ein *bin*-Verzeichnis.

Modellieren Sie auf Basis der vorhandenen Informationen ein Komponentendiagramm, welches darstellt, wie die verschiedenen Dateien voneinander abhängen.



Fachhochschule
Dortmund
University of Applied Sciences and Arts

we
focus
on
students

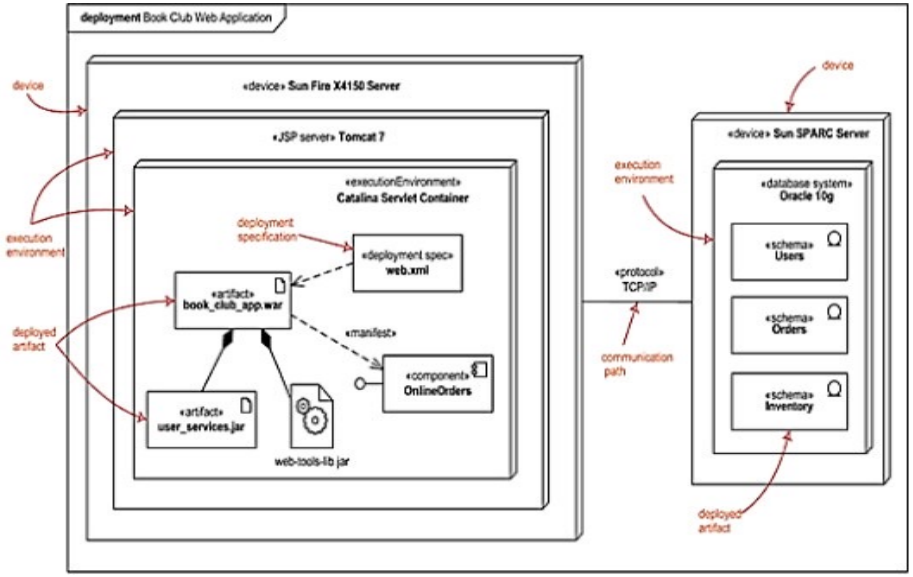
Struktur

Verteilungsdiagramm

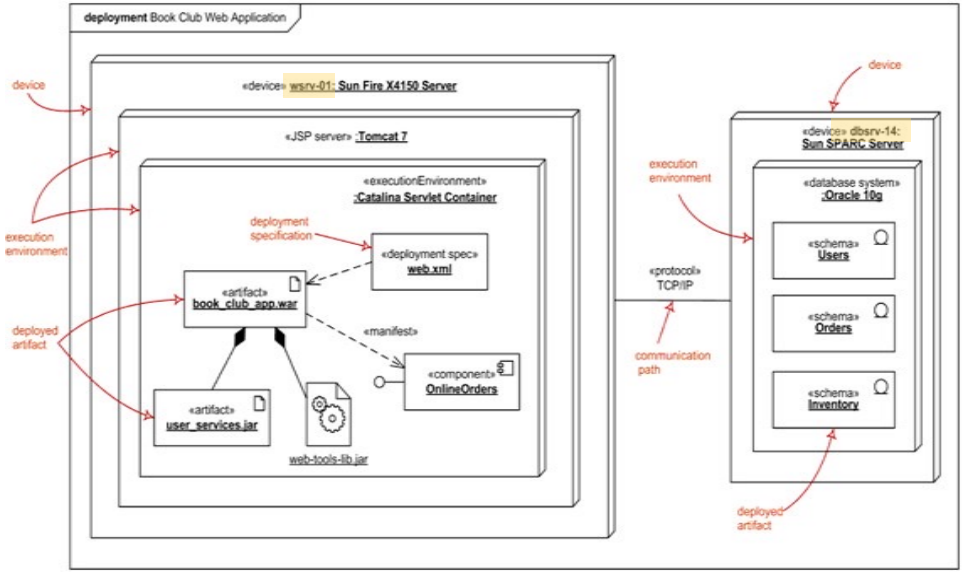
- Liefert **Antwort auf die Fragen**:
  a) „Auf welcher **Hardware läuft die Software,**
  b) auf welchem **Technologiestack und**
  c) über welche **Verbindungen** läuft die Kommunikation?“

- Wesentliche Modellierungselemente
  - Modellierung der eingesetzten **Hardware- und Softwaretopologie**
  - Modellierung des zugeordneten **»Laufzeitsystems«**
  - Verteilungsdiagramm kann auf **Typ- oder Instanzebene** modelliert werden

Verteilungsdiagramm



Typebene

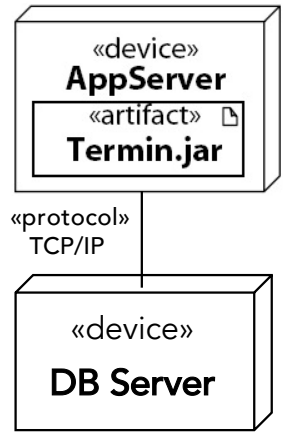


Instanzebene

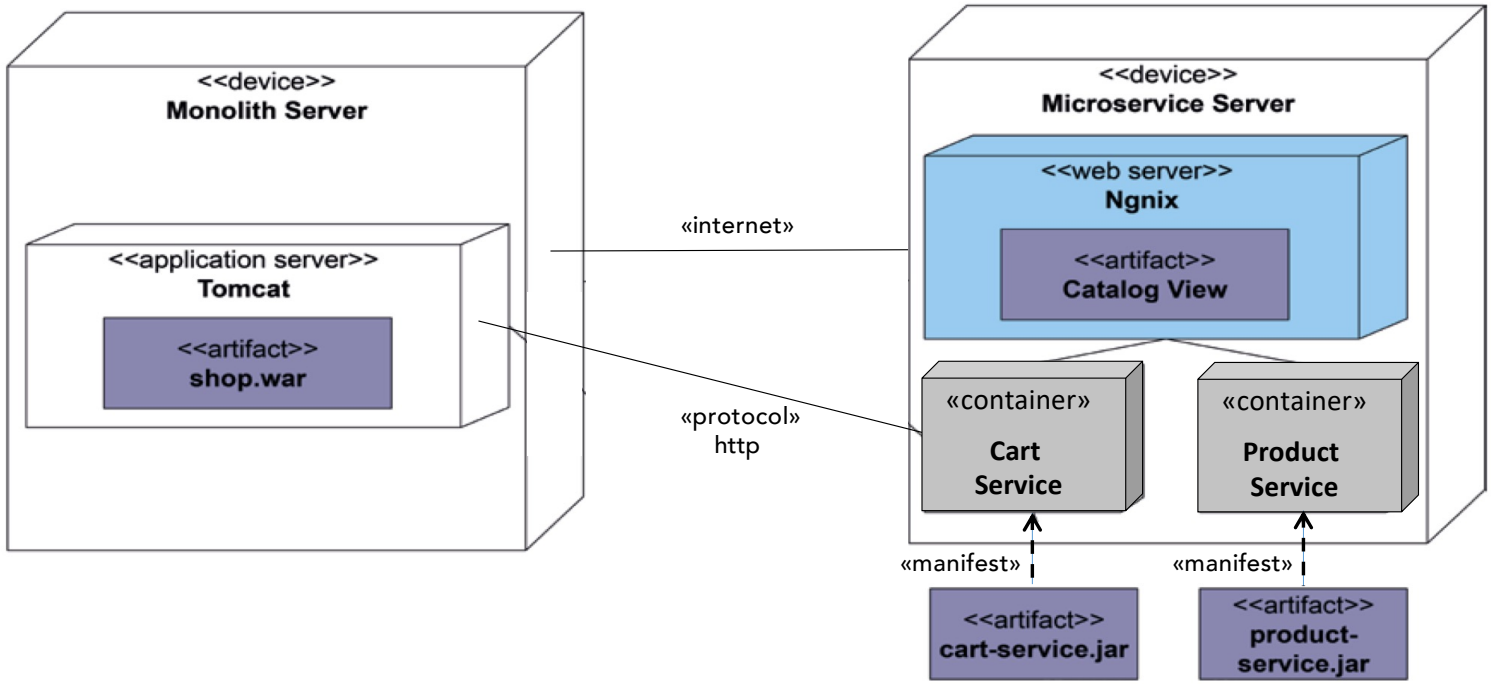
Die Modellierung auf Instanzebene kann beispielsweise verwendet werden, um Unterschiede zwischen Entwicklungs-, oder Produktionsumgebungen über den Namen bzw. die ID bestimmter Server anzuzeigen. Im obigen Beispiel wird die Webanwendung auf dem Anwendungsserver wsrv-01 und mehrere Datenbankschemata auf dem Datenbankserver dbsrv-14 bereitgestellt .

Verteilungsdiagramm

- Knoten
  repräsentieren **Geräte oder Ausführungsumgebungen**, auf die Artefakte verteilt werden
  - Gerät **«device»** repräsentiert Hardware und besitzt Rechenkapazität
  - Ausführungsumgebung **«execution env»** repräsentiert die notwendige Software, um das Laufzeitsystem auszuführen
  - **Hierarchische Knoten** können durch Schachtelung oder durch interne Strukturen erstellt werden Technologiestack
  - **Kommunikationsbeziehungen** sind spezielle Assoziationen und können durch **Stereotypen** näher definiert werden



Verteilungsdiagramm

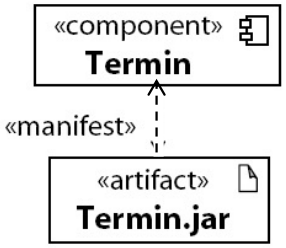


Serverbasierte Architekturbausteine

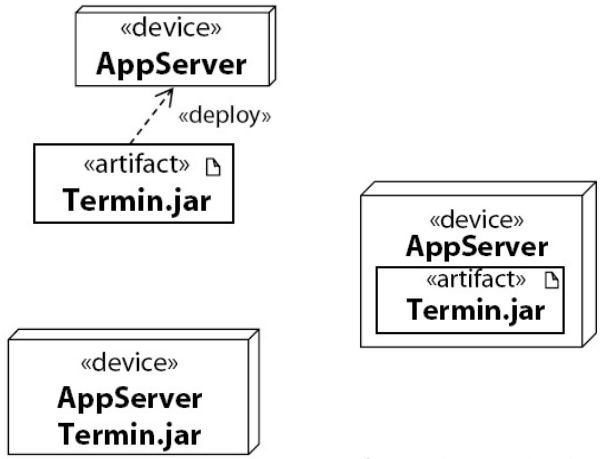
- Vielen Systeme sind nach dem Client/Server-Konzept strukturiert.

- Typische serverbasierte Architekturbausteine sind u.a.:
  - Webserver
  - Portalserver
  - Applikationsserver
  - Datenbankserver
  - Nachrichtenserver (Message Queue), Kommunikationshub
  - Dokumentations-Management-System (DMS)
  - Transaktionsserver
  - Verzeichnisserver (Directory)
  - Stapelverarbeitung (Batch Processing)
  - Public Key Infrastructure (PKI)

Artefakte und ihre Verteilung



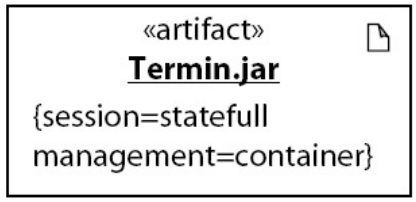
- Artefakte stellen die **»Objekte« der Verteilung** dar
- unter Verteilung wird die **Zuweisung von Artefakten** an Knoten verstanden
  - ein Artefakt kann auf mehrere Knoten verteilt werden
- **3 Notationsvarianten**
  - explizite Modellierung als Abhängigkeit mit Schlüsselwort «deploy»
  - grafische Schachtelung von Artefakten auf Knoten
  - Notation der Artefaktnamen im Knotensymbol

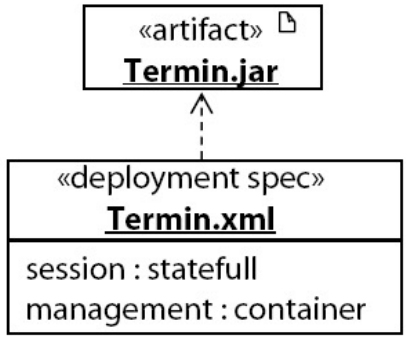


Einsatzspezifikation

- Verteilungsdetails eines Artefakts durch Parameter in **Einsatzspezifikation** (deployment specification) definierbar
  - z.B. Modellierung von Konfigurationsdateien
  - für eine Verteilung können verschiedene Einsatzspezifikationen definiert werden

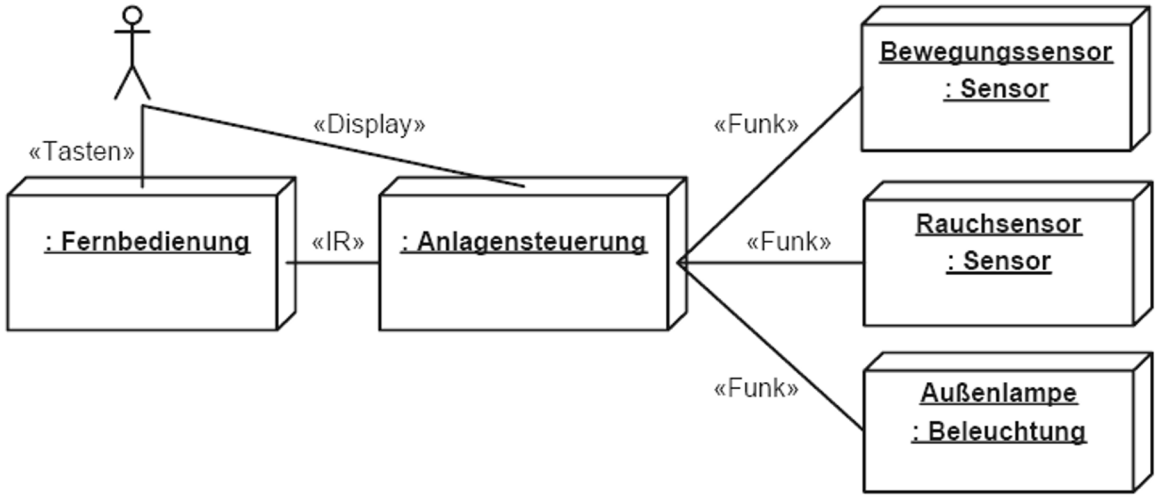
- Notationsalternativen





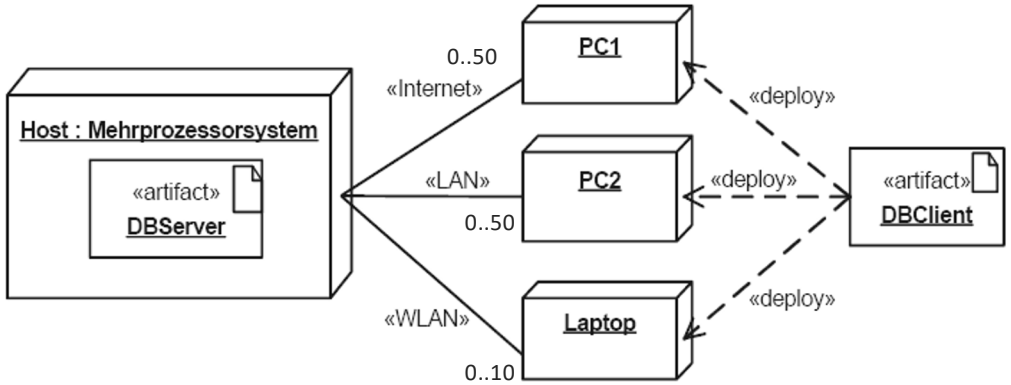
Sonstiger Einsatz

- **Physikalische Kontextabgrenzung** des Systems, wenn Nachbarsysteme existent
  - Blackbox-Modellierung und Darstellung von System und Nachbarsystem

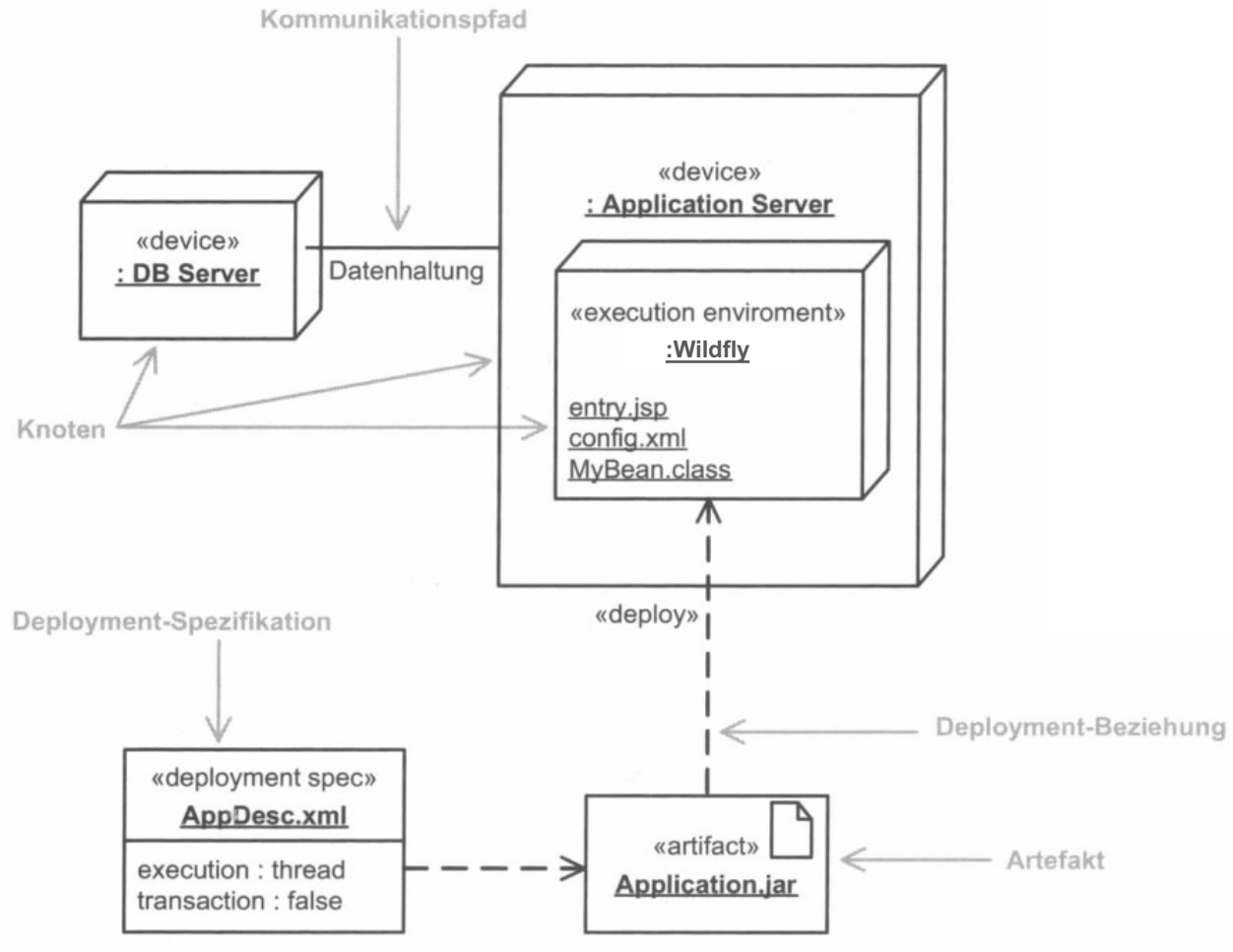


Sonstiger Einsatz

- **Dokumentation von Hardwarevorgaben** in visualisierter Form ist besser als rein textuelle Darstellung



Verteilungsdiagramm – Syntax im Überblick



Fachhochschule Dortmund
University of Applied Sciences and Arts

VL-Aufgabe: Verteilungsdiagramm

we focus on students

Modellieren Sie das folgende Verteilungsdiagramm (Instanzebene):

- Das System erfordert verschiedene Geräte: ein Handy, einen Anwendungsserver und einen Datenbankserver. Anwendungsserver und Datenbankserver kommunizieren über Ethernet, während Handy und Anwendungsserver über Internet, konkret JSON über http, kommunizieren. Die drei Hardwareknoten werden nun detailliert beschrieben.

- **Handy:** Bei dem Handy handelt es sich um ein Smartphone vom Typ „Nexus S“. Auf diesem muss Android in der Version 2.2 installiert sein. Zur Laufzeit läuft auf dem Handy eine App als ausführbare .apk-Datei namens Uno.apk in der Dalvik VM (Anm. virtuelle Maschine unter Android).

- **Anwendungsserver:** Die Hardware dieses Servers ist ein „HP ProLiant ML350 GH5“-Rechner, der unter dem Namen „SP002“ angesprochen werden kann , auf dem als Betriebssystem „openSUSE“ in der Version 11.0 installiert ist. Als Applikationsserver läuft auf dem Betriebssystem ein „JBoss Application Server 7“.
- Der JBoss enhält als Laufzeitumgebung zwei Container: einen „Web Container“ und
- einen „Component Container“. Im Web Container können die Dateien: „administration.jsp“ und „serverConfig.jsp“ ausgeführt werden. Im Componenten Container können die Dateien „playerPortal.jar“, „gameManagement.jar“, „uno.jar“ laufen.

- **Datenbankserver:** Dieser Server kann im Netz unter dem Namen „DBServer“ angesprochen werden. Auf ihm ist ein MYSQL-Server installiert.

Struktur

Weitere Fragen