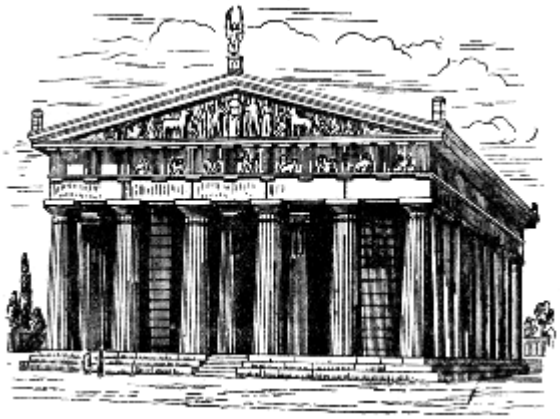
Softwarearchitektur

(Architektur: *αρχή* = Anfang, Ursprung + *tectum* = Haus, Dach)

4. Sichten bei der Softwarearchitektur – Teil 1



Vorlesung Wintersemester 2010 / 2011

Technische Universität München
Institut für Informatik
Lehrstuhl von Prof. Dr. Manfred Broy

Dr. Klaus Bergner, Prof. Dr. Manfred Broy, Dr. Marc Sihling

Inhalt

- Motivation und Eigenschaften
  - Was sind Sichten?
  - Überblick über die Sichten
- Fachliche Sicht
  - Vom Analysemodell zum fachlichen Entwurf
  - Bildung fachlicher Komponenten
- Technische Sicht
  - Technische Komponenten und Schnittstellen
  - Umsetzung der Fachlichkeit
- Verteilungssicht
  - Verteilung der Komponenten
- Zusammenfassung

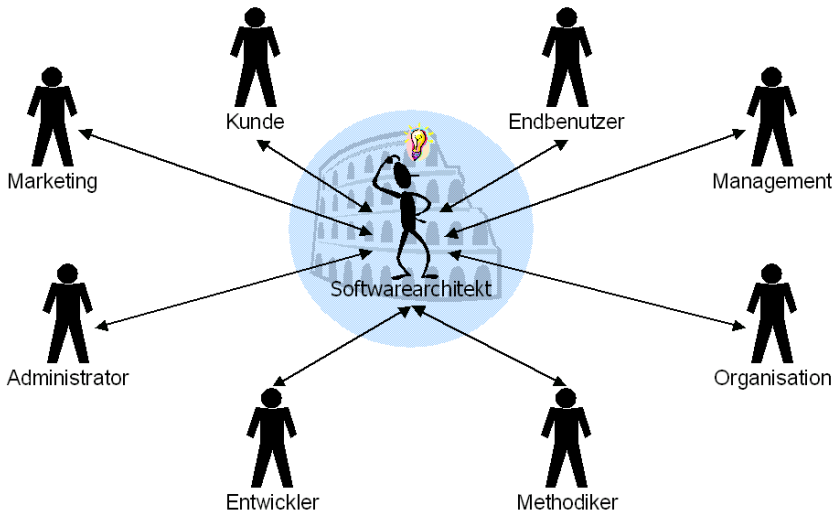
Inhalt

- Motivation und Eigenschaften
  - Was sind Sichten?
  - Überblick über die Sichten
- Fachliche Sicht
  - Vom Analysemodell zum fachlichen Entwurf
  - Bildung fachlicher Komponenten
- Technische Sicht
  - Technische Komponenten und Schnittstellen
  - Umsetzung der Fachlichkeit
- Verteilungssicht
  - Verteilung der Komponenten
- Zusammenfassung

Warum Sichten?

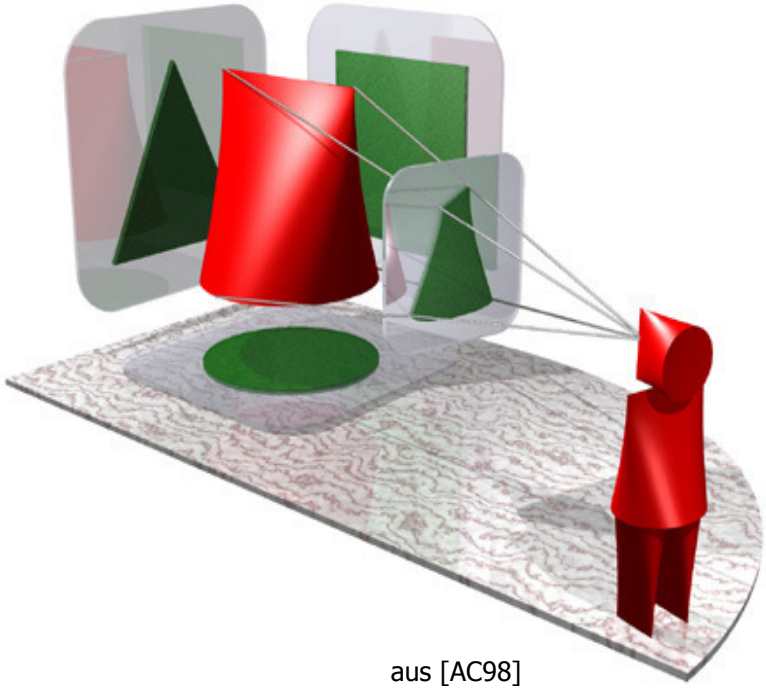
- Komplexe Systeme lassen sich nicht übersichtlich mit einer einzigen Beschreibung erfassen.
- Unterschiedliche Rollen im Bezug auf das System arbeiten mit unterschiedlichen Informationen.
- Nötig sind geeignete Architekturbeschreibungen als
  - Kommunikations- und Diskussionsplattform
  - Plan für Entwurf und Implementierung

  für unterschiedliche Zielgruppen und Aufgaben.



Was sind Sichten?

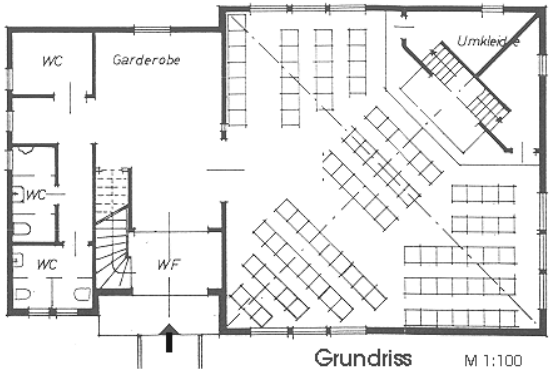
- Ein System wird aus mehreren Blickwinkeln betrachtet, die jeweils unterschiedliche Schwerpunkte haben.
- Je nach Blickwinkel treten bestimmte Eigenschaften in den Vordergrund, andere werden nur grob oder gar nicht dargestellt.
- Eine Beschreibung für einen definierten Blickwinkel heißt Sicht.
- IEEE STD 1471-2000: A view is a representation of a whole system from the perspective of a related set of concerns.

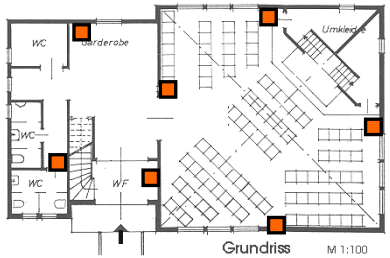


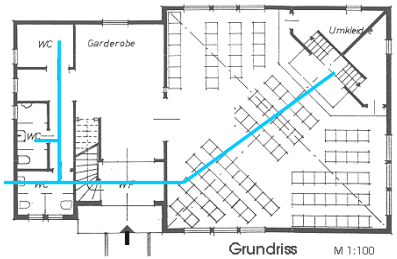
aus [AC98]

Beispiel: Baupläne

- Bei einem Gebäude sind viele unterschiedliche Sichten zu berücksichtigen:
  - Grundriss
  - Aufrisse
  - Traglasten
  - Elektroinstallationsplan
  - Installationsplan
  - Fluchtwegeplan
  - etc.
- Jede ist für bestimmte Aufgaben und Rollen zugeschnitten.







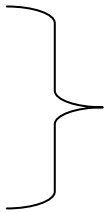
Eigenschaften von Sichten

- Eine Sicht stellt jeweils bestimmte Eigenschaften eines Systems dar.
- Sichten sind typischerweise nicht völlig unabhängig voneinander – bestimmte Informationen aus einer Sicht können sich in anderen Sichten wiederfinden.
- Widersprechen sich zwei Sichten nicht (und beschreiben damit ein mögliches System), so heißen sie konsistent miteinander.
- Für die einzelnen Sichten können jeweils geeignete Beschreibungstechniken verwendet werden.
- Eine Beschreibungstechnik kann aber auch für unterschiedliche Sichten verwendet werden.

Arten von Sichten auf die Architektur eines Systems

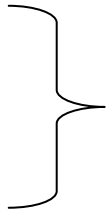
- Wir unterscheiden in der Folge folgende Sichten:

- Spezifikationssichten
  - Fachliche Sicht
  - Technische Sicht
  - Verteilungssicht



Thema der heutigen Doppelstunde

- Erstellungssichten
  - Entwicklungssicht
  - Testsicht
  - Bereitstellungssicht



Thema der nächsten Doppelstunde

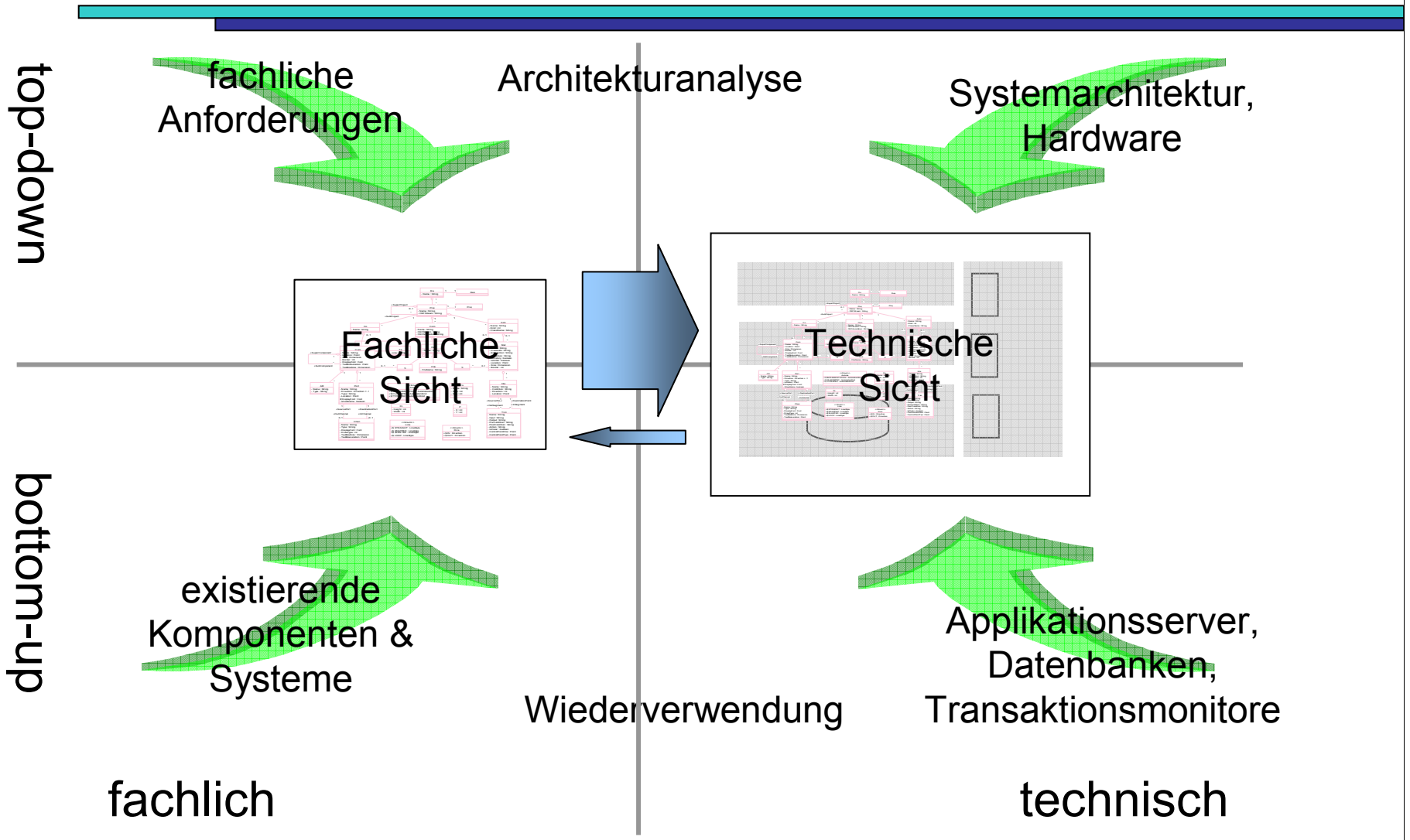
Fachliche Sicht - Überblick

- Zweck
  - Darstellung der Konzepte des Anwendungsgebiets möglichst unabhängig von einer softwaretechnischen oder plattform-spezifischen Lösung
- Inhalte
  - Fachliche Komponenten inklusive Schnittstellen und Verhalten
  - Beziehungen zwischen fachlichen Komponenten
  - Systemgrenze und Anbindung existierender fachlicher Komponenten
- Beschreibung
  - Überblick möglich durch CRC-Karten, Box-and-Line-Diagramme
  - objektorientierte Modellierungstechniken (UML + Erweiterungen)
  - in Spezialfällen formale Techniken (OCL, ADLs, DSLs etc.)

Technische Sicht - Überblick

- Zweck
  - Darstellung der softwaretechnischen Lösung für das System inklusive der Umsetzung der fachlichen Sicht
- Inhalte
  - Technische Komponenten des Systems und deren Schnittstellen
  - Beziehungen zwischen den Komponenten, inklusive der Abbildung der fachlichen Komponenten in die technische Sicht
  - Beziehung zu Basissoftware und Abbildung auf die Hardware
- Beschreibung
  - Überblick möglich durch Box-and-Line-Diagramme
  - objektorientierte Modellierungstechniken (UML + Erweiterungen)
  - Interface Definition Languages (IDLs)
  - Quellcode-Fragmente für Details von Mechanismen
  - in Spezialfällen formale Techniken (OCL, ADLs, DSLs etc.)

Fachliche und Technische Sicht



Trennung von fachlicher und technischer Sicht: Vorteile

- Entwurf, Realisierung und Validierung der Fachlichkeit werden vereinfacht
  - technische Details müssen nicht betrachtet werden
  - wesentliche Reduzierung der Komplexität
  - Einbeziehung der Anwender wird möglich
- Möglichkeit, die fachliche Sicht auf unterschiedliche technische Architekturen abzubilden
  - fachliches Verhalten bleibt bei allen Realisierungen gleich
  - Evolution und Optimierungen der technischen Architektur unabhängig von der Fachlichkeit möglich
  - langlebige, wertvolle Geschäftslogik kann unabhängig von speziellen Techniken leben und weiterentwickelt werden
  - Integration unterschiedlicher Systeme vereinfacht

MDA – Model-Driven Architecture

- Ansatz der Object Management Group (OMG) zur Modellierung bei der Software-Entwicklung
- Unterscheidet drei aufeinander aufbauende Sichten
  - Computation-Independent Model (CIM) = Analysemodell
    - Anforderungen und Umgebung eines Systems
    - Abstrahiert von Struktur und Informationsverarbeitung
  - Platform-Independent Model (PIM) = Fachliche Architektur
    - Spezifiziert Struktur und Informationsverarbeitung
    - Enthält nur die Anteile, die für alle Plattformen gleich sind
  - Platform-Dependent Model (PSM) = Technische Architektur
    - Spezifiziert konkrete Mechanismen für spezielle Plattform
- Entwicklung erfolgt mittels Modelltransformationen
- Teile von MDA sind standardisiert: MOF, XMI, UML, QVT

Inhalt

- Motivation und Eigenschaften
  - Was sind Sichten?
  - Überblick über die Sichten
- Fachliche Sicht
  - Vom Analysemodell zum fachlichen Entwurf
  - Bildung fachlicher Komponenten
- Technische Sicht
  - Technische Komponenten und Schnittstellen
  - Umsetzung der Fachlichkeit
- Verteilungssicht
  - Verteilung der Komponenten
- Zusammenfassung

Fragestellungen in der fachlichen Sicht

1. Wie wird die fachliche Funktionalität repräsentiert?
   - Welche Operationen gibt es und wo hängen sie?
   - Wie sind die fachlichen Abläufe?
2. Wie sind die Schnittstellen der fachlichen Komponenten?
   - Welche Operationen sind nach außen zugänglich?
3. Wie lassen sich fachliche Komponenten bilden?
   - Wann werden Komponenten gebildet?
   - Wie erfolgen Zugriff auf und Verwaltung von Komponenten?
   - Welche Beziehungen haben die Komponenten untereinander?
   - Wie lassen sich Komponenten voneinander entkoppeln?

Vom Analysemodell zum fachlichen Entwurf

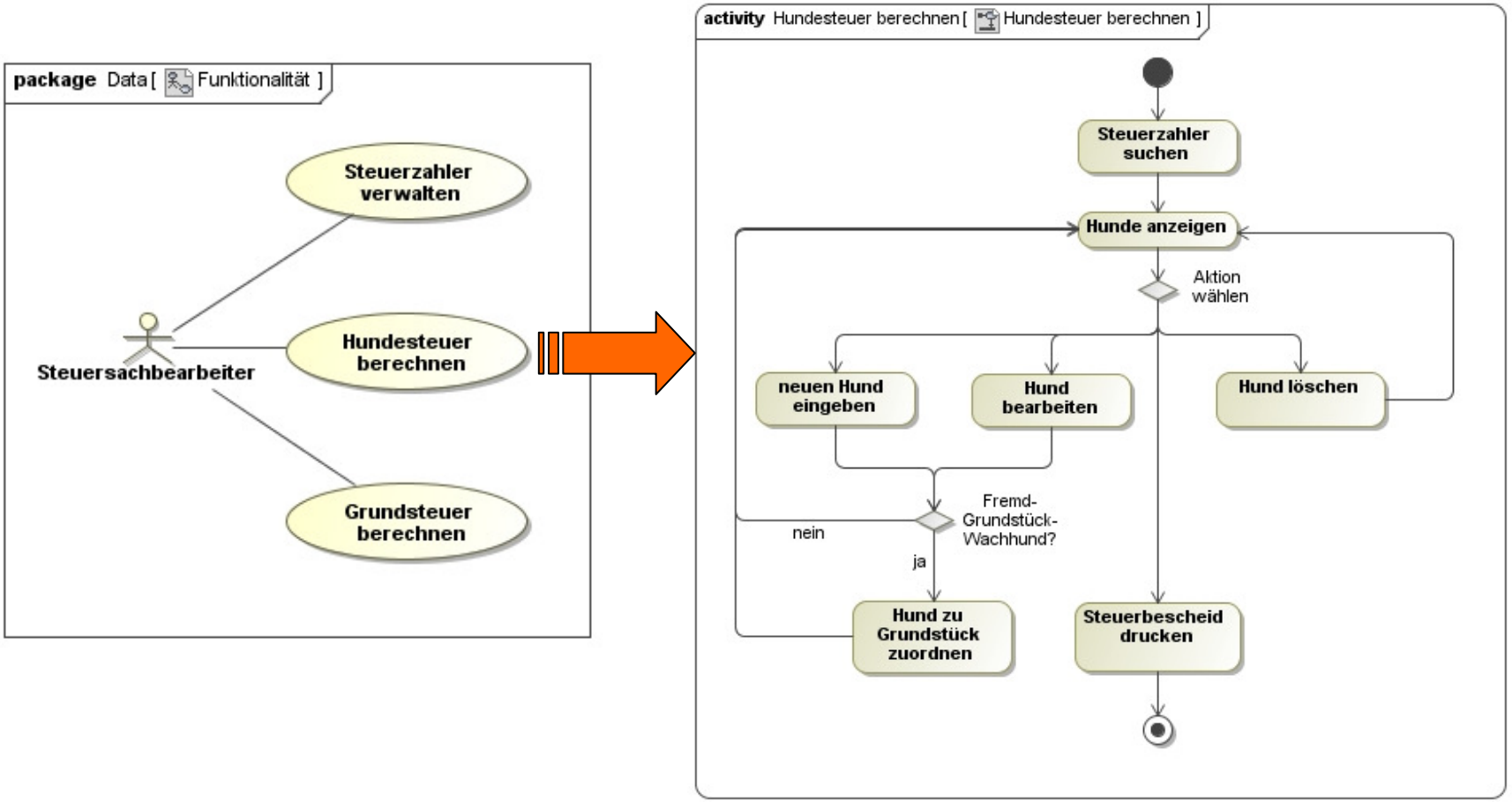
- Die genannten Fragestellungen werden in einem iterativen Prozess beantwortet:
  - Start mit erstem fachlichen Analyse-Modell basierend auf den Anforderungen (meist in Form eines Klassendiagramms mit den Daten des Systems und zusätzlicher Anwendungsfälle und Aktivitäten).
  - Schrittweise Verfeinerung des Modells durch
    - Hinzufügen von Operationen
    - Schnittstellenbildung
    - Komponentenbildung
- Das entstehende Modell ist plattformunabhängig, zielt aber typischerweise bereits auf einen bestimmten Architekturstil ab.
- Dazu enthält es bereits Konzepte, die seine Abbildung auf die Technik erleichtern.
- OMG-Terminologie: CIM-to-PIM-Mapping.

Anwendungsbeispiel: eSteuer 2010

- Neues Informationssystem „eSteuer 2010“ für die Berechnung neuer Steuern in Finanzämtern
- Steuern
  - Grundsteuer
  - Hundesteuer
  - leicht erweiterbar für beliebige neue Steuerarten
  - Wegfall von Steuerarten soll grundsätzlich ebenfalls möglich sein
- Funktionalität und Anwendungsfälle von eSteuer 2010
  - Steuerzahler verwalten
  - Grundsteuerinformationen verwalten, Grundsteuer berechnen
  - Hundesteuerinformationen verwalten, Hundesteuer berechnen
  - Steuerermäßigungen für Wachhunde auf Fremdgrundstücken beachten!
- Anwender des Systems sind Finanzbeamte

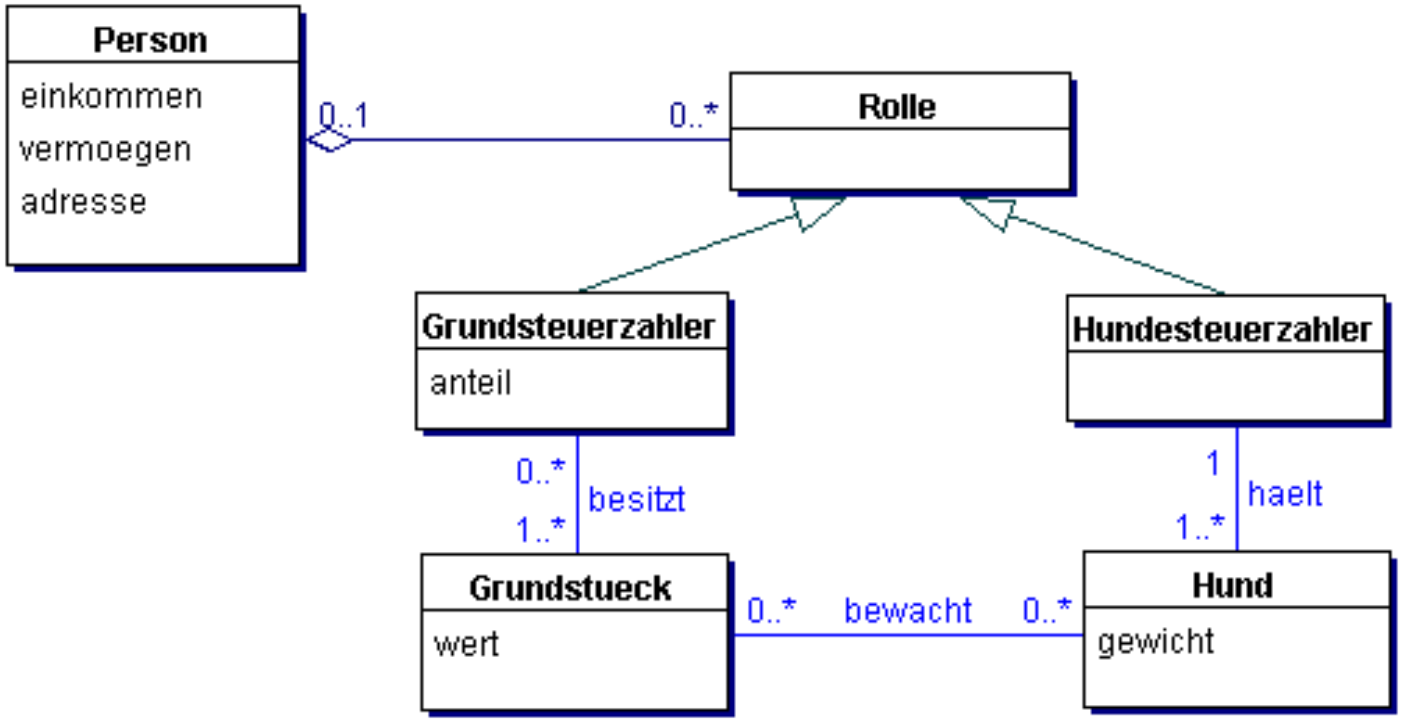
Ergebnis der Anforderungsanalyse - Funktionalität

- Anwendungsfälle und Ablaufspezifikationen



Ergebnis der Anforderungsanalyse - Daten

- Fachliches Datenmodell in Form eines UML-Klassendiagramms

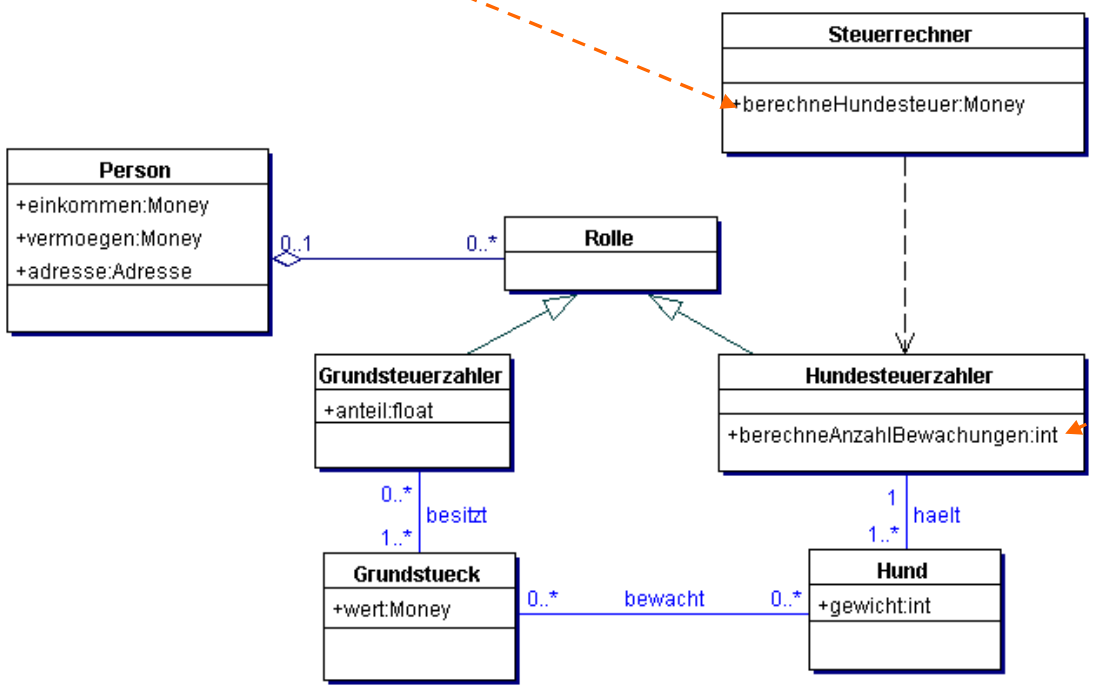


Zuordnung der Funktionalität

- In der fachlichen Sicht wird die Funktionalität in Form von Operationen der modellierten Klassen dargestellt.
- Hier gibt es im Wesentlichen zwei Möglichkeiten:
  1. Operationen können direkt an den Daten-Objekten (Entity-Klassen) hängen.
  2. Es können eigene Control-Klassen für Operationen gebildet werden.
- Beobachtung: Daten sind meist stabiler als Funktionalität.
  - Als Strategie ist deshalb sinnvoll, instabile Funktionalität eher auszulagern und nur stabile Basisfunktionalität bei den Entity-Klassen zu belassen.

Zuordnung der Funktionalität im Beispiel

`berechneHundesteuer` ist eine instabile Operation (das Berechnungsschema kann sich ändern) und wird deshalb einer neuen Control-Klasse `Steuerrechner` zugeordnet



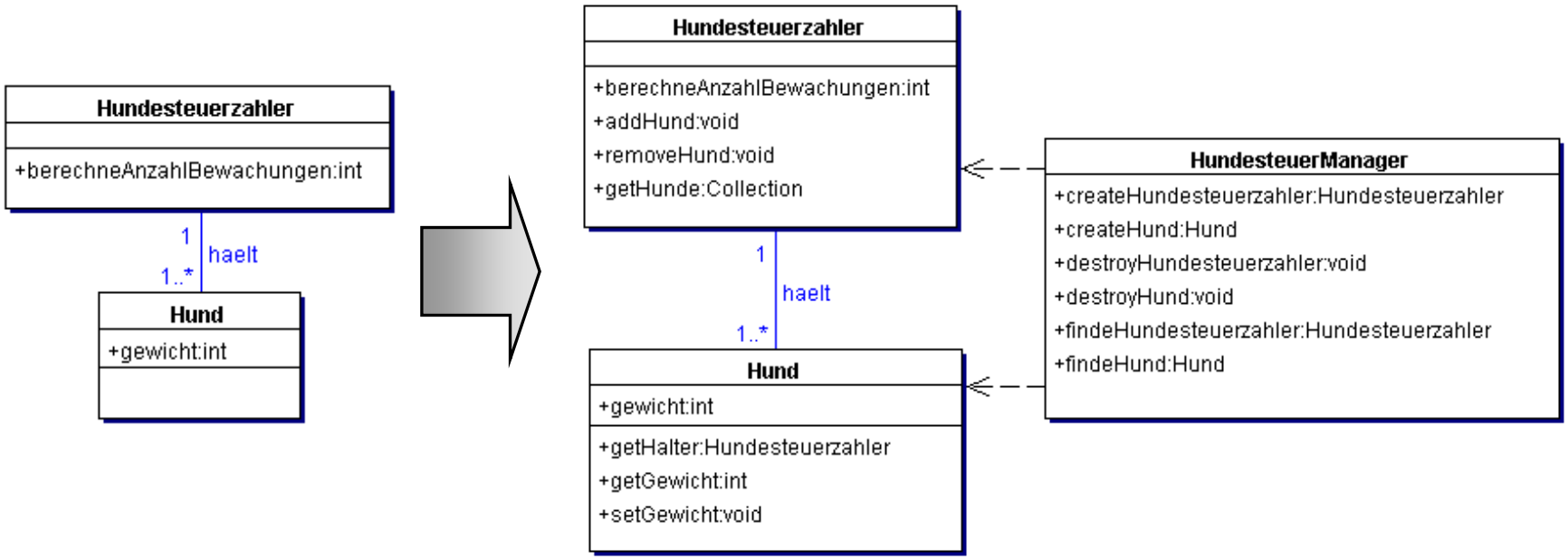
`berechneAnzahlBewachungen` ist eine stabile Operation und wird deshalb der bestehenden Entity-Klasse `Hundesteuerzahler` zugeordnet

Navigations- und Verwaltungsoperationen

- Neben den Operationen für die Aktionen der Anwendungsfälle gibt es auch Operationen, die sich aus der Umsetzung des Klassendiagramms ergeben.
- Zugriff auf die Daten
  - typischerweise über die Einführung von get- und set-Operationen für die Attribute
  - Beispiel: Operationen `getGewicht():int` und `setGewicht(int):void` der Klasse `Hund`
- Verwaltung der Beziehungen zwischen Klassen
  - typischerweise über die Einführung von Navigations- und Verwaltungsmethoden
  - Beispiel: Operationen `getHunde():Collection`, `addHund(Hund):void` und `removeHund(Hund):void` der Klasse `Hundesteuerzahler`

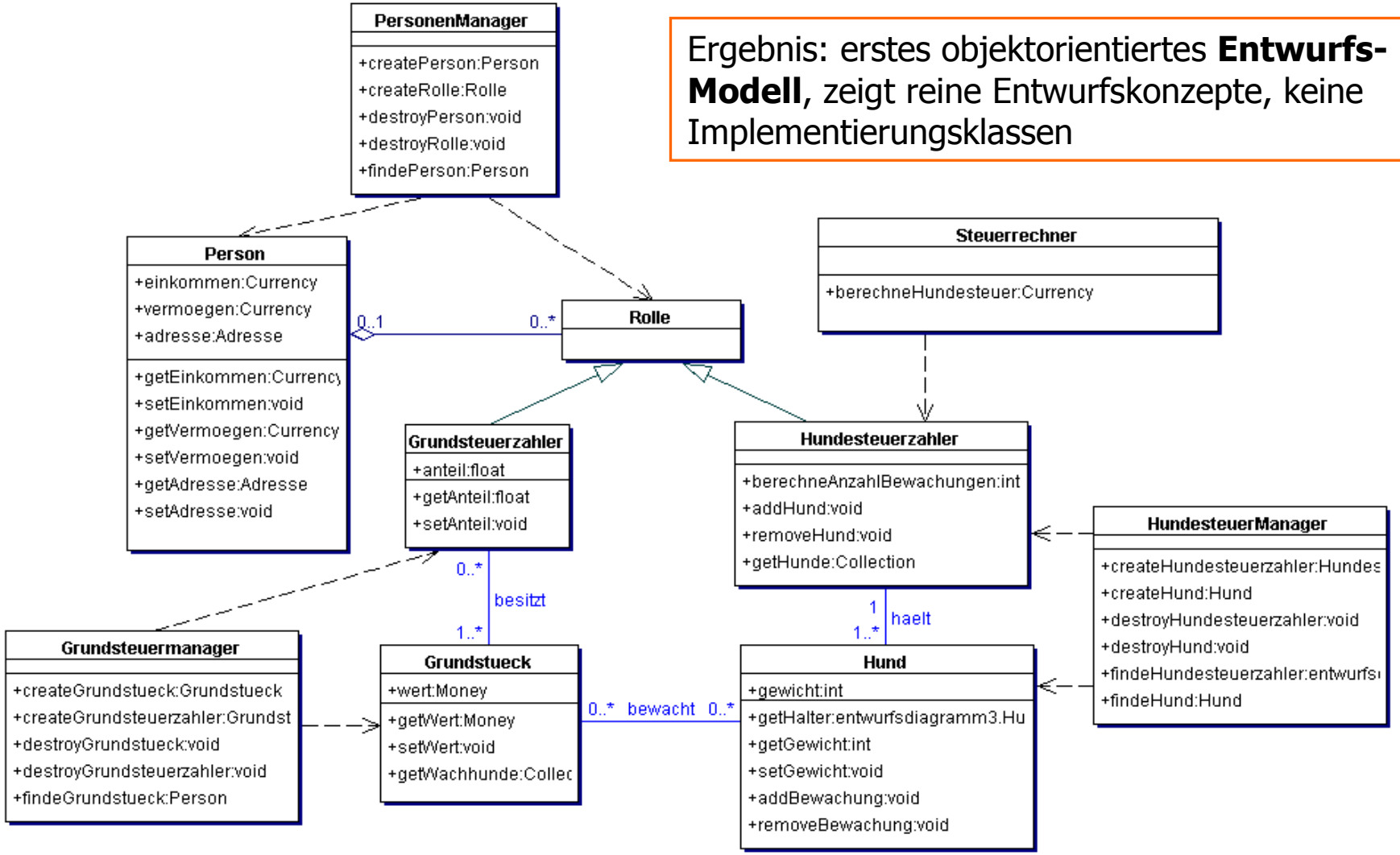
Zugriffs- und Verwaltungsoperationen

- Verwaltung der Instanzen
  - typischerweise über die Einführung von Verwaltungsklassen
  - Beispiele: Verwaltungsklasse `HundesteuerManager` mit Operationen zum Erzeugen, Löschen und Suchen von `Hundesteuerzahlern` und `Hunden`



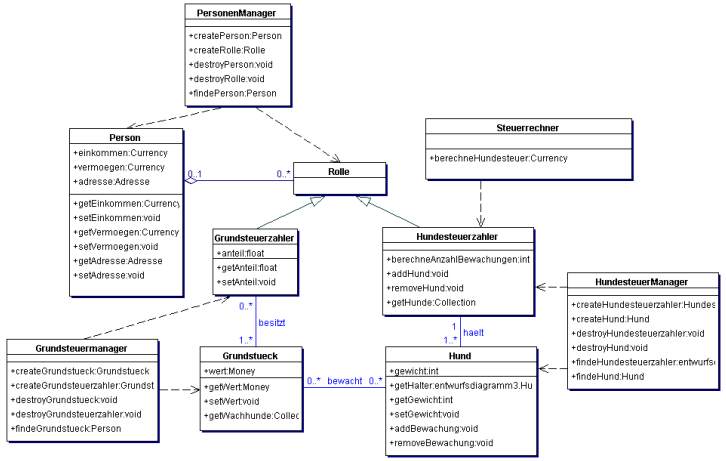
Zwischenstand der fachlichen Sicht

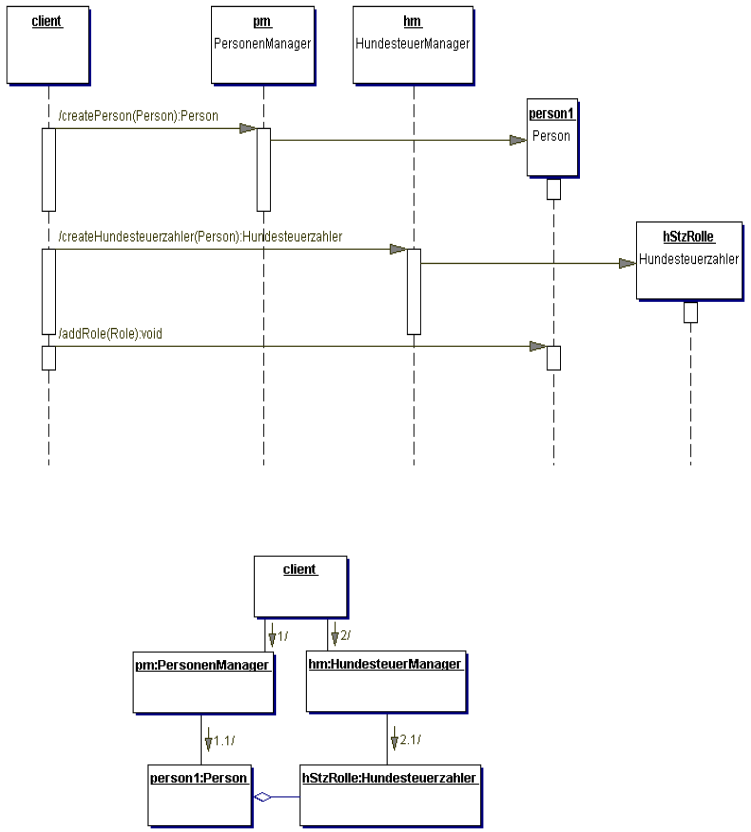
Ergebnis: erstes objektorientiertes **Entwurfs-Modell**, zeigt reine Entwurfskonzepte, keine Implementierungsklassen



Validierung durch Beispiele

- Für die Operationen im Klassendiagramm werden nun fachliche Abläufe durchgespielt und beispielhafte Objektstrukturen gebildet.





Klassen zur Modellierung

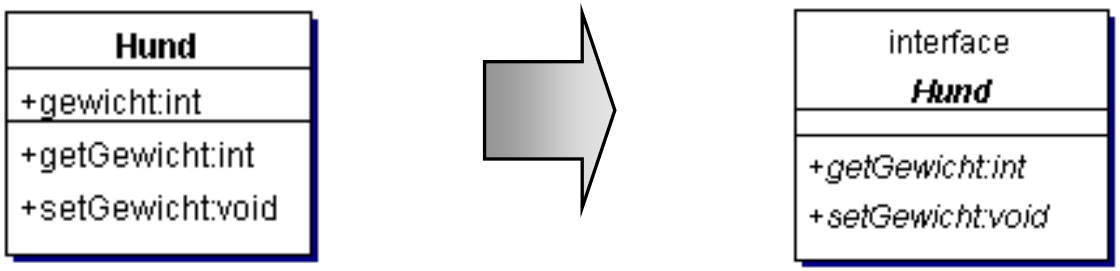
Beispiel-Instanzen zur Validierung

Inhalt

- Motivation und Eigenschaften
  - Was sind Sichten?
  - Überblick über die Sichten
- Fachliche Sicht
  - Vom Analysemodell zum fachlichen Entwurf
  - Bildung fachlicher Komponenten
- Technische Sicht
  - Technische Komponenten und Schnittstellen
  - Umsetzung der Fachlichkeit
- Verteilungssicht
  - Verteilung der Komponenten
- Zusammenfassung

Schnittstellenbildung

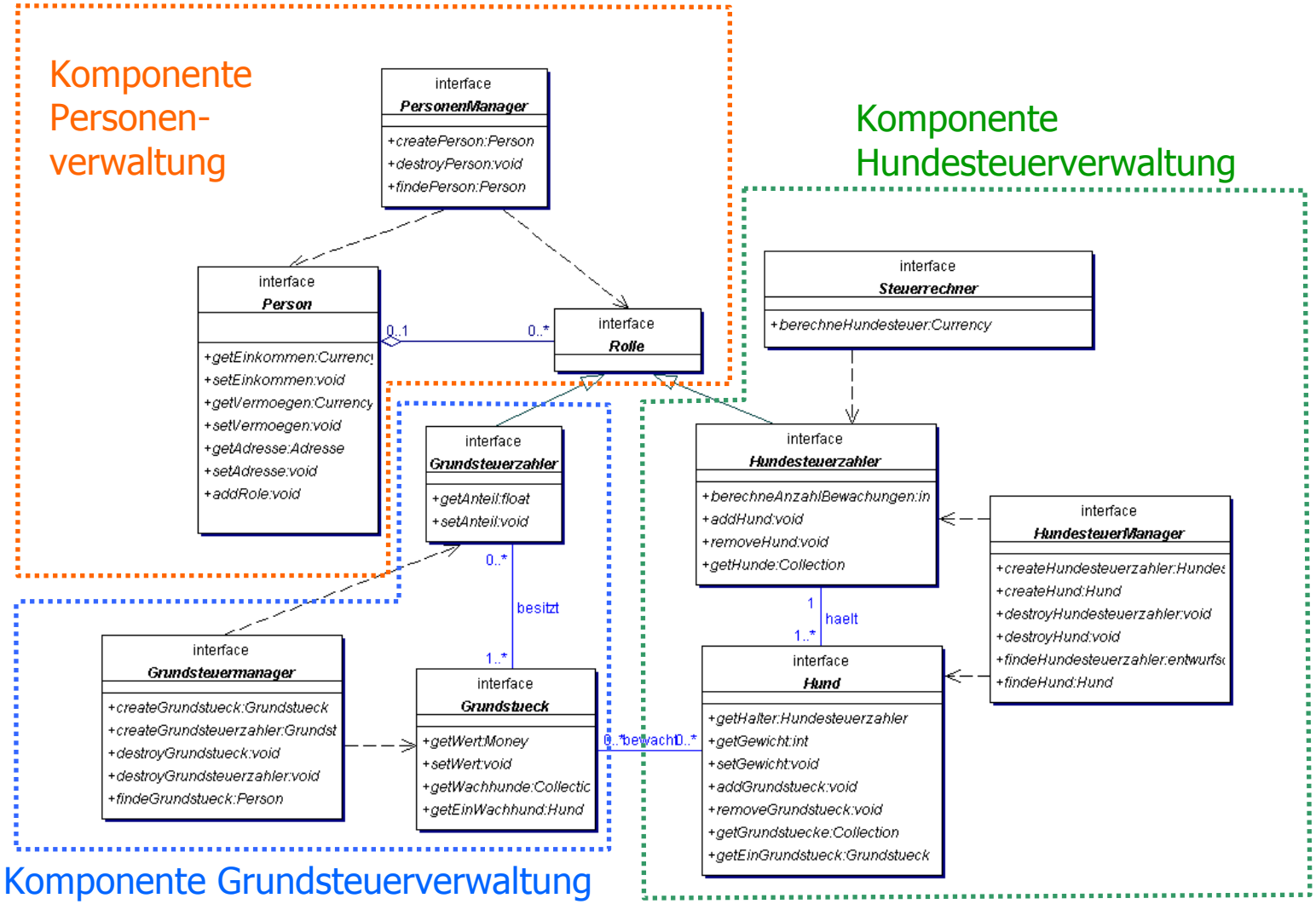
- Zur Vorbereitung der Komponentenbildung werden (möglichst schmale) Schnittstellen definiert und mit geeigneten Beschreibungstechniken beschrieben.
- Die Schnittstellen enthalten nur die Operationen, die später von der Komponente nach außen angeboten werden.
- Gegebenenfalls kann es unterschiedliche Sätze von Schnittstellen geben (z.B. Schnittstellen mit extern zugänglichen Operationen, Schnittstellen mit nur komponentenintern zugänglichen Operationen).



Bildung fachlicher Komponenten

- Die Bildung übergreifender fachlicher Komponenten erfolgt nach folgenden Gesichtspunkten:
  - Lose Kopplung – starke Bindung
  - Wiederverwendbarkeit für andere Systeme
  - Einfache Austauschbarkeit der Komponente
  - Konfigurierbarkeit des Systems und der Komponente
- In einem ersten Schritt wird dazu das Klassendiagramm partitioniert.
  - `Person` und `Rolle` stellen Basisfunktionalität für viele Anwendungen zur Verfügung → eigene Komponente.
  - Für die unterschiedlichen Steuerarten werden jeweils eigene Komponenten erstellt, um flexibel auf Änderungen reagieren zu können. Außerdem kann dann jede derartige Komponente von den betreffenden Fachleuten realisiert und getestet werden.

Partitionierung am Beispiel

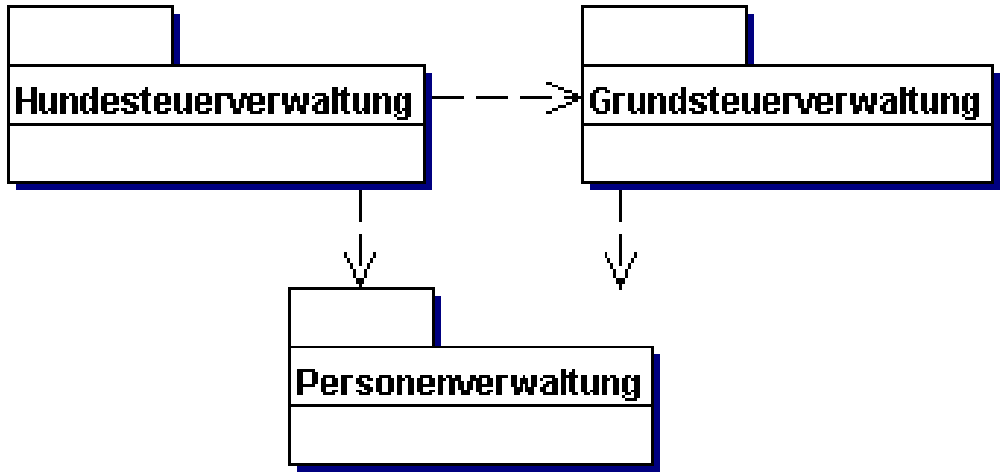


Abhängigkeiten zwischen Komponenten definieren

- Der Entwurf der Abhängigkeitsstruktur erfolgt nach folgenden Gesichtspunkten:
  - Komponenten mit Basisfunktionalität sollten von möglichst wenig anderen Komponenten abhängig sein. Umgekehrt: Je instabiler eine Komponente ist, desto weniger Komponenten sollten von ihr abhängig sein.
    - Fehlerfreiheit
    - Minimierung des Aufwands bei Änderungen
    - Auslieferung unterschiedlicher Systemkonfigurationen
  - Zyklen in der Abhängigkeitsstruktur sollten möglichst vermieden werden
    - leichtere Austauschbarkeit von Komponenten
    - Weglassen von Komponenten möglich
    - einfachere Abstimmung der Teams während der Entwicklung

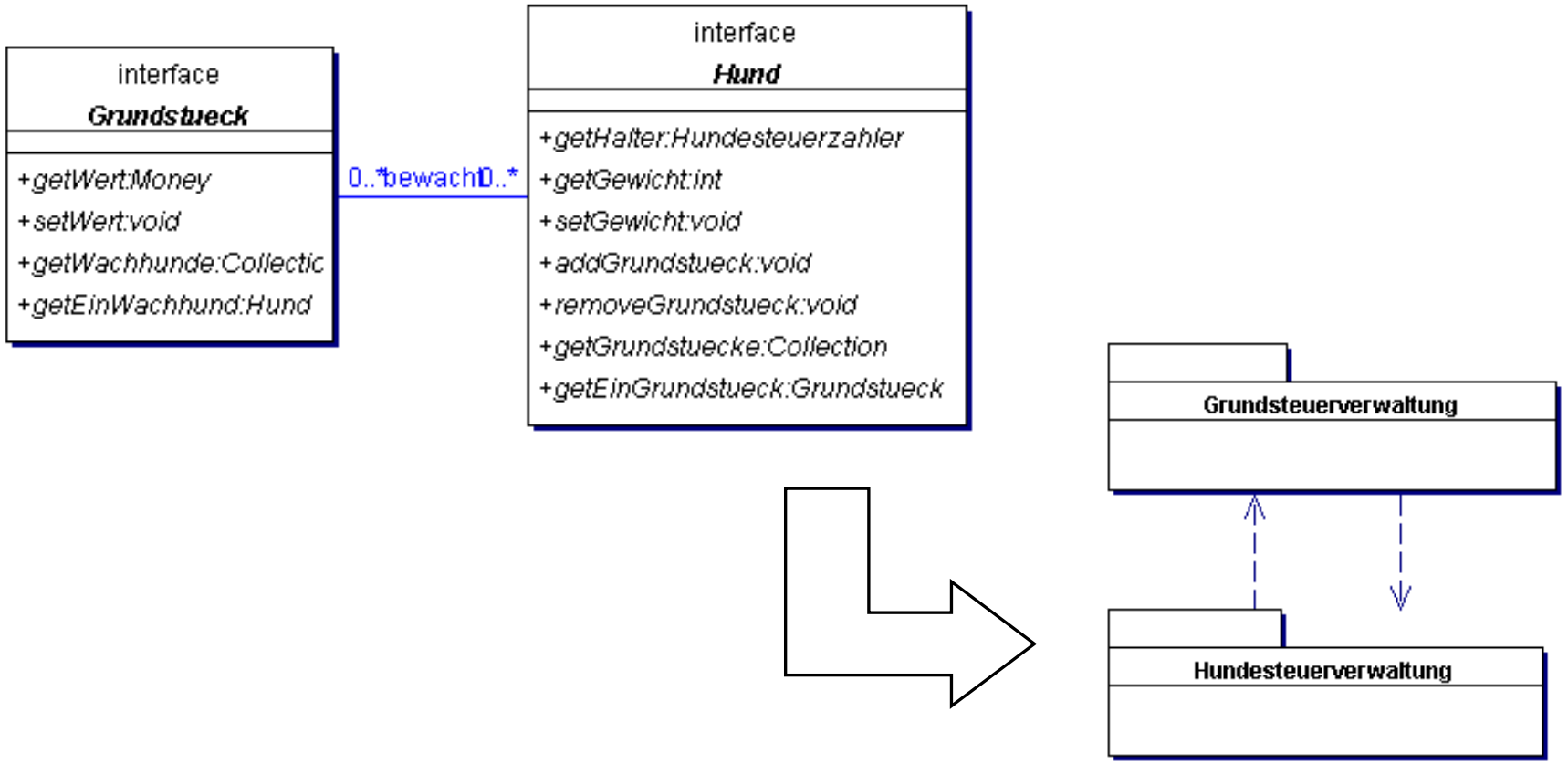
Abhängigkeitsstruktur im Anwendungsbeispiel

- Im Beispiel gibt es drei Komponenten:
  - `Personenverwaltung` als Basiskomponente
  - `Grundsteuerverwaltung` als relativ stabile Komponente
  - `Hundesteuerverwaltung` als relativ instabile Komponente
- Die Ahängigkeitsstruktur wird im Beispiel wie folgt entworfen:



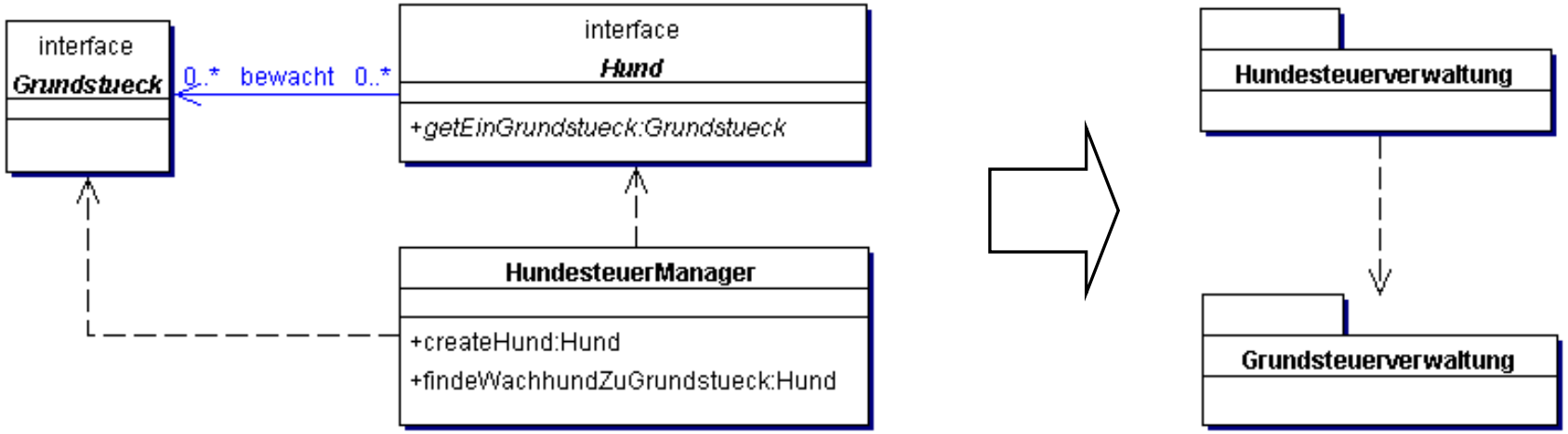
Bidirektionale Abhängigkeiten

- Bidirektional navigierbare Assoziationen resultieren normalerweise in zyklischen Abhängigkeiten zwischen den betreffenden Komponenten.



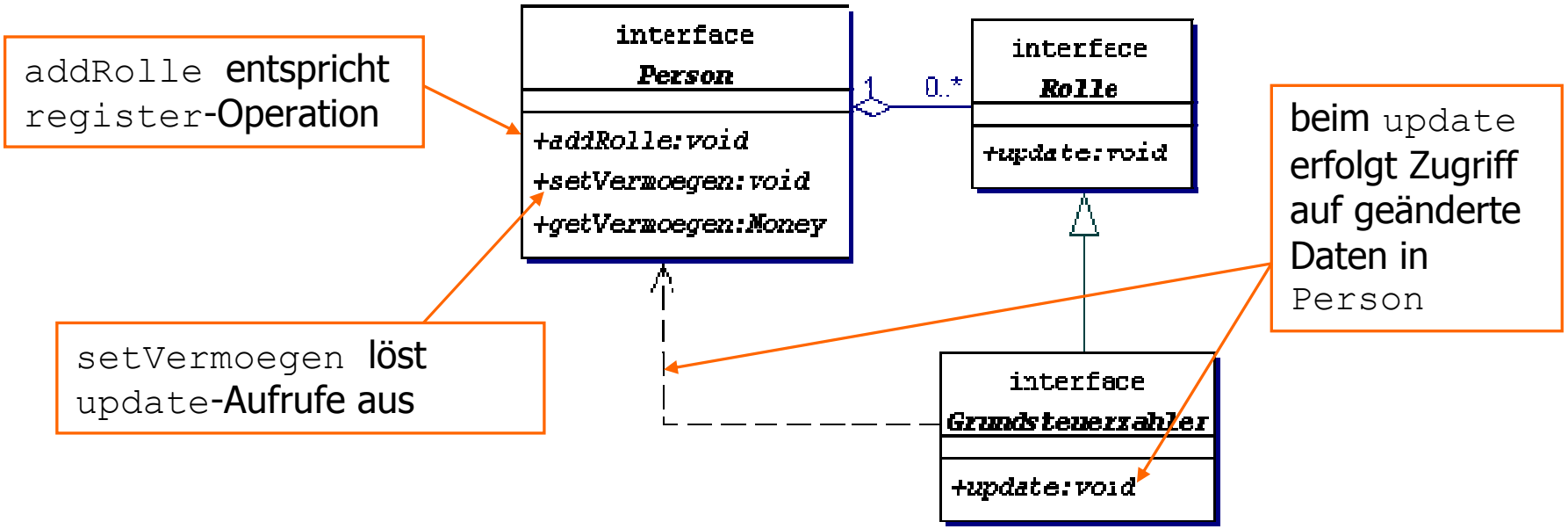
Aufbrechen von bidirektionalen Abhängigkeiten

- Diese Abhängigkeiten lassen sich beispielsweise über die Einführung einer neuen finde-Operation in dem Manager der Komponente aufbrechen.

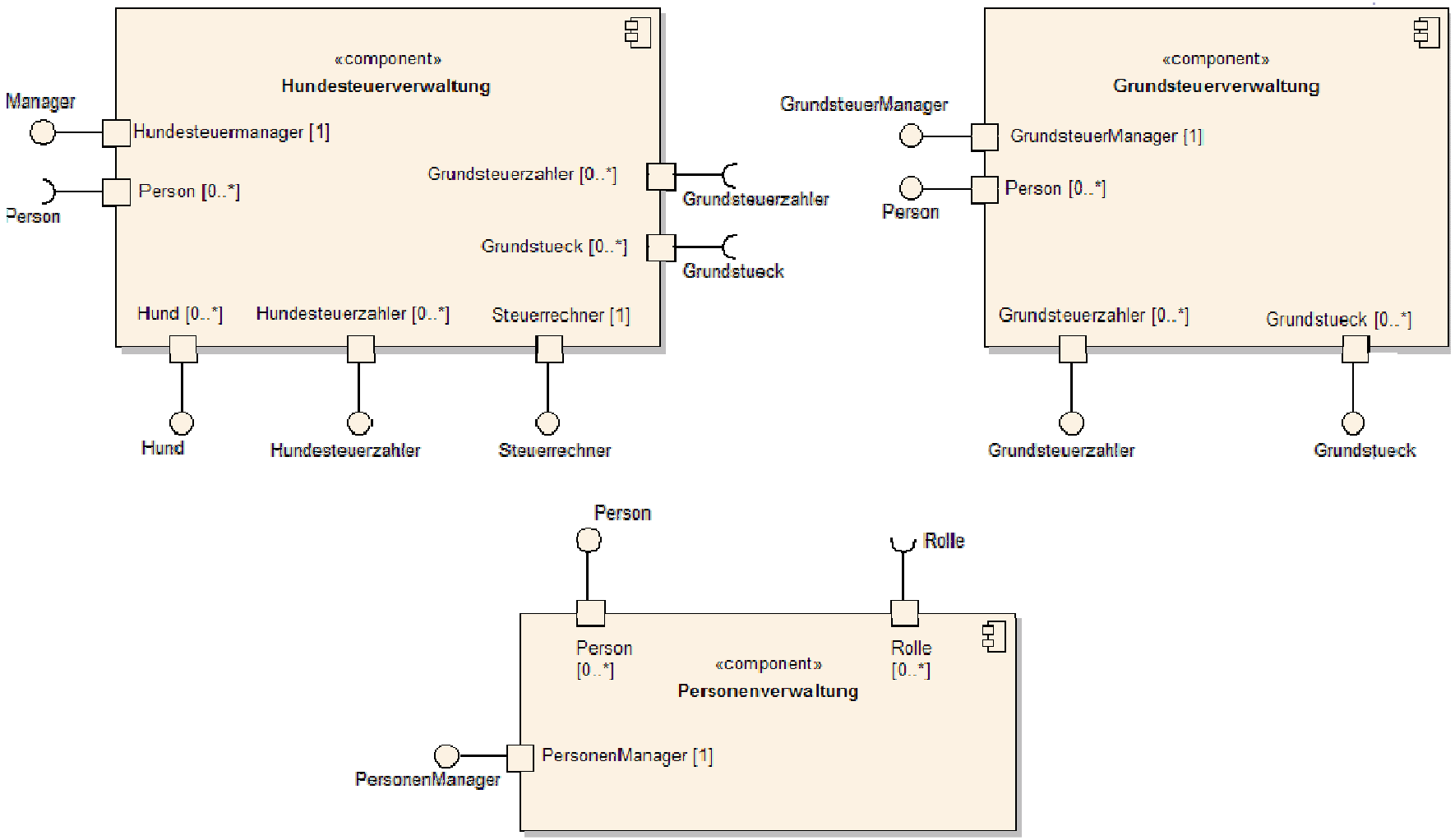


Kommunikation von Basis- zu abhängigen Komponenten

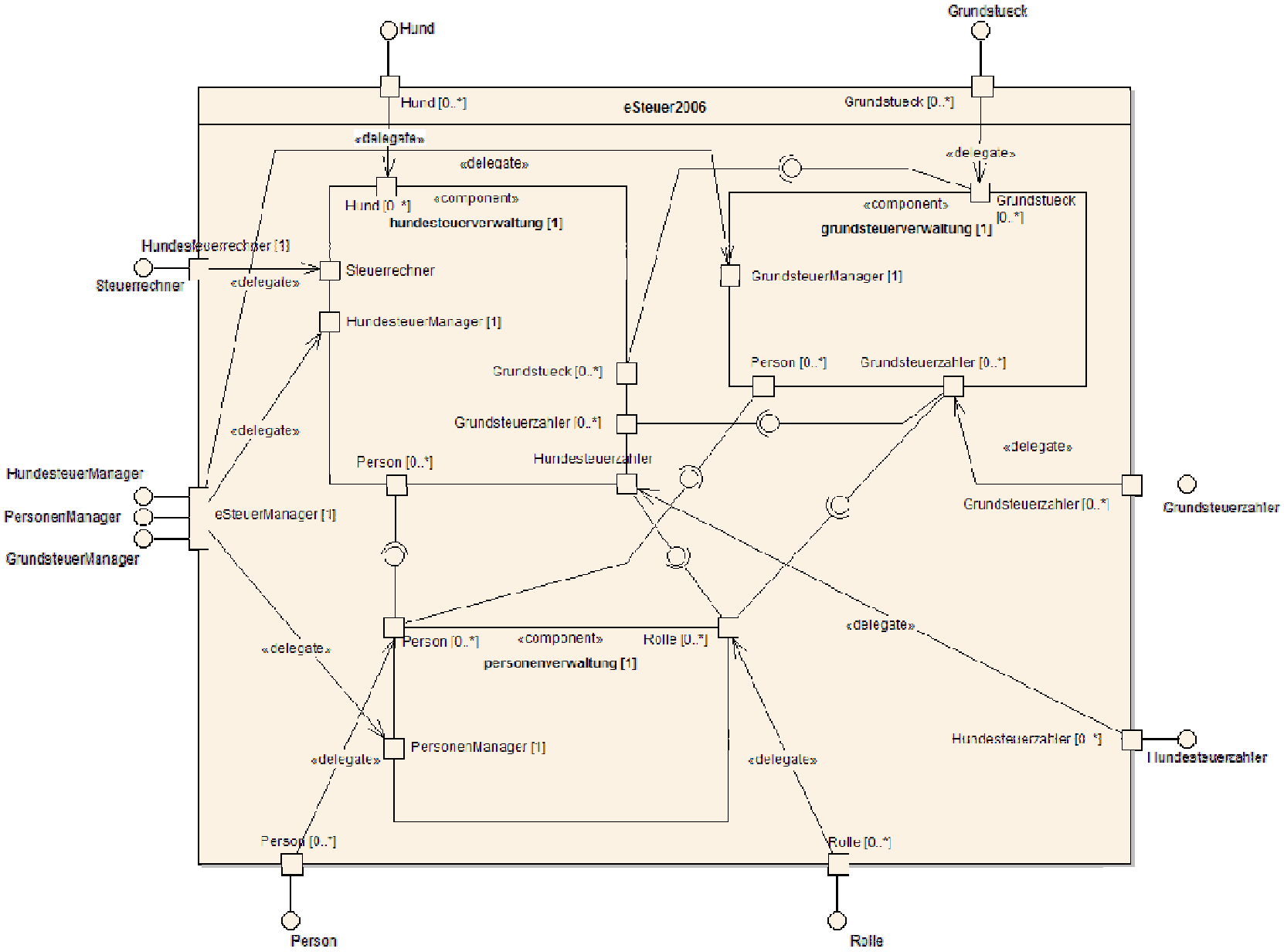
- Basiskomponenten können abhängige Komponenten nicht über deren eigene Schnittstellen rufen, da sich sonst eine zyklische Abhängigkeit ergäbe.
- Möglich ist jedoch beispielsweise eine Kommunikation über das Observer-Pattern durch Ableiten von der Schnittstelle `Rolle`.



Resultierende Komponenten



Mögliche Verbindungsstruktur der Komponenten



Inhalt

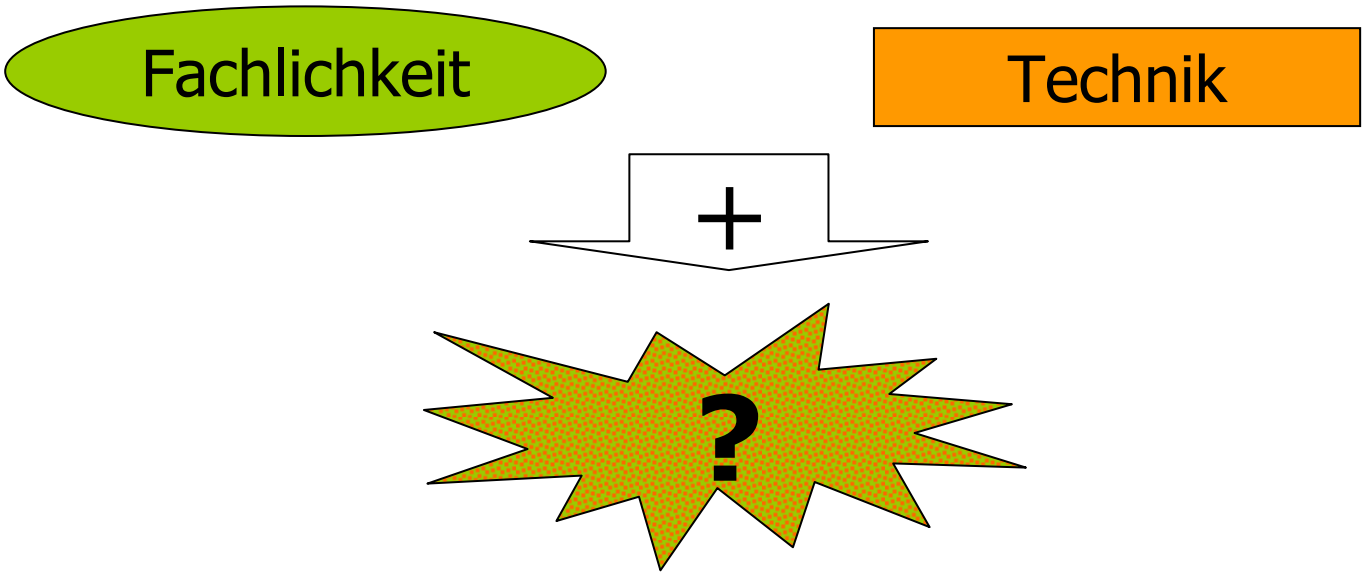
- Motivation und Eigenschaften
  - Was sind Sichten?
  - Überblick über die Sichten
- Fachliche Sicht
  - Vom Analysemodell zum fachlichen Entwurf
  - Bildung fachlicher Komponenten
- Technische Sicht
  - Technische Komponenten und Schnittstellen
  - Umsetzung der Fachlichkeit
- Verteilungssicht
  - Verteilung der Komponenten
- Zusammenfassung

Fragestellungen in der technischen Sicht

1. Wie wird die technische Funktionalität repräsentiert?
   - Wie werden die nicht-funktionalen Anforderungen erfüllt?
   - Welche Komponenten und Schnittstellen gibt es?
   - Welche Querschnittsmechanismen und Basistechniken gibt es?
   - Wie arbeiten sie zusammen?
   - ➡ Vorgehen und Beschreibung wie bei Fachlicher Sicht
   - ➡ Beispiele in später folgenden Vorlesungen
2. Wie ist der Zusammenhang zur fachlichen Sicht?
   - Was ist die Schnittmenge zwischen den beiden Sichten?
   - Wie finden sich die fachlichen Konzepte in der technischen Sicht wieder?
   - Beispiele in später folgenden Vorlesungen
   - ➡ Möglichkeiten auf den folgenden Folien

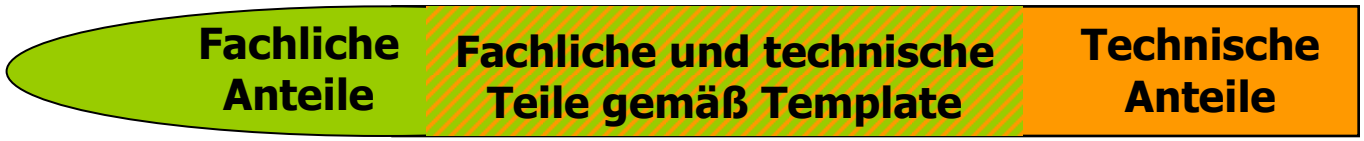
Entkopplung der technischen Funktionalität

- Damit die fachlichen Komponenten arbeiten können, stützen sie sich auf Dienste der technischen Komponenten ab.
- Problem: Bei explizit verwendeten technischen Diensten keine Entkopplung und damit eine enge Bindung von fachlichen und technischen Komponenten!
- Frage: Wie schafft man die Entkopplung?



Strategie: Templates für schematische Kopplung

- Programmierrichtlinien oder Templates geben vor, wie die Technik im Programm angesprochen werden kann bzw. muss.



- Probleme
  - Einhaltung der Richtlinien schwierig zu überwachen
  - Templates können Code sehr unübersichtlich machen
  - Keine echte Trennung von fachlichem und technischem Code: Wechsel auf neue technische Infrastruktur kann ggf. sehr aufwändig sein
- Beispiel
  - Methoden-Template zum Tracing von Methoden und zum Überprüfen von Pre- und Postconditions

Methodentemplate (Preconditions & Trace Entry) (1)

```
public void doSomeThing(String arg1, int arg2) {
   if (CheckManager.CHECK_PRECONDITION) {
      CheckManager.assertPreCondition(this, new
         NotNullAssertion(arg1));
      CheckManager.assertPreCondition(this,
         arg2 > 0, "arg2 <= 0");
   }
   boolean exceptionThrown = false;
   TraceManager.traceCallEntry(this,
      new java.lang.Object[] { s });
   try {
       … fachlicher Code …
```

Methodentemplate (Trace Exceptions) (2)

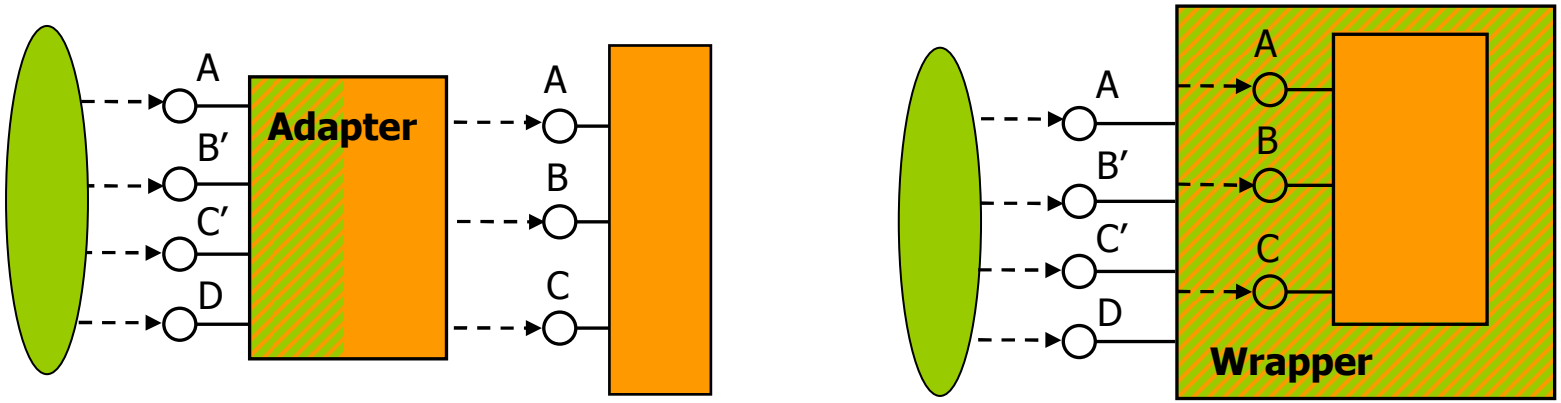
```
} // try
catch (RuntimeException ex) {
   exceptionThrown = true;
   TraceManager.traceExceptionThrown(this, ex);
   throw ex;
}
catch (Error err) {
   exceptionThrown = true;
   TraceManager.traceExceptionThrown(this, ex);
   throw err;
}
```

Methodentemplate (Postconditions & Trace Exit) (3)

```
    finally {
        if (!exceptionThrown) {
            TraceManager.traceCallExit(this);
            if (CheckManager.CHECK_POSTCONDITION) {
                CheckManager.assertPostCondition(this, new
                    NotNullAssertion(arg1));
            }
        }
    }
} // doSomeThing
```

Strategie: Technik hinter Schnittstellen kapseln

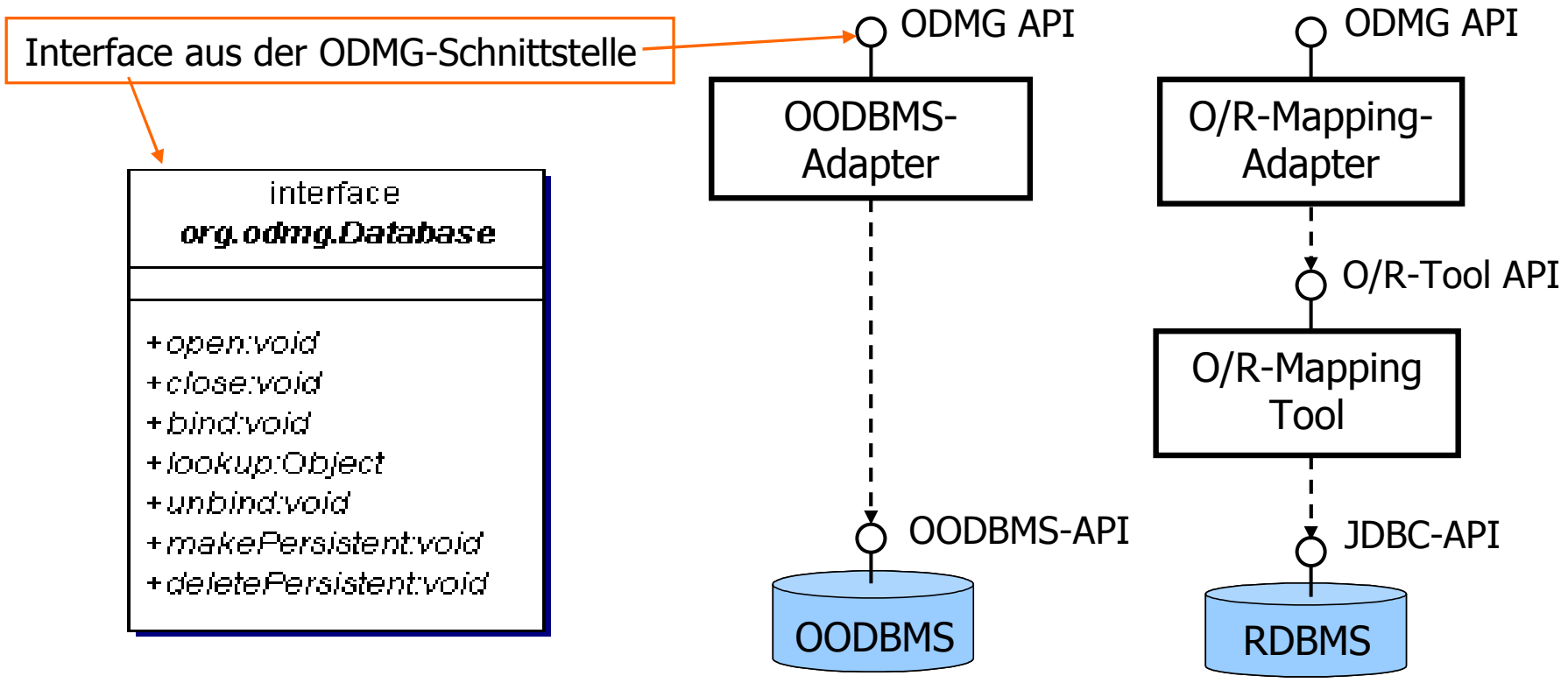
- Adapter und Wrapper dienen dazu
  - komplexe und/oder proprietäre technische Schnittstellen und Mechanismen vor der fachlichen Sicht zu verstecken und
  - stattdessen einfache, möglichst standardisierte Schnittstellen für Anwendungsdienste bereitzustellen.



- Mögliche Probleme:
  - Technik scheint immer noch durch
  - Komponenten sind von Schnittstellen der Adapter/Wrapper abhängig

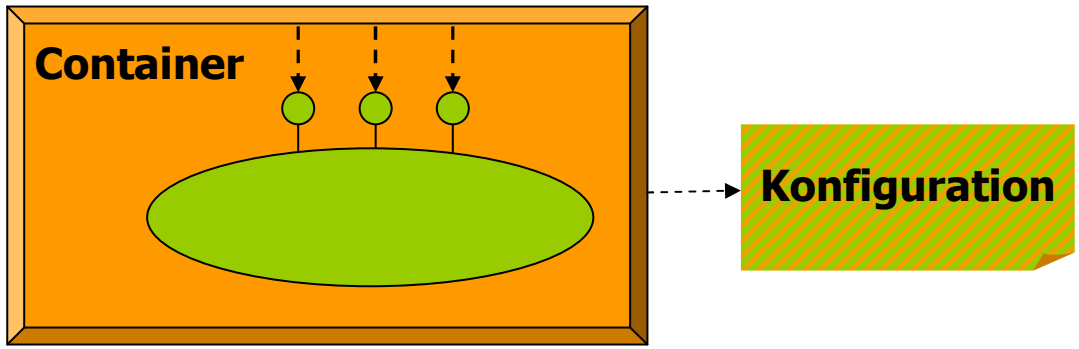
Beispiel: Standardschnittstellen für Persistenz

- Der Zugriff auf die Verwaltungsfunktionalität geschieht über einfache, möglichst standardisierte Schnittstellen.



Strategie: Verantwortung auf Container abwälzen

- Fachlichkeit wird möglichst ganz ohne Bezug auf technische Basisfunktionalität realisiert (Plain Old Java Objects, POJOs).
  - Ein Container versorgt die fachlichen Komponenten mit allem, was sie brauchen und regelt die Technik im Hintergrund.
  - Der Container benötigt dazu Wissen über die Komponente, das er z.B. über eine Konfigurationsdatei oder über reflexive Introspektion erhält.



- Mögliche Probleme
  - Nur möglich, wenn Verwaltung der technischen Funktionalität konfigurationsgesteuert (ohne Programmierung) möglich ist
  - Konfigurationsdateien können sehr komplex werden

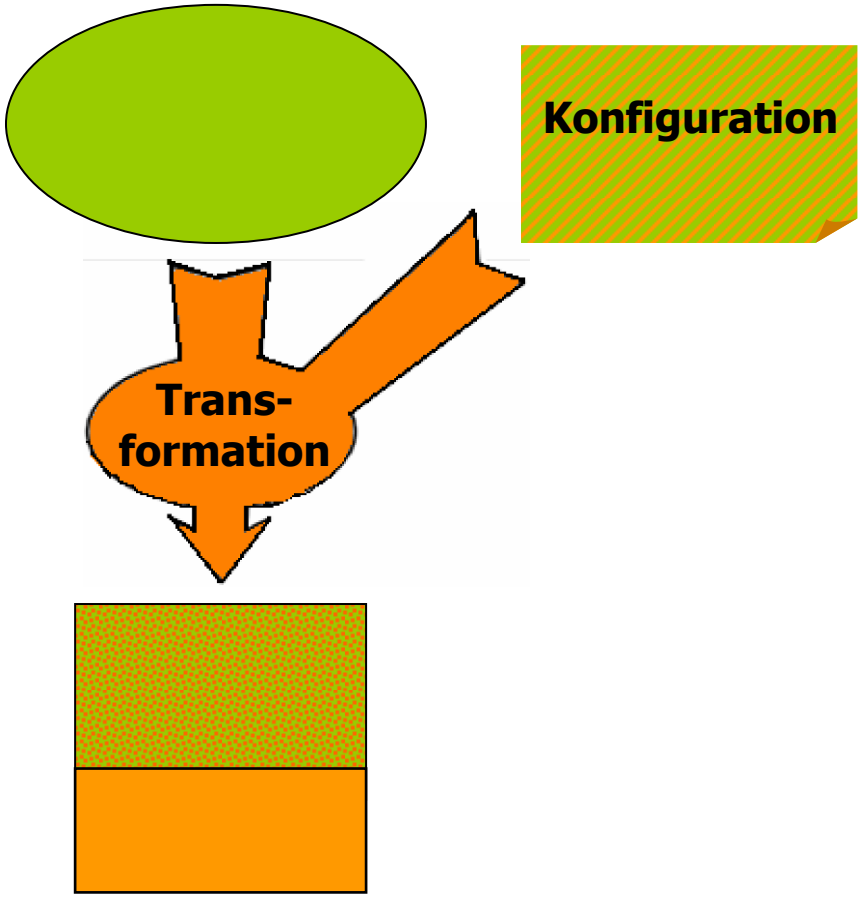
Beispiel: Deklaratives Transaktionsmanagement

- Transaktionssteuerung bei einem Anwendungsserver
  - Component-Managed Transactions: Die fachliche Komponente erzeugt Transaktionen und ruft `begin, commit` und `abort` auf ihnen auf.
  - **Container-Managed Transactions: Der Application Server bestimmt, wann eine Transaktion geöffnet und beendet wird.**
- Beispiel: Ausschnitt aus Konfigurationsdatei für deklarative Transaktionsverwaltung im Spring Application Framework

```
<bean id=„trx_hundesteuermanager"
  class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
  <property name="transactionManager" ref="txManager"/>          Spring-Mechanismus
  <property name="target" ref=„hundesteuermanager"/>             Verwaltete Komponente
  <property name="transactionAttributes"><props>
    <prop key="create*">PROPAGATION_REQUIRED</prop>
    <prop key=„finde*">PROPAGATION_REQUIRED,readOnly</prop>
  </props></property>                                            Transaktionsattribute für Operationen
</bean>
```

Strategie: Entkopplung durch Generierung

- Die Fachlichkeit wird möglichst ohne Bezug auf die Technik spezifiziert.
- Ein Generator erzeugt mit Hilfe einer Konfiguration die Implementierung.
- Probleme:
  - Änderungen am generierten Ergebnis müssen definiert behandelt werden
  - Konfiguration kann sehr komplex und schwer verständlich werden
  - Schlechte Generatoren erzeugen schlechten Code
  - Bau von Generatoren ist nicht einfach; Erfahrung mit manueller Abbildung ist hilfreich



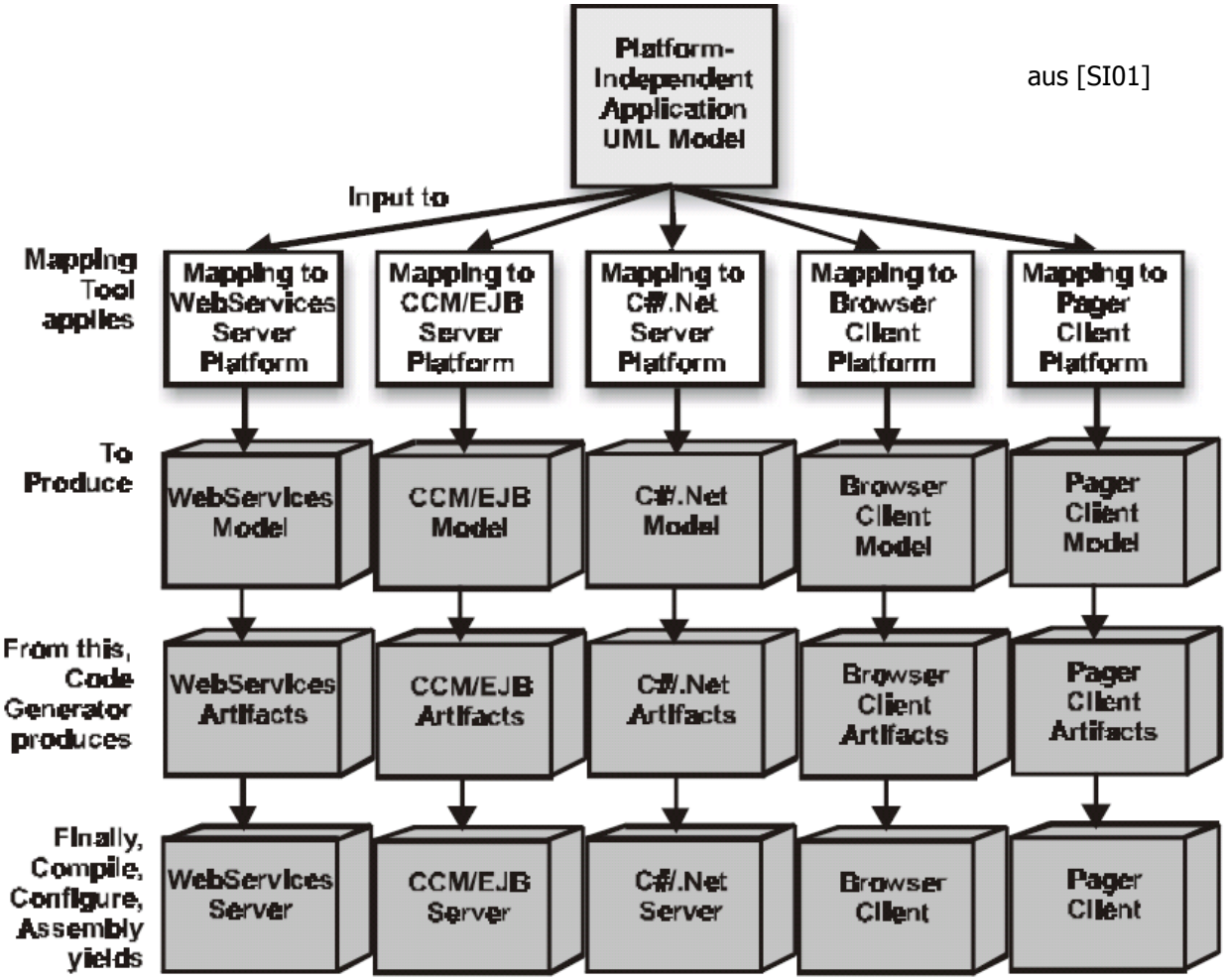
Beispiel: PIM-to-PSM-Mappings bei MDA

Fachliche Sicht (PIM)

Technische Sicht (PSM)

Entwicklungssicht (PSM)

Bereitstellungs-Sicht (PSM)



aus [SI01]

Beispiel: EJB3 Annotations

```
@Entity
@Table(name=„HUND")
@NamedQueries( { @NamedQuery(name =
   "findHundById", query = "from HUND i where i.id = ?")})
public class Hund implements IdentifiableObject {
   @Id @GeneratedValue private long id ;
   ...
}
```

- Fachlicher Quellcode und technische Konfiguration stehen beisammen, sind leichter konsistent zu halten.
- Nachteil: Technische Aspekte vermischt mit Fachlichkeit; Wechsel auf andere technische Plattform erschwert.

Beispiel: Aspect-Oriented Programming (AOP)

```
public aspect Tracing {
  private static Tracer tracer = Tracer.getTracer();

  pointcut myClass(): this(Break) || this(BreakPlan) ||
                      this(Teacher);
  pointcut myConstructor(): myClass() &&
                            initialization(new(..));
  pointcut myMethod(): myClass() && execution(* *(..));

  before (): myConstructor() {
    tracer.entering("Entering " +
                    thisJoinPoint.getSignature());
  }
  after(): myConstructor() {
    tracer.exiting("Exiting " +
                   thisJoinPoint.getSignature());
  }
  before (): myMethod() {
    tracer.entering ("Entering " +
                     thisJoinPoint.getSignature());
  }
  after(): myMethod() {
    tracer.exiting("Exiting " +
                   thisJoinPoint.getSignature());
  }
}
```

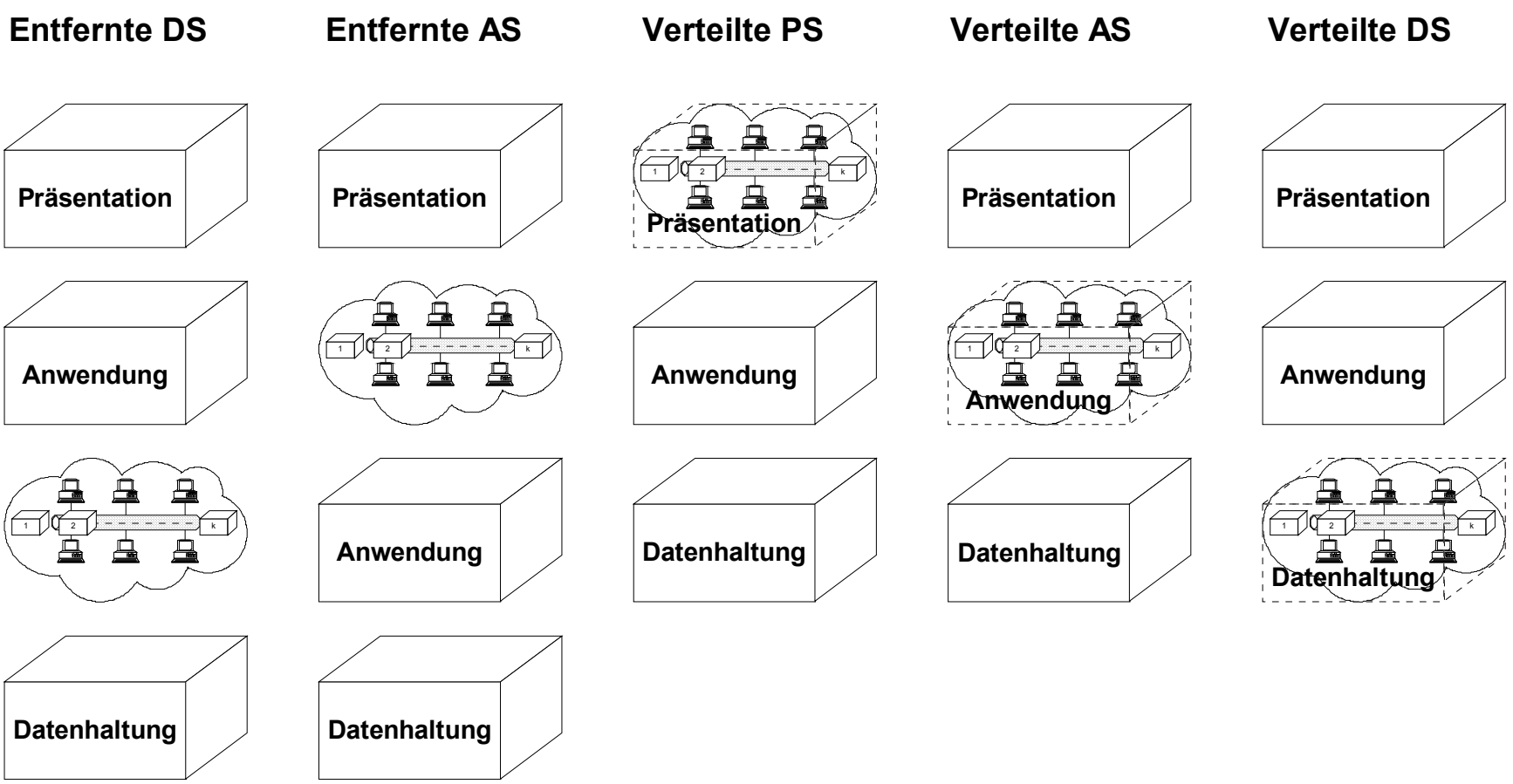
Inhalt

- Motivation und Eigenschaften
  - Was sind Sichten?
  - Überblick über die Sichten
- Fachliche Sicht
  - Vom Analysemodell zum fachlichen Entwurf
  - Bildung fachlicher Komponenten
- Technische Sicht
  - Technische Komponenten und Schnittstellen
  - Umsetzung der Fachlichkeit
- Verteilungssicht
  - Verteilung der Komponenten
- Zusammenfassung

Verteilungssicht - Überblick

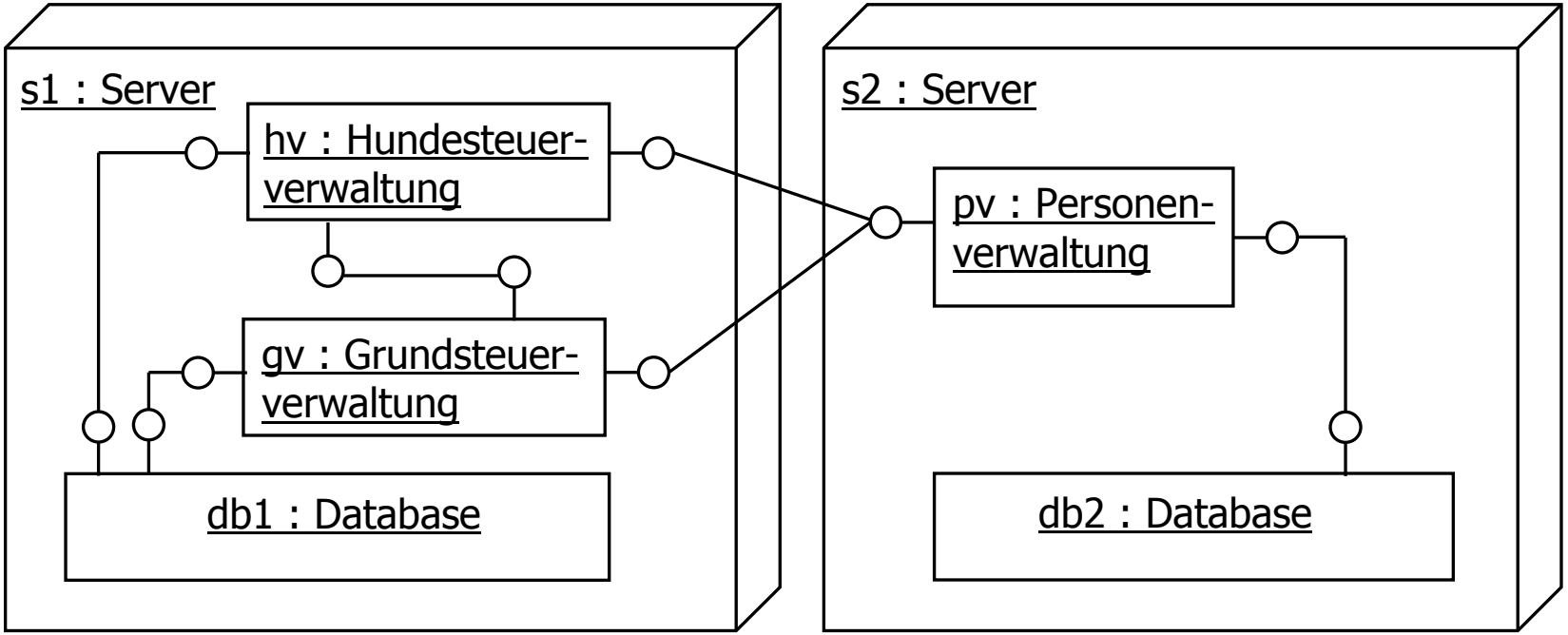
- Zweck
  - Darstellung der Verteilung der Komponenten zur Ausführungszeit auf die Prozessoren und Rechner
- Inhalte
  - Abbildung der Komponenten des Systems auf die Elemente der Systemarchitektur zur Laufzeit
  - Darstellung des Client-/Server-Schnitts
  - Darstellung der ausführungsnahen Qualitätseigenschaften, wie zum Beispiel Performance
- Beschreibungen
  - Textuelle Beschreibung und informelle Box-and-Line-Diagramme
  - Kompositionsstrukturdiagramme von UML

Kern-Entwurfsaufgabe: der Client-Server-Schnitt



Beschreibung der Verteilungssicht

- Die Verteilungssicht zeigt die Verteilung von fachlichen und technischen Komponenteninstanzen auf die Elemente der Systemarchitektur.
- Beispiel eSteuer 2010:



Inhalt

- Motivation und Eigenschaften
  - Was sind Sichten?
  - Überblick über die Sichten
- Fachliche Sicht
  - Vom Analysemodell zum fachlichen Entwurf
  - Bildung fachlicher Komponenten
- Technische Sicht
  - Technische Komponenten und Schnittstellen
  - Umsetzung der Fachlichkeit
- Verteilungssicht
  - Verteilung der Komponenten
- Zusammenfassung

Zusammenfassung

- Eine Sicht stellt jeweils bestimmte, zusammengehörige Eigenschaften eines Systems dar.
- Es gibt zwei Hauptarten von Sichten: Spezifikationssichten zur Beschreibung des Systems und Entwurfssichten zur Beschreibung seiner Entwicklung.
- Bei den Spezifikationssichten sollte die fachliche Sicht vor der technischen Sicht bearbeitet und ausmodelliert werden.
- Sowohl in der fachlichen als auch in der technischen Sicht ist die Bildung von Komponenten mit klar definierten Schnittstellen und Abhängigkeitsbeziehungen wichtig.
- Zur Entkopplung von fachlicher und technischer Sicht gibt es eine Reihe von Standard-Techniken. Dazu gehören insbesondere die Kapselung der Technik hinter Schnittstellen, Container-Architekturen sowie Generierung.

Literaturhinweise

- [BRS97] Klaus Bergner, Andreas Rausch, Marc Sihling: Using UML for Modeling a Distributed Java Application, Technischer Bericht der Universität München, TUM-I9735, http://wwwbib.informatik.tu-muenchen.de/infberichte/1997/TUM-I9735.ps.gz, 1997.
- [CBB02] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford: *Documenting Software Architectures – Views and Beyond*, Addison-Wesley, 2002.
- [HNS99] Christine Hofmeister, Robert Nord, Dilip Soni: *Applied Software Architecture*, Addison Wesley – Object Technology Series, 1999.
- [HS99] Peter Herzum, Oliver Sims. Business Component Factory: *A Comprehensive Overview of Component-Based Development for the Enterprise*, John Wiley & Sons, 1999.
- [Kr94] Mario Jeckle, Chris Rupp, Jürgen Hahn, Barbara ZenglerStefan Queins: *UML2 glasklar*, Carl Hanser Verlag, 2004.
- [OM01] Object Management Group: *Model Driven Architecture (MDA) Website*, www.omg.org/mda, OMG 2005.
- [SI01] J. Siegel, OMG Staff Strategy Group: *Developing in OMG´s Model-Driven Architecture*, OMG White Paper, November 2001.
- [Sp05] Spring Framework Website, www.springframework.org, 2005.