



Technische Hochschule  
Ingolstadt

Fakultät für Elektrotechnik  
und Informatik

*Zukunft in  
Bewegung*

# *Software Kategorien Betrieblicher Informationssysteme*

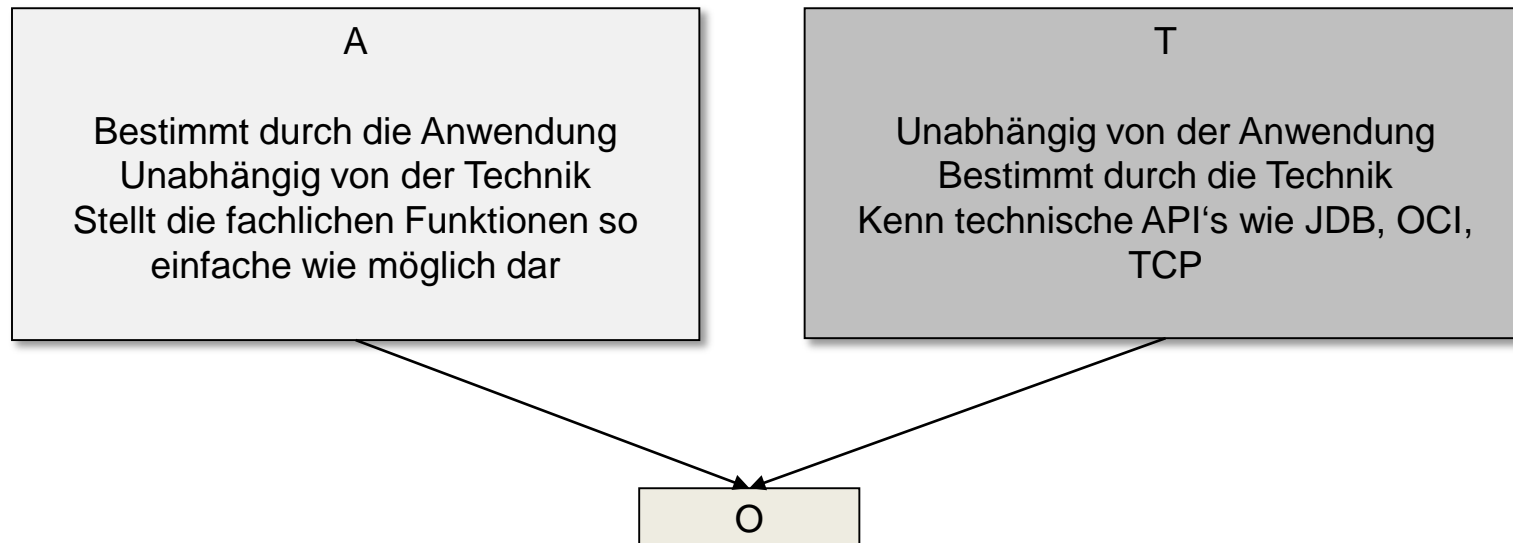
*Architektur- und Entwurfsmuster  
der Softwaretechnik*

Prof. Dr. Bernd Hafenrichter 06.03.2015



## Software Kategorien – Wie findet man Komponenten

### Standardkategorien für betriebliche Informationssystem



- Die strikte Trennung zwischen A und T führt zu einer strikten Trennung der Verantwortlichkeiten
- Die strikte Trennung führt zu einer hohen „Software-Hygiene“



### Software Kategorien – Wie findet man Komponenten

#### Standardkategorien für betriebliche Informationssystem

- Komponenten der Kategorie A und T dürfen nicht miteinander vermischt werden
- Andernfalls entsteht Software der unreinen Kategorie AT
- Kombination verschiedener Softwarekategorien

+	0	A	T	R
0	0	A	T	R
A		A	AT	./.
T			T	./.
R				R

./. – erlaubte Mischform da reine Transformationssoftware



### Software Kategorien – Wie findet man Komponenten

#### Standardkategorien für betriebliche Informationssystem

Warum ist die Trennung von A und T Software wichtig

- Parnas (1972): Software die sich unterschiedlich schnell ändert soll in verschiedenen Modulen untergebracht sein
- Beispiel:
  - Eine Client-/Server-Software wurde auf Basis von Corba entwickelt
  - CORBA soll nun durch SOAP ersetzt werden
  - Technische Standards entwickeln sich weiter. z.B. EJB 1.0, 2.0, 3.0

Die technische Basis heutiger System verändert sich permanent/schnell. Deshalb ist die Trennung wichtig um Flexibilität und Erweiterbarkeit einfach zu gestalten



## Software Kategorien – Wie findet man Komponenten

### Standardkategorien für betriebliche Informationssystem

- Die Trennung von A und T kostet Performance
  - Ja, aber die erzielten Vorteile überwiegen den Verlust von Performance
  - Insbesondere bei heutiger Hardware
- Die Trennung von A und T ist aufwendig
  - Ja, aber nur im Entwurfsprozess
  - Später sind weniger Experten notwendig um die einzelnen Komponenten zu realisieren (Separation of Concerns)
- Die Trennung ist überflüssig da die Technik niemals getauscht wird
  - eine Trennung von Verantwortlichkeiten ist immer richtig, unabhängig ob die Technologie ausgetauscht wird oder nicht
  - Stichpunkt: Gute Softwarehygiene

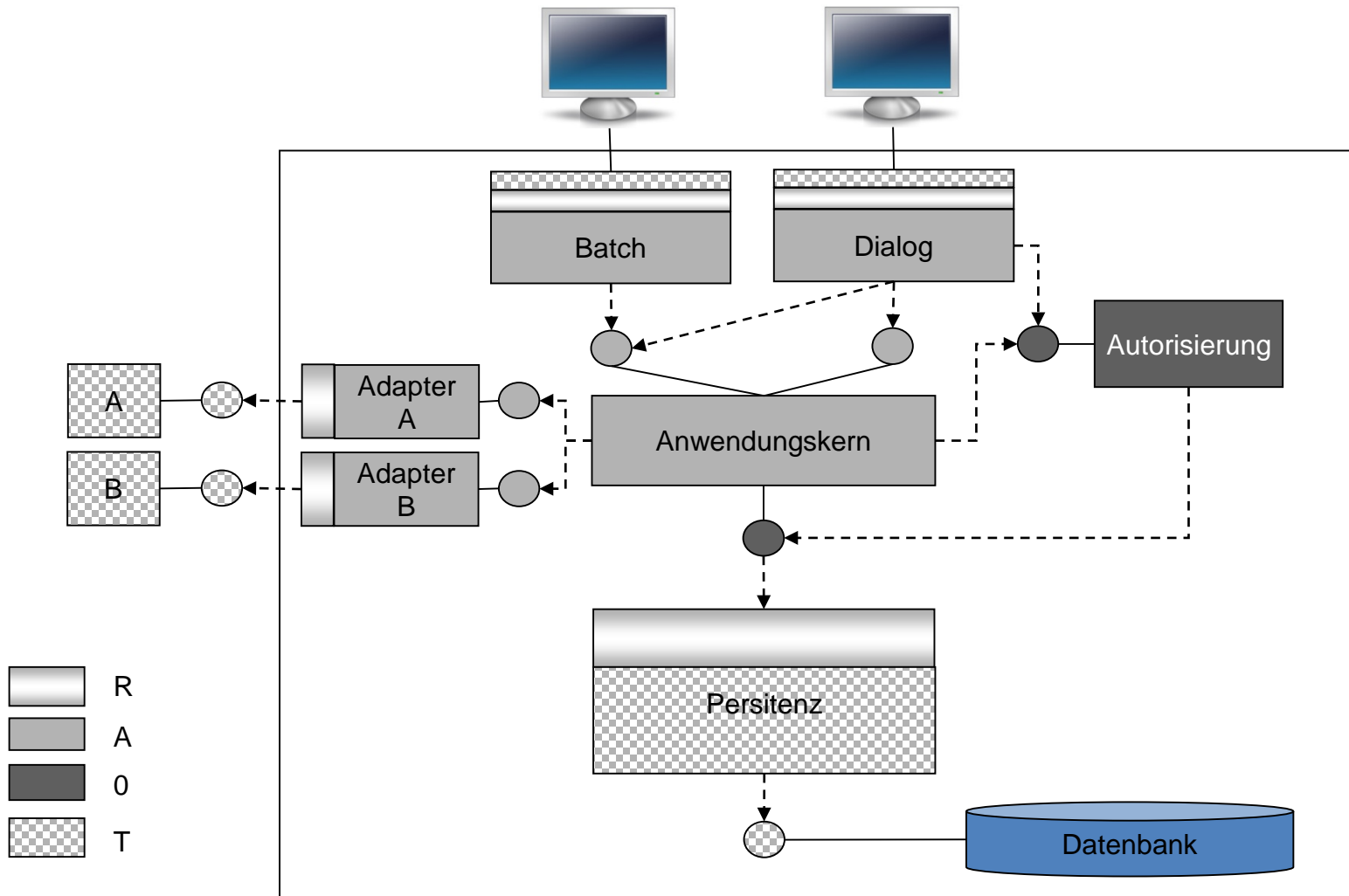


## Software Kategorien – Wie findet man Komponenten

### Fazit – Fünf Regeln zu Softwarekategorien

- Definiere die Softwarekategorien nach den definierten Regeln und Prinzipien
- Jede einfache Komponente (Modul) gehört zu einer definierten Kategorie
- Verbinde Software verschiedener Kategorien mit Hilfe von 0-Schnittstellen und R-Software
- Unreine Kategorien sind nur in begründeten Ausnahmen gestattet
- Schreibe möglichst wenig Software der speziellen Kategorien und möglichst viel der allgemeinen

## Architektur eines Informationssystems





## Definition von Komponenten-Schnittstellen

### Enge & Losekopplung

- Kopplung innerhalb eines Programmsystems entsteht durch öffentliche Methoden die von Schnittstellen exportiert und vom Komponenten benutzt werden
  - Jede Methode definiert Argumente, Rückgabewerte und Ausnahme
  - Die definierten Typen müssen öffentlich für alle Importeure zugänglich sein.
  - Dies kann bei falscher Anwendung zu ungewollten Effekten führen
- Um die Kopplung eines Systems kontrollieren zu können müssen Schnittstellen so definiert werden, das die Forderung nach loser/enger Kopplung bewusst ausgewählt wird






## Definition von Komponenten-Schnittstellen

### Enge & Losekopplung

- Eine falsche Anwendung der Kopplung führt zu hohen Abhängigkeiten zwischen verschiedenen Komponenten.
  - Wartbarkeit: Jede Änderung in einer Komponente kann unzählige Änderungen bei den Importeuren bewirken
  - Betrieb: Kann das System stabil betrieben werden oder ist es zu komplex?

## Definition von Komponenten-Schnittstellen

### Enge & Losekopplung

- Für die Übergabe von Entitätstypen in einer Schnittstelle stehen folgende Varianten zur Verfügung
    - Domänenobjekte
    - Entitätsschnittstelle
    - Transportklasse (spezieller Datentyp)
- 
- |  |                   |
|--|-------------------|
|  | Hohe Kopplung     |
|  | Niedrige Kopplung |

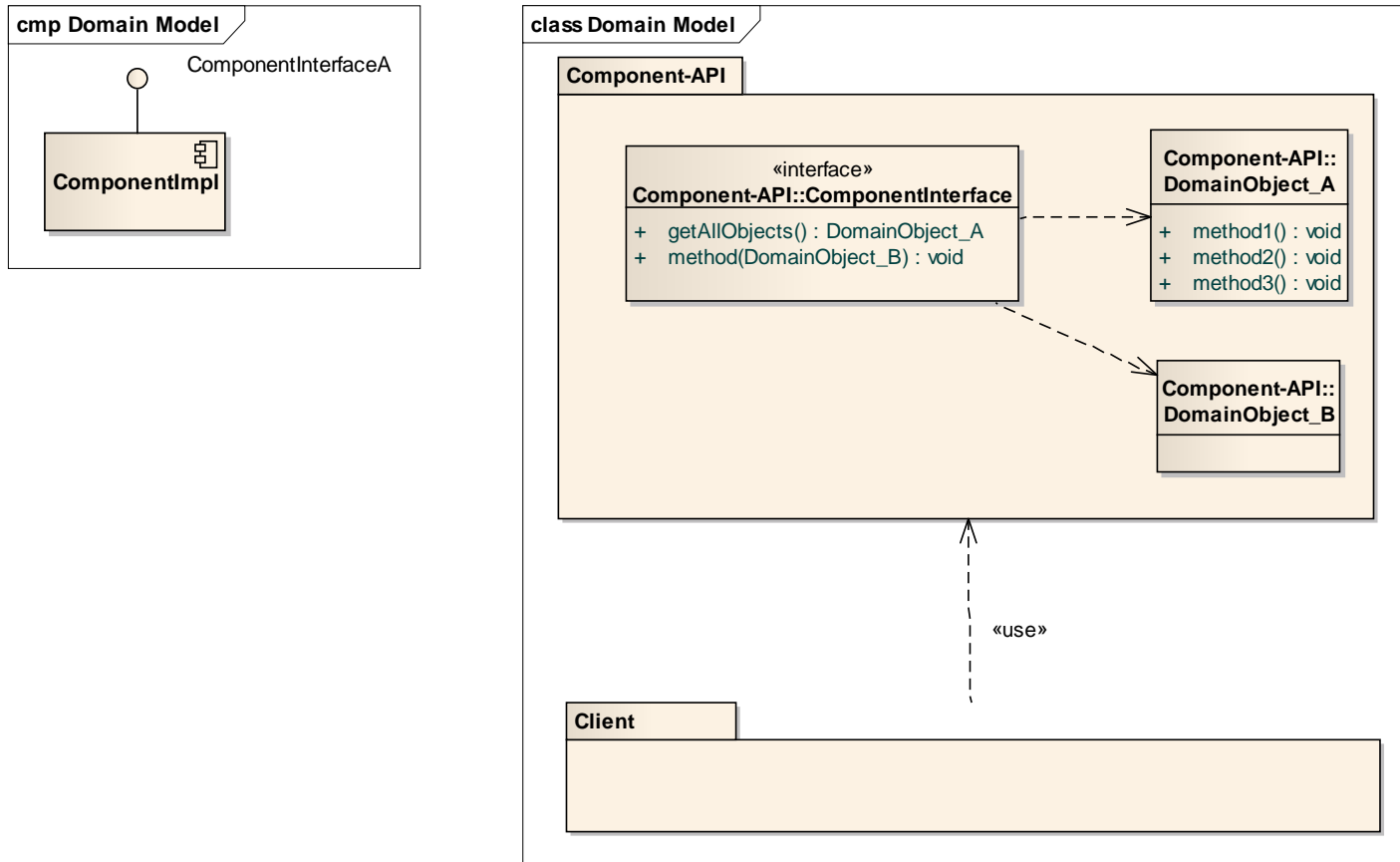


## Kopplung über Domänenobjekte

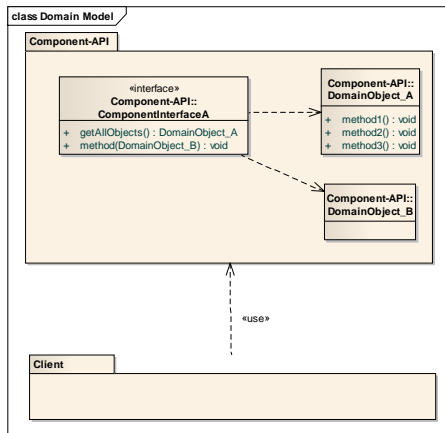
### Enge & Losekopplung

- Die direkte Übergabe von Domänenobjekten führt zu einer hohen Kopplung aufgrund des Lebenszyklus sowie im Bereich Information Hiding
- Begründung:
  - Domänenobjekte bieten einen umfangreichen Satz an Methoden um die Objekte zu verändern (setter).
  - Dies kann zu ungewollten Veränderung führen wenn ein Domänenobjekte über Komponentengrenzen hinweg verwendet wird. (Prinzip der Datenhoheit verletzt)
  - Ebenfalls wird über die getter evtl. zuviel Information preisgegeben (Informationhiding)

## Kopplung über Domänenobjekte



## Kopplung über Domänenobjekte



```
public interface ComponentInterface {

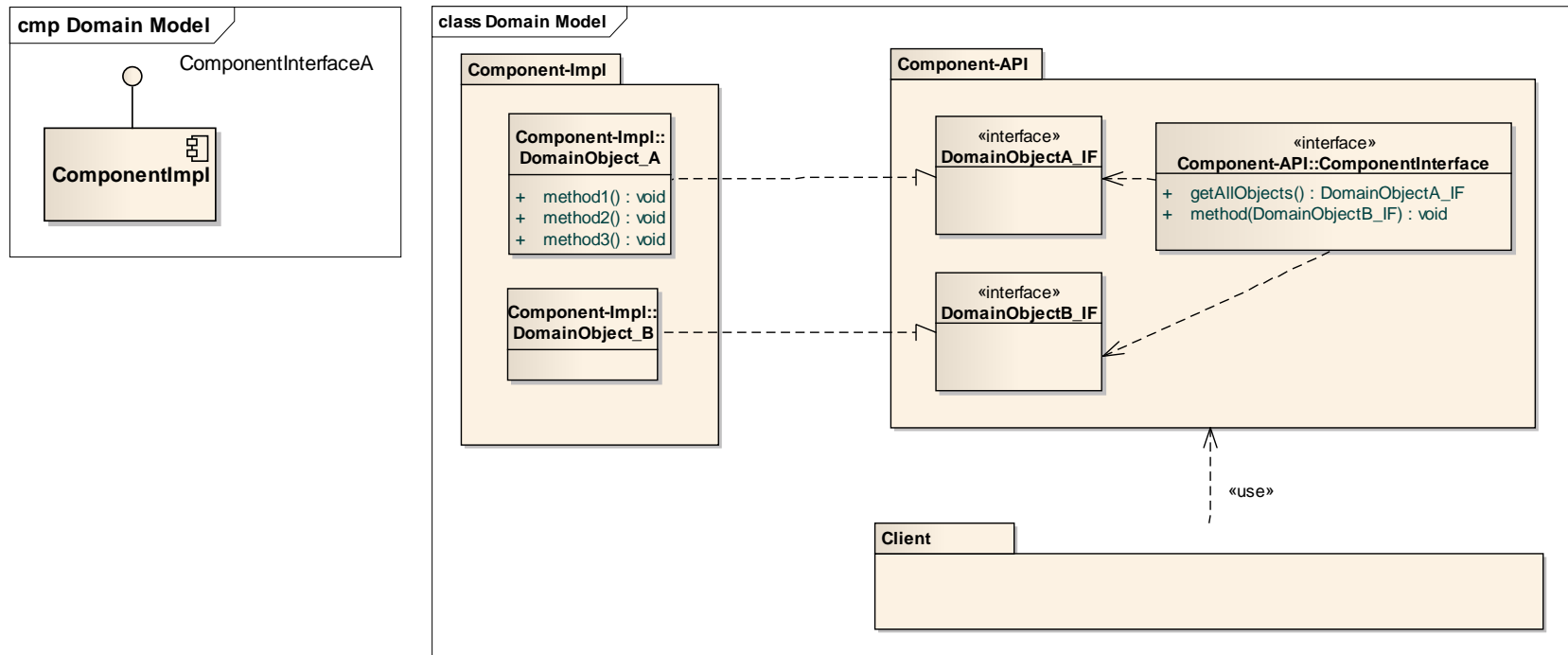
    public DomainObject_A getAllObjects();
    public void          method( DomainObject_B value );
}
```



## Kopplung über Entitätsschnittstellen

- Idee:
  - Verhindere den direkten Zugriff auf die Domänenobjekte.
  - Verstecke die Domänenobjekte hinter einer Schnittstelle (Entitätsschnittstelle) welche einen reduzierten Zugriff auf die Objekte darstellt
- Entitätsschnittstellen
  - Der Entitätstyp erhält eine eingeschränkte Sichtweise auf das referenzierte Objekt.
  - Die Einschränkung der Sichtweise erfolgt über eine Entitätsschnittstelle
  - Schnittstellen welche von den Entitätsklassen implementiert werden aber nur eine reduzierte (erlaubte) Teilmenge der Funktionen zur Verfügung stellen
  - Dadurch ist es möglich die Datenhoheit über die Entität zu beeinflussen

## Kopplung über Entitätsschnittstellen





### Kopplung über Entitätsschnittstellen

```
public interface ComponentInterface {  
  
    public DomainObjectA_IF getAllObjects();  
    public void          method( DomainObjectB_IF value );  
}
```

```
public interface DomainObjectA_IF {  
  
    public void method1();  
}
```

```
public class DomainObject_A implements DomainObjectA_IF {  
  
    ...  
}
```

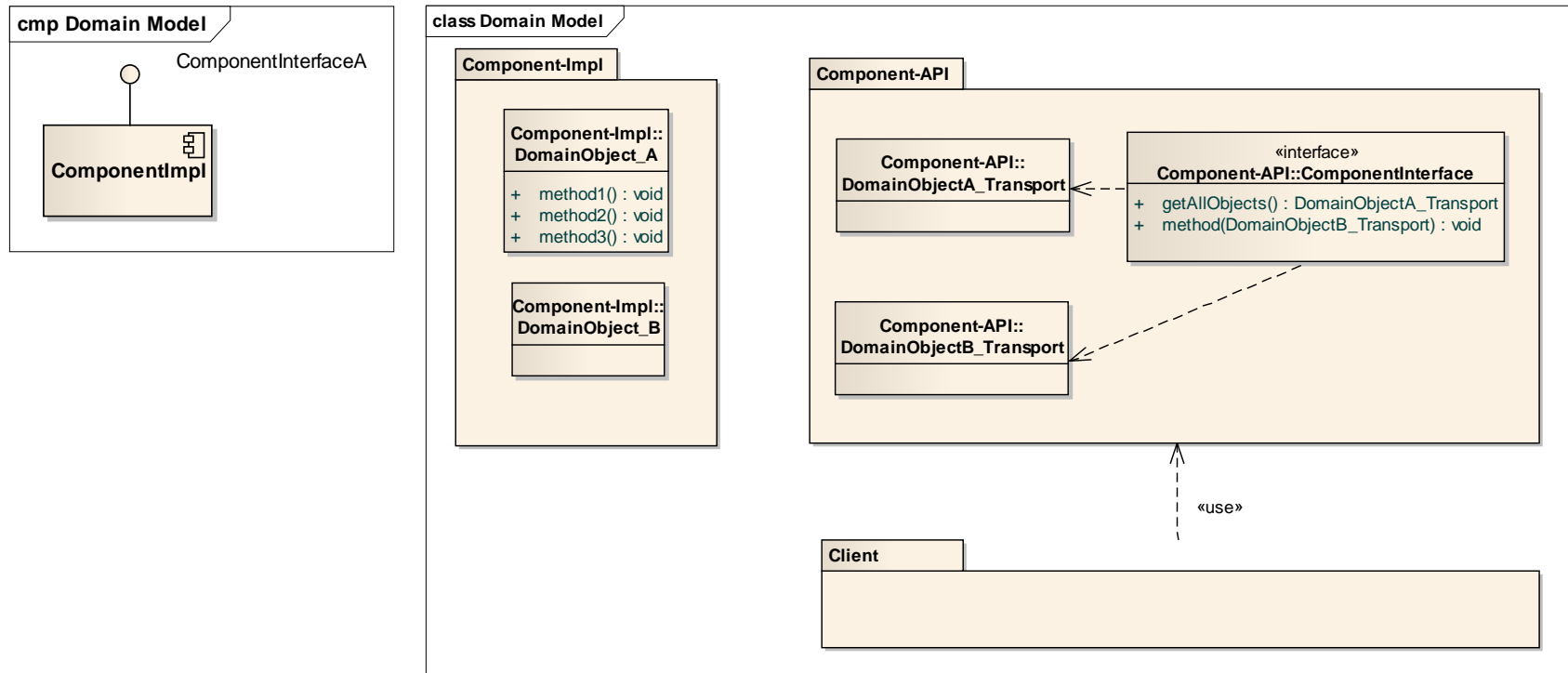




## Kopplung über Transportklassen

- Idee
  - Es werden niemals die eigentlichen Domänenobjekte nach außen weitergegeben.
  - Es werden Kopien (=Transportklasse) der Original-Objekte geliefert welche nur die beschreibenden Attribute enthalten
- Transportklassen
  - Eine Klasse welche identische Attribute der Entitätsklasse erhält
  - Alle Referenzen werden durch Fremdschlüssel dargestellt
  - Die Klasse enthält ansonsten keine Logik
  - Sind gut geeignet zum Transport der Objekte über Netzwerkgrenzen hinweg

## Kopplung über Transportklassen





### Kopplung über Entitätsschnittstellen

```
public interface ComponentInterface {  
  
    public DomainObjectA_Transport getAllObjects();  
    public void                      method( DomainObjectB_Transport value );  
}
```

```
public class DomainObjectA_Transport {  
  
    <attributes that are relevant to the enduser>  
}
```



### Implementierung des Produktmodells und Aufbau von Schnittstellen

#### Enge & Losekopplung – Objektorientierte u. Dienstorientierte Schnittstellen

Kategorie	Beispiel	Objektorientierte Schnittstelle	Dienstorientierte Schnittstelle
Transportklasse	DomainObjectTransport	nein	ja
Entitätsschnittstelle	DomainObjectIF	ja	Nein
Domainobject	DomainObject	ja	nein

Grundidee:

- Objektorientierte Schnittstellen enthalten Objektreferenzen auf Domain-Objects
- Dienstorientierte Schnittstellen enthalten nur Werte



## Implementierung des Produktmodells und Aufbau von Schnittstellen

### Enge & Losekopplung – Objektorientierte u. Dienstorientierte Schnittstellen

- Objektorientierte Schnittstellen bedeuten enge Kopplung
  - Der Importeur ist eng an den Exporteur gekoppelt
- Dienstorientierte Schnittstellen bedeuten lose Kopplung
  - Seiteneffekte sind ausgeschlossen da Objekte als Kopie (per Value) übergeben werden
  - Kopplungsform erfolgt über Datenobjekte



## Implementierung des Produktmodells und Aufbau von Schnittstellen

### Enge & Losekopplung – Objektorientierte u. Dienstorientierte Schnittstellen

- Die Kopplungsart zwischen verschiedenen Komponenten muss bewusst gewählt werden.
- Wichtig ist hierbei immer die Frage wie viel Kopplung kann eine Exporteur einem Importeur zumuten
- Empfehlung ( Teil 1):
  - Definieren Gruppen von eng gekoppelten Komponenten.
  - Komponenten verschiedener Gruppen werden lose gekoppelt
  - Falls Entscheidung nicht eindeutig sollte lose Kopplung gewählt werden



## Implementierung des Produktmodells und Aufbau von Schnittstellen

### Enge & Losekopplung – Objektorientierte u. Dienstorientierte Schnittstellen

Empfehlung (Teil 2):

- Objektorientierte Schnittstellen können über einen Adapter in dienstorientierte Schnittstellen konvertiert werden.
- D.h. beim Design kann man zuerst objektorientiert modellieren.
- Falls notwendig können über einen Adapter service-orientierte Schnittstellen definiert werden



### Implementierung des Produktmodells und Aufbau von Schnittstellen

#### Enge & Losekopplung – Objektorientierte u. Dienstorientierte Schnittstellen

- Objektorientierte Schnittstelle „ComponentInterfaceObjectOriented“

```
public interface ComponentInterfaceObjectOriented {  
    public DomainObject getDomainObject( ... );  
}
```





### Implementierung des Produktmodells und Aufbau von Schnittstellen

#### Enge & Losekopplung – Objektorientierte u. Dienstorientierte Schnittstellen

- Dienstorientierte Schnittstelle „ComponentInterfaceServiceOriented“

```
public interface ComponentInterfaceServiceOriented {  
  
    public DomainObjectTA getDomainObject( ...);  
}
```

```
public class DomainObjectTA {  
  
    ... One ore more attriutes ..  
  
    ... No methods ...  
}
```



### Implementierung des Produktmodells und Aufbau von Schnittstellen

#### Enge & Losekopplung – Objektorientierte u. Dienstorientierte Schnittstellen

```
public class DomainAdapter implements ComponentInterfaceServiceOriented {

    private ComponentInterfaceObjectOriented objectIf;

    public void setComponent ( ComponentInterfaceObjectOriented objectIf ) {
        this.objectIf = objectIf;
    }

    public DomainObjectTA getDomainObject( ... ) {

        DomainObject    objResult = objectIf.getDomainObject( ... );
        DomainObjectTA  srvResult = new DomainObjectTA();

        /* *****
        * Transformiere das objektorientierte Interface in
        * serviceorientiertes Interface
        * ***** */

        srvResult.<attribute> = objResult.get<Atr()>;

        return srvResult;
    }
}
```