Technische Hochschule Ingolstadt
Fakultät für Elektrotechnik und Informatik

Zukunft in Bewegung

Komponentenarchitektur mit Quasar

Architektur- und Entwurfsmuster der Softwaretechnik

Prof. Dr. Bernd Hafenrichter 06.03.2015

Software Kategorien – Wie findet man Komponenten

Problem: Es existiert kein Algorithmus der ein in System in Komponenten aufteilt

Ausgangspunkt:

- Analyse von Abhängigkeiten
- und Definition von Softwarekategorien

Informelle Definition von Softwarekategorien:

- Software wird verwendet um verschiedene Anwendungsbereiche abzudecken.
- z.B. Buchungen, Datenbankanbindung, Bildverarbeitung
- Die oben genannten Beispiele stellen Softwarekategorien dar.
- D.h. verschiedene Wissensgebiete welche durch Software abgedeckt wird

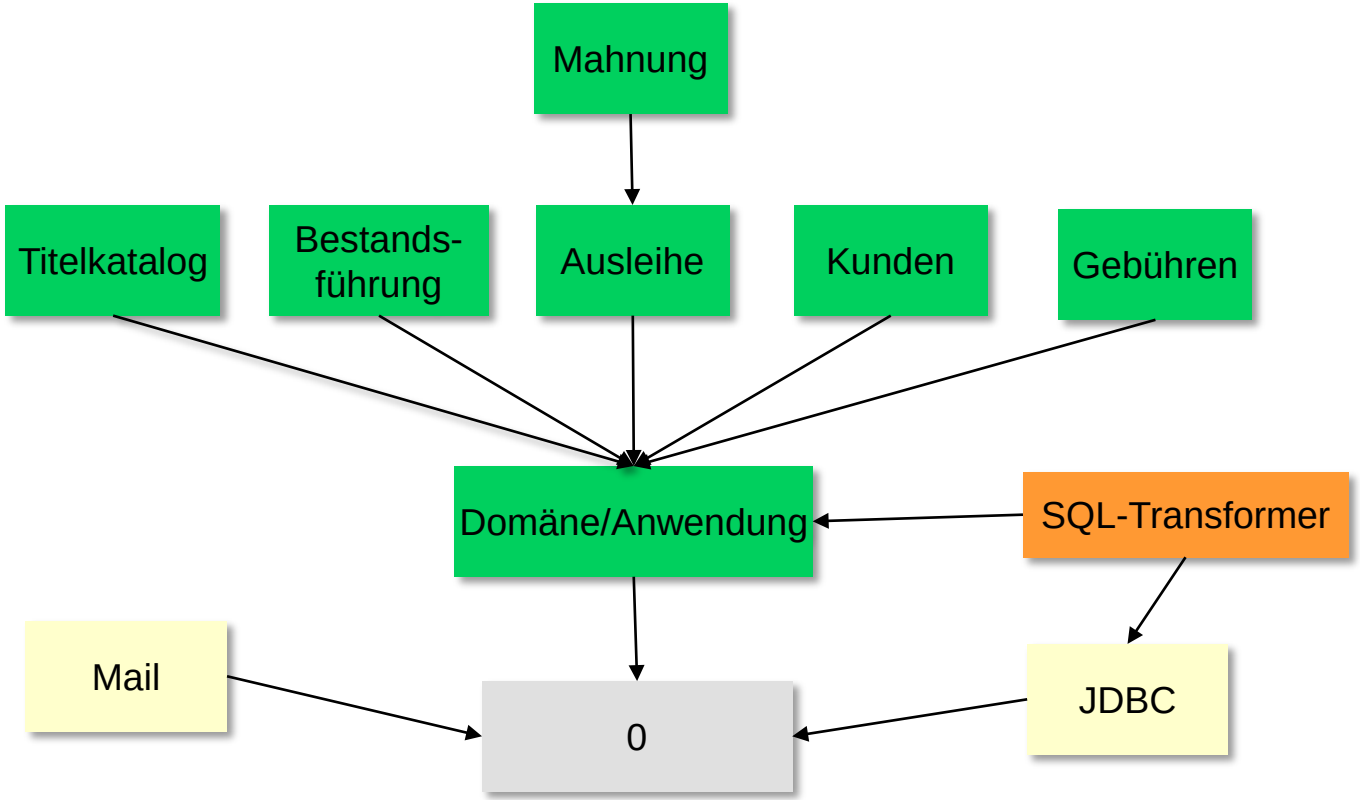
Software Kategorien – Wie findet man Komponenten

Grundlegende Einteilung in Kategorien

0	Allgemeingültige Weisheiten der Informatik – Optimale Wiederverwendbarkeit
A	Bestimmt durch die Applikation selbst (Kunden, Bestellungen, Lieferungen)
T	Bestimmt durch technische Schnittstellen (z.B. File, Datenbank, Swing)
AT	Vermischung zwischen Applikation & Technik
R	Representational Software (Beschränkte/milde Form von AT)

Software Kategorien – Wie findet man Komponenten

Beispiel: Softwarekategorien einer Bibliotheksapplikation



Software Kategorien – Wie findet man Komponenten

Problem: Es existiert kein Algorithmus der ein in System in Komponenten aufteilt

Vorgehensweise:

- Schritt 1) Definiere Softwarekategorien
- Schritt 2) Definiere die Abhängigkeiten zwischen verschiedenen Kategorien
- Schritt 3) Zerlege das System in Komponenten auf Basis der Kategorien

Hinweise:

- Softwarekategorien stellen die Vorstufe zu einer sinnvollen Komponentenstruktur dar
- Die Suche nach Kategorien ist einfacher als die Suche nach Komponenten
- Die Kontrolle der Abhängigkeiten ist enorm wichtig um das System planbar und flexibel Weiterentwickeln zu können.

Software Kategorien – Wie findet man Komponenten

Softwarekategorien

Der Nutzen von Softwarekategorien:

- Softwarekategorien dienen als Richtschnur beim Entwurf von Komponenten
- Dienen als Kontrollinstanz bei zukünftigen Änderungen
- Verbessertes Know-How-Management: Bei der Entwicklung ist es möglich sich auf einzelne Sachgebiete des Wissens zu konzentrieren

Regeln zur Bildung von Softwarekategorien

Prinzipien:

- Kategorien sind teilgeordnet
- Jede Kategorie kann eine oder mehrere Kategorien verfeinern
- Der so entstehende Kategoriengraph ist zyklenfrei
- Die Wurzel bildet die Kategorie 0 – das Allgemeinwissen
- Ein Kategorie a heißt **rein**, wenn es von a genau einen Weg zu Kategorie 0 gibt
- Ansonsten heißt die Kategorie **unrein**

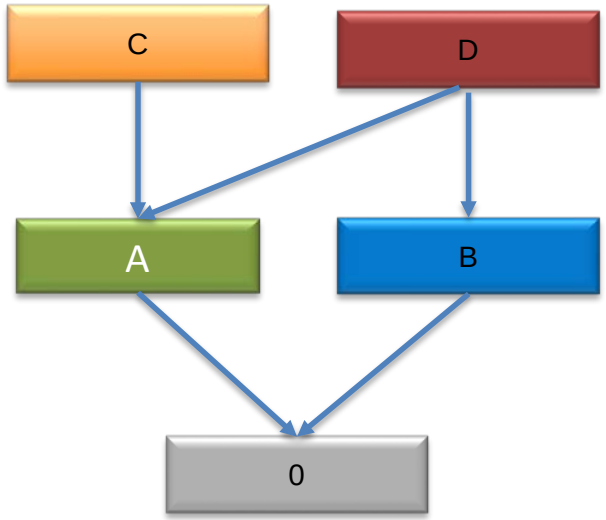
Software Kategorien – Wie findet man Komponenten

Regeln zur Bildung von Softwarekategorien

Prinzipien:

- Reine Kategorien entstehen durch stückweise Verfeinerung von Kategorie 0
- Jede einfache Komponente (= Modul) Schnittstelle und Klasse/Datentypen sollte genau zu einer reinen Kategorie gehören (zu möglichst wenigen)

Regeln zur Bildung von Softwarekategorien



Reine Kategorien: A, B, C

Unreine Kategorien: D

Software Kategorien – Wie findet man Komponenten

Die Kategorie 0

Prinzipien:

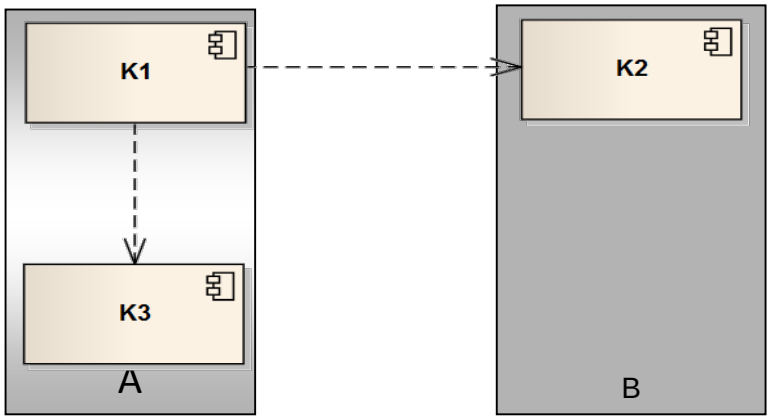
- 0-Software ist Allgemeingut und überall vorhanden
- Sie enthält ewige Wahrheiten der Informatik
- 0-Software ist global
- Eine Softwarebaustein gehört in Kategorie 0 wenn er sich nur mit geringer Wahrscheinlichkeit ändert und ausschließlich 0-Software enthält
- **Die Kategorie-0 ist besonders wichtig für Schnittstellen. Sie erlaubt die Kopplung zwischen Komponenten verschiedener Kategorien**
- **Die verwendeten Datentypen in einer Schnittstellen müssen genauso allgemeingültig sein**

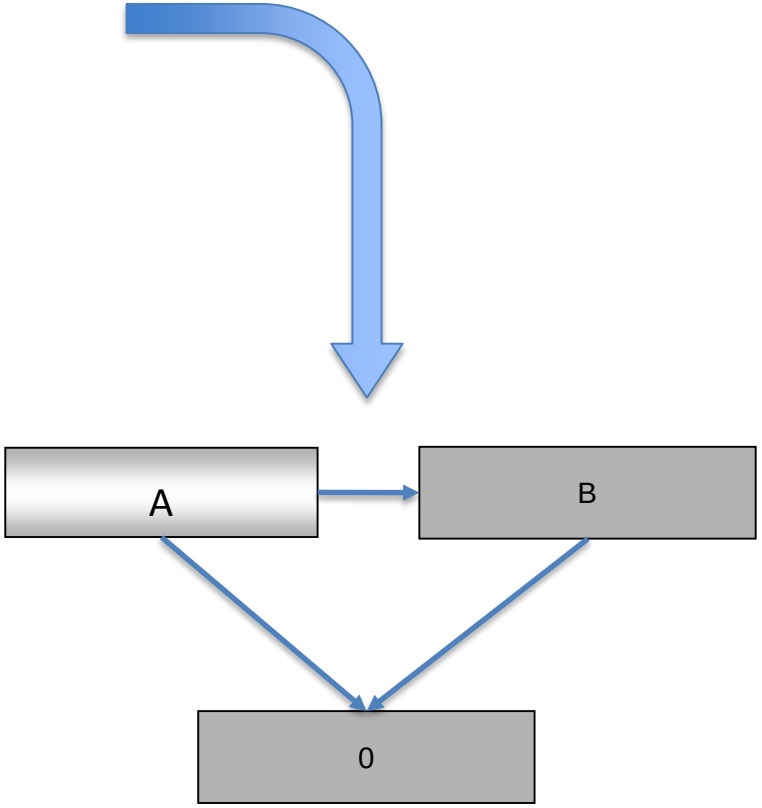
Kommunikation zwischen Komponenten verschiedener Kategorien

Prinzipien:

- Wenn eine Komponente K_1 eine Komponente K_2 aufruft erwirbt K_1 die gleiche Kategorie wie K_2
- Das ist kein Problem wenn K_1 die gleiche Kategorie wie K_2 hat oder eine Verfeinerung von K_2 ist
- Andernfalls entsteht eine unerwünschte Mischform, da Wissensgebiete vermischt werden. Dies führt zu einer unsauberen Softwarestruktur und hohen Abhängigkeiten

Kommunikation zwischen Komponenten verschiedener Kategorien

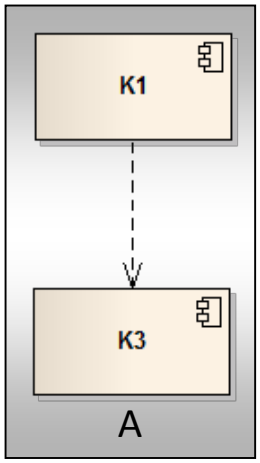


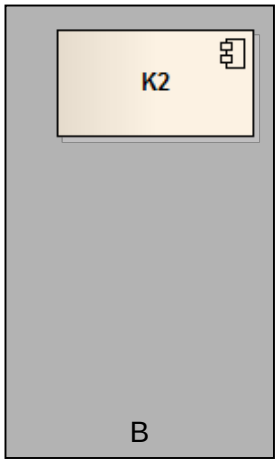


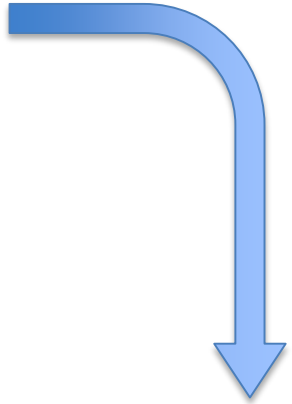
Bewertung:

- Der zugehörige Kategoriegraph enthält die unreine Kategorie “A”.
- Komponentenarchitektur benötigt zum Verständnis mehr Wissen

Kommunikation zwischen Komponenten verschiedener Kategorien

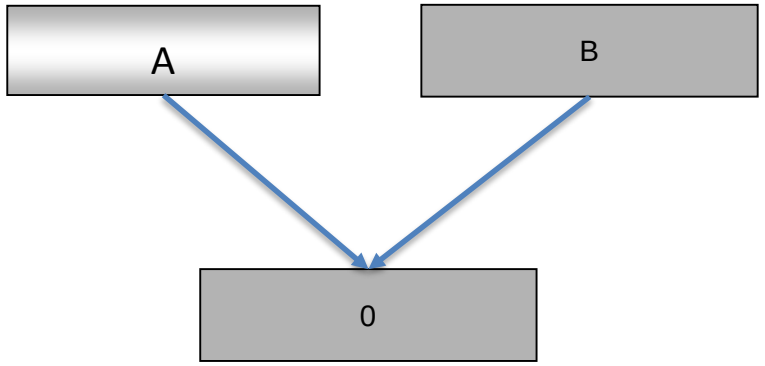






Bewertung:

- Der zugehörige Kategoriegraph enthält nur reine Kategorien.



Kommunikation zwischen Komponenten verschiedener Kategorien

Ein schlechtes Beispiel für AT-Software

```
public void increaseSalary( double faktor ) {

  Connection        con  = getConnection();
  PreparedStatement stmt = con.prepareStatement("select * from mitarbeiter");
  ResSet            resSet  = stmt.executeQuery();

  while( resSet.next() ) {

    String maID = resSet.getString(1);
    String gehalt = resSet.getDouble(2);

    gehalt = gehalt * faktor;

    con.executeSQL( "update mitarbeiter set gehalt = " + gehalt +
                    " where maid = " + maID );
  }

  resSet.close();
}
```

Kommunikation zwischen Komponenten verschiedener Kategorien

Verbesserte Version

```
public void increaseSalary( double faktor ) {

  List<Mitarbeiter> mitarbs = persistence.getMitarbeiter();

  for( Mitarbeiter ma : mitarbs ) {
      ma.setGehalt( ma.getGehalt() * faktor );
      persistence.update( ma );
  }
}
```

- Der entstandene Code ist wesentlich übersichtlicher
- Der Anwendungscode ist unabhängig von „SQL“ bzw. technischen Wissen
- Die Persistenzschicht kann problemlos ausgetauscht werden

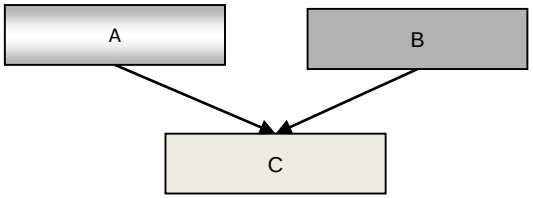
Kommunikation zwischen Komponenten verschiedener Kategorien

Sichtbarkeitsregel:

- Software einer hohen Kategorie darf nur auf Software einer niedrigeren Kategorie zugreifen (Vorgänger im Kategoriegraphen)
- Formal: Es sei K eine Komponente der Kategorie a. K darf Schnittellen einer andere Kategorie b gdw. exportieren/importieren wenn a eine Verfeinerung von b ist.

Erlaubt

K_1 (A) → K_2(C)



Problematisch

K_1 (A) → K_2(B)

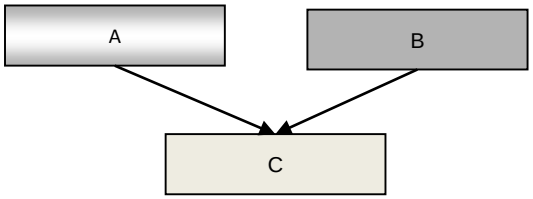
Kommunikation zwischen Komponenten verschiedener Kategorien

Sichtbarkeitsregel:

- Komponenten dürfen kommunizieren sobald sie eine gemeinsame Gesprächsebene haben
- Umgangssprachlich: Versuch die Kommunikation möglichst allgemeine zu halten
- Achtung: Schnittstellen der Kategorie 0 stellen somit Allgemeingut dar

Erlaubt

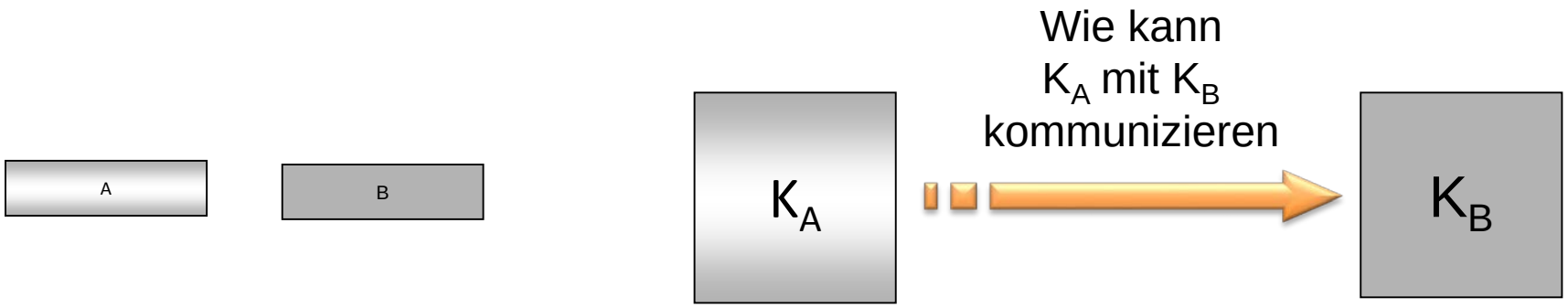
K_1 (A) → K_2(C)



Problematisch

K_1 (A) → K_2(B)

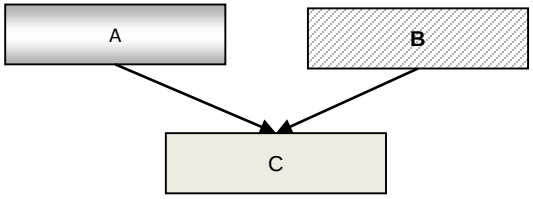
Kommunikation mit neutralen Schnittstellen

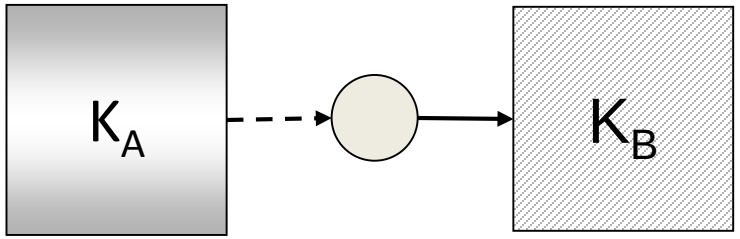


Kommunikation mit neutralen Schnittstellen

Idee:

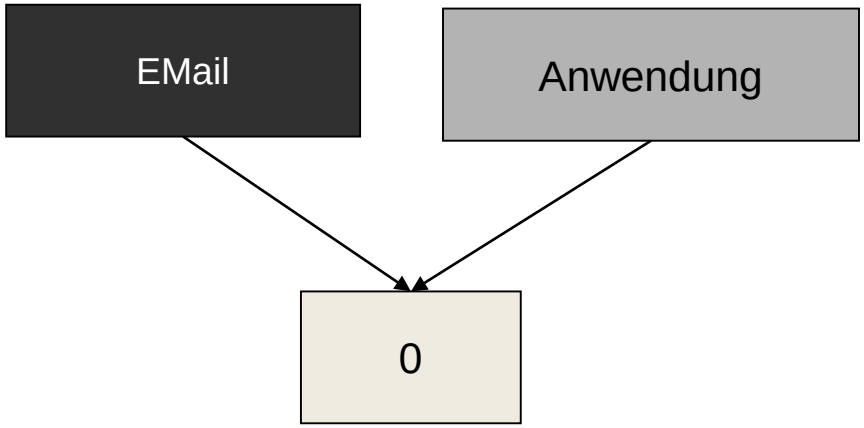
- Definiere eine Schnittstelle in Kategorie C welche von A und B verfeinert
- Im allgemeinsten Fall ist das Kategorie „0“
- Die Parametertypen sind so zu definieren das nur allgemeine Typen verwendet werden. (Parametertypen die in C oder einem Vorgänger liegen)
- Es dürfen keine Details der beteiligten Komponenten A und B verwendet werden
- Gesichtspunkt „Gemeinsame Gesprächsbasis“





Kommunikation mit neutralen Schnittstellen

Beispiel: Versende Emails aus einer Anwendung



Kommunikation mit neutralen Schnittstellen

Beispiel: Versende Emails aus einer Anwendung

```
public interface Mail {

   public void sendMail( String empfaenger,
                         String sender,
                         String betreff,
                         Sring body );
}
```

Kategorie 0

Kommunikation mit neutralen Schnittstellen

Beispiel: Versende Emails aus einer Anwendung

```
public class MailImp implements Mail {

  public void sendMail( String empfaenger,
                        String sender, String betreff, Sring body ) {
    // versende die EMails
    // über die Java-API
  }

}
```

Kategorie EMail

Kommunikation mit neutralen Schnittstellen

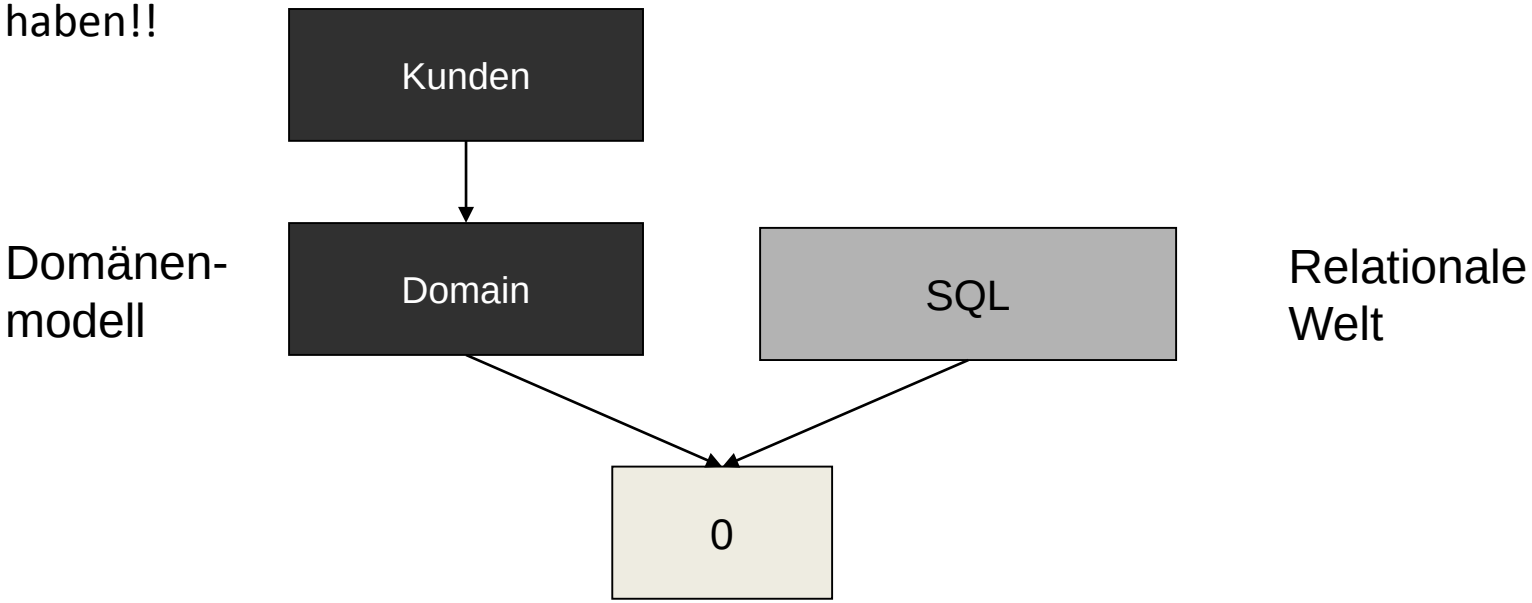
Beispiel: Versende Emails aus einer Anwendung

```
/*
 * Einfache Variante der Konfiguration.
 * Verbinde Interface mit der Componente
 */

public static EMail getEMailComponent() {
  return new EMailImpl();
}

public static void main( String [] args ) {

   EMail email = getEMailComponent();

   email.sendMail( "x@y.de", "x@y.de",
                   "Nachricht aus der Anwendung", "Wie gehts");
}
```

Kommunikation mit R-Software

Beispiel: Anwendungen und Persistenz

- Wie kann das Kundenmodul seine Daten in einer Datenbank abspeichern?
- Problem wie wird das Domänenmodell auf die relationale Welt übersetzt?
- Eine 0-Schnittstelle soll ja nur Allgemeingut enthalten, also kein SQL-Kenntnisse haben!!

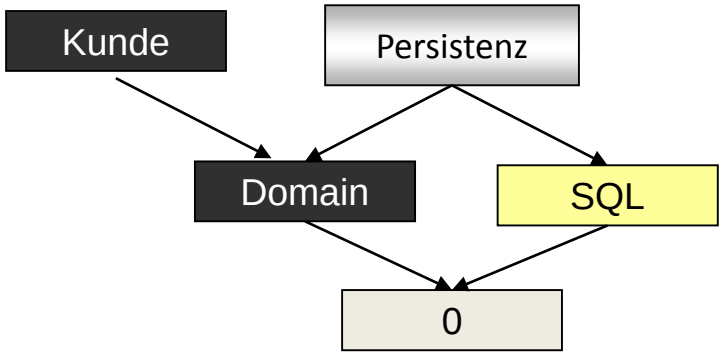


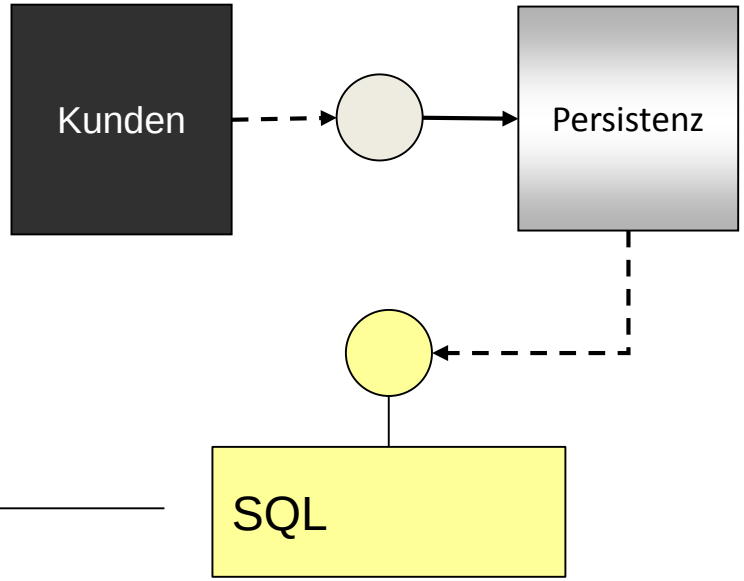
Kommunikation mit R-Software

Lösungsansatz:

- Verwende Transformationssoftware um die Daten zwischen verschiedenen Welten zu „transformieren“
- R-Software: Software die nur transformiert aber keine fachlichen Aufgaben übernimmt
- Die Vermengung der Kategorien ist ausschließlich an dieser Stelle erlaubt.
    - R-Software kann generiert werden
    - R-Software kann extern definiert werden

Kommunikation mit R-Software





Der Persistenz (Transformator) kennt sowohl
- Domänenmodell
- SQL-Objekte
- aber keine Fachlogik

Kommunikation mit R-Software

Beispiel: Persistenz

```
public interface Persistenz {

    public Object lookup( String key, Class type );

    public void store( Object object );

}
```

Kategorie 0

Kommunikation mit neutralen Schnittstellen und R-Software

Beispiel: Persistenz

```
public class PersistenzImpl implements Persistenz {

  @Override
  public Object lookup(String key, Class type ) {

    SQLObject sqlObj = loadFromDB( type );

    return translateToDomainObject( sqlObj );
  }

  @Override
  public Object store(Object object) {

    SQLObject sqlObj = translateFromDomain( object );

    storeToDB( sqlObj );
  }
}
```

Kategorie Persistenz

Kommunikation mit neutralen Schnittstellen und R-Software

Beispiel: Persistenz

```
public class Example {

  public static void main(String[] args) {

    Persistenz    persistenz   = new PersistenzImpl();
    Kunde     kunde        = null;

    kunde = (Kunde)persistenz.lookup( "AB12“, Kunde.class );

    kunde.setName( "Neuer vorname" );

    persistenz.store( kunde );
  }
}
```

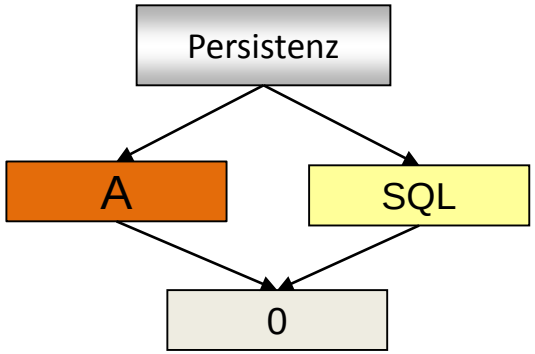
Die Verwendung

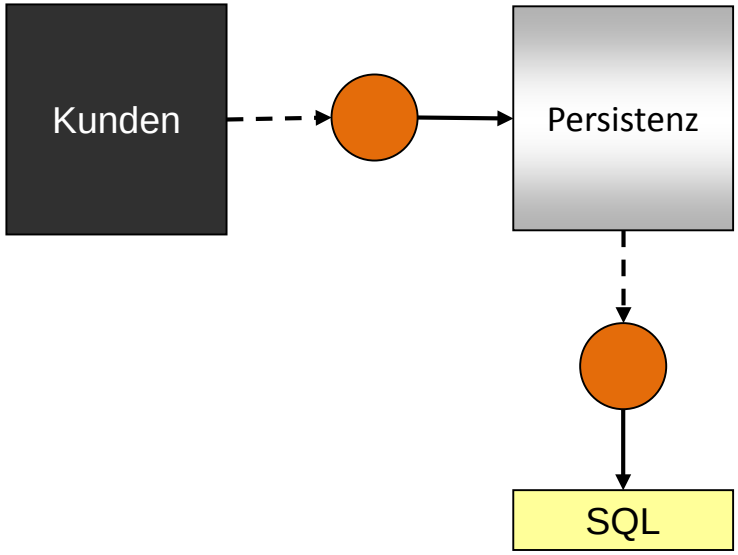
Kommunikation zwischen Komponenten

Problem:

- Die Kommunikation mit R-Software über neutrale Schnittstellen die in Kategorie 0 liegen bedeutet dass eine direkte Übergabe von Domänen-Objekten nicht möglich ist
- Es müssen stets allgemeine Datentypen z.B. „Object“, Maps, Strings verwendet werden.
- Dies führt in der Regel zu aufwendigen Konvertierungen und Code der nicht typsicher ist
- Es sind Cast-Operationen nötig um zwischen den verschiedenen Datentypen zu konvertieren.

Kommunikation über das Domänenmodell und R-Software





Kommunikation mit R-Software

Beispiel: Persistenz

```
public interface Persistenz {

    public Kunde loadKunde( String key);

    public void store( Kunde object );

}
```

Kategorie A

Kommunikation mit neutralen Schnittstellen und R-Software

Beispiel: Persistenz

```
public class PersistenzImpl implements Persistenz {

  SQLTransformer transformer;

  @Override
  public Kunde loadKunde( String key) {

    SQLObject sqlObj = loadFromDB( type );

    Kunde kunde = new Kunde();

    kunde.set<Attribute>( sqlObj.get<Attribute>() );

    return kunde ;
  }

  @Override
   public void store( Kunde object ) {

    …
  }
}
```

Kategorie Persistenz

Kommunikation mit neutralen Schnittstellen und R-Software

Beispiel: Persistenz

```
public class Example {

  public static void main(String[] args) {

    Persistenz    persistenz   = new PersistenzImpl();
    Kunde         kunde        = null;

    kunde = persistenz.lookup( "AB12” );

    kunde.setName( "Neuer vorname" );

    persistenz.store( kunde );
  }
}
```

Die Verwendung