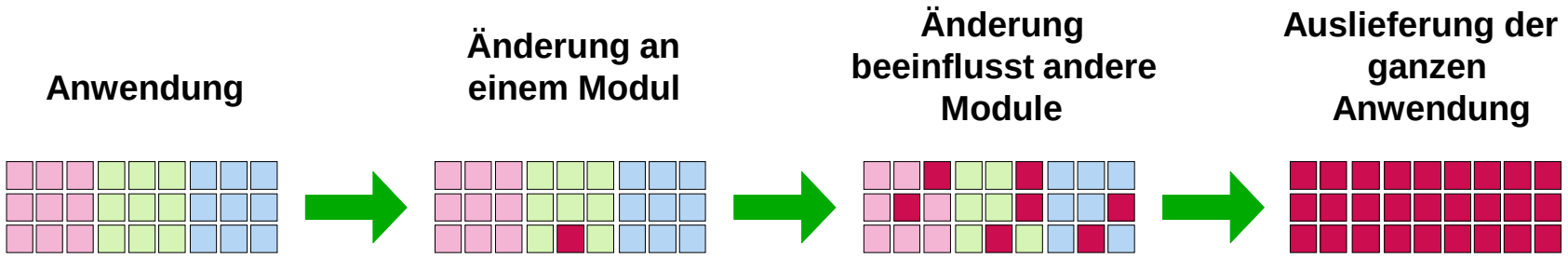
Programmieren 3

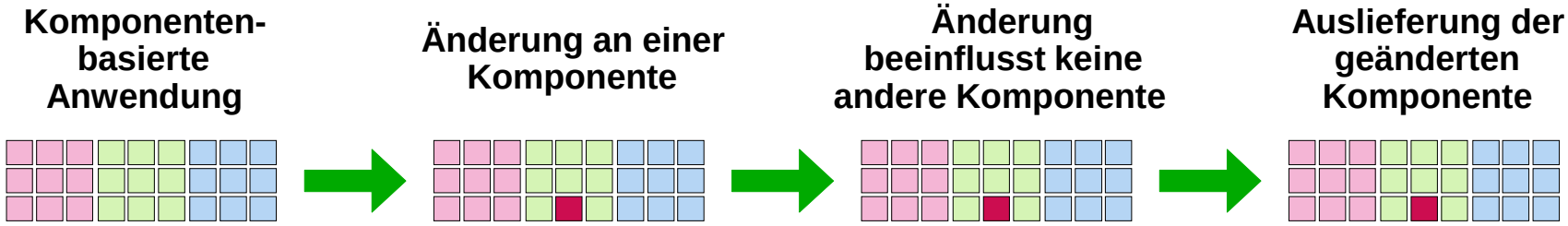
Kapitel 4: Komponenten und Schnittstellen

- Mail: eine kleine Komponente
- Konfiguration mit Fabriken
- Paketstruktur
- Sichten auf Komponenten
- Beispiel für Komponenten

Software-Komponenten: Motivation



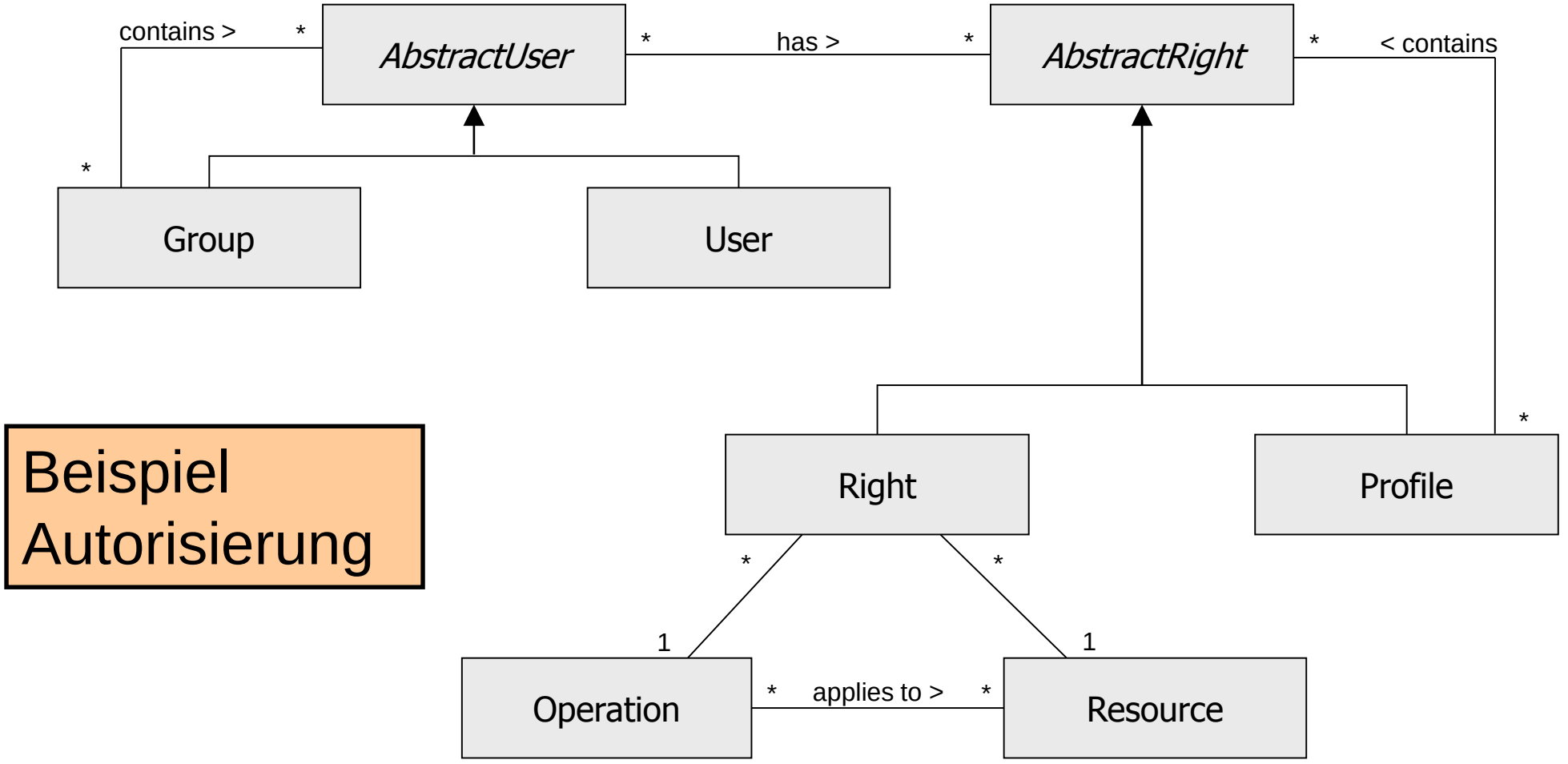
Ziel: Änderungen an nur wenigen Stellen im Programmcode und nicht über den globalen Zustandsraum der Software-Anwendung verteilt!



Wartbarkeit!

[Quelle: Business Component Factory; Herzum, Sims]

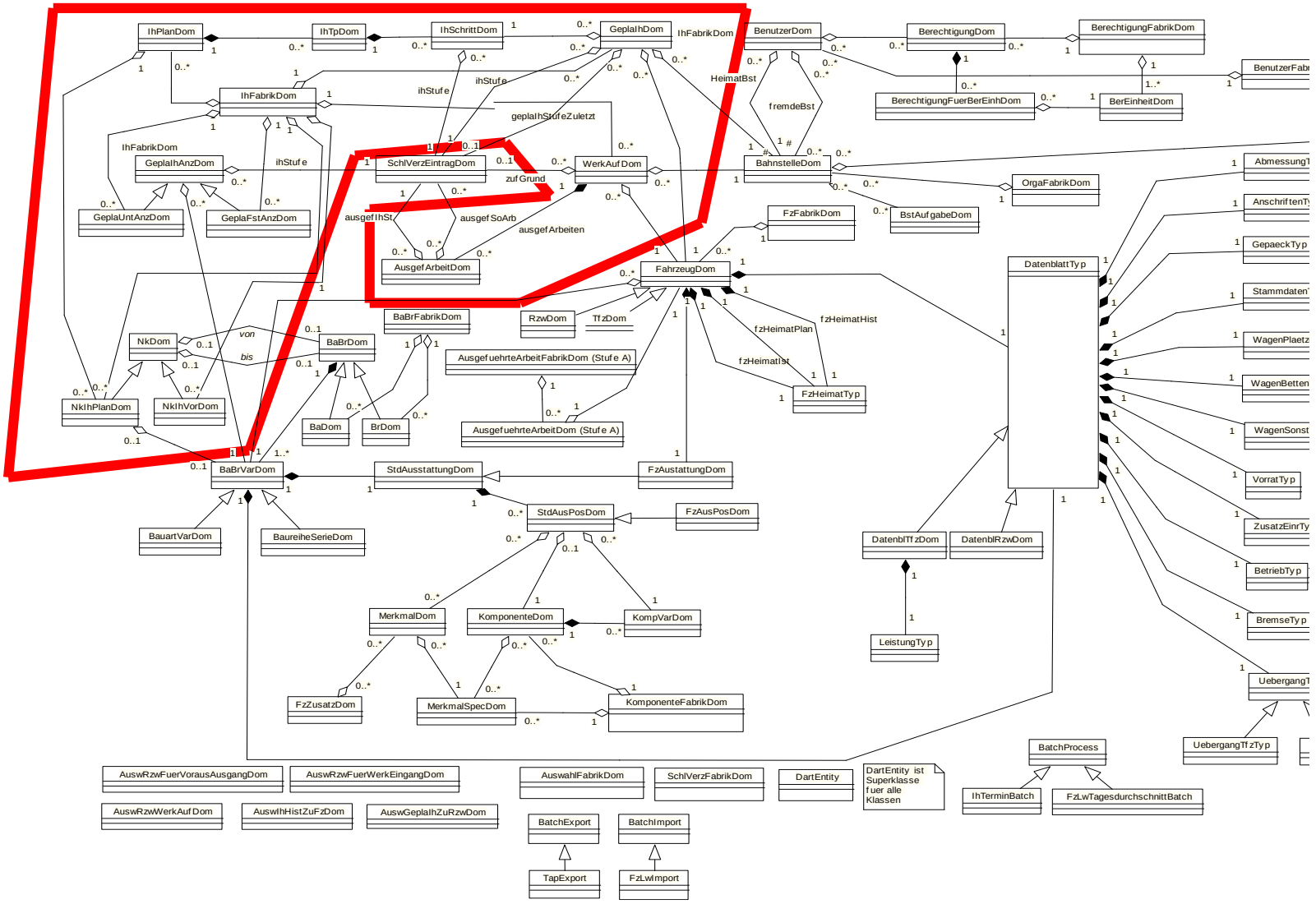
Objektorientierung funktioniert im Kleinen ganz gut.



Beispiel Autorisierung

Objektorientierung skaliert schlecht.

Wo ist die Instandhaltung ???

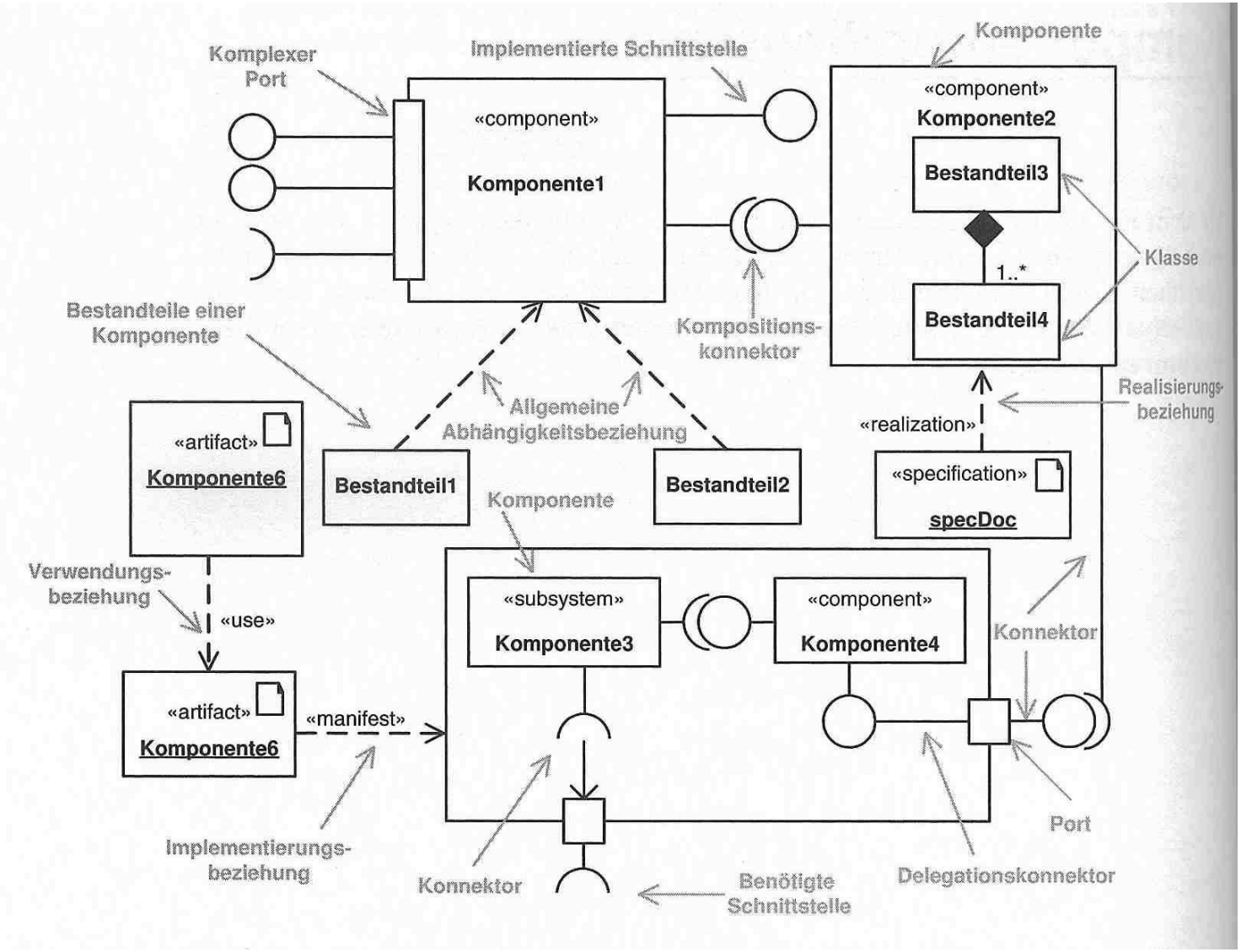


Merkmale einer Komponente

- Sie **exportiert eine oder mehrere Schnittstellen**, die im Sinn eines Vertrags garantiert sind. Dazu gehört insbesondere die genaue Semantik der Schnittstellen.
- Sie **importiert andere Schnittstellen**. Der Import einer Schnittstelle bedeutet, dass die Komponente die Methoden dieser Schnittstelle benutzt. Sie ist erst lauffähig, wenn alle importierten Schnittstellen zur Verfügung stehen, und dies ist Aufgabe der *Konfiguration.*
- Sie **versteckt die Implementierung** und kann durch andere Komponenten ersetzt werden, die dieselbe Schnittstelle exportieren
- Sie eignet sich als **Einheit der Wiederverwendung**, denn sie kennt nicht die Umgebung, in der sie läuft, sondern macht nur minimale Annahmen über die Umgebung
- Sie **kann andere Komponenten enthalten**. Man kann neue Komponenten aus vorhandenen Komponenten zusammensetzen (oder komponieren), und dies über beliebig viele Stufen.
- Sie ist neben der Schnittstelle die **wesentliche Einheit des Entwurfs, der Implementierung und der Planung**

Kurzer Exkurs: Komponentendiagramm in UML

- Elemente im Komponenten-diagramm
  - Komponente
  - Schnittstelle
  - Klasse
  - Artefakt
  - Port
  - Realisierung-, Implementierungs-, Verwendungs-beziehung



[UML glasklar, dpunkt.verlag]

Begriffe im Komponentendiagramm

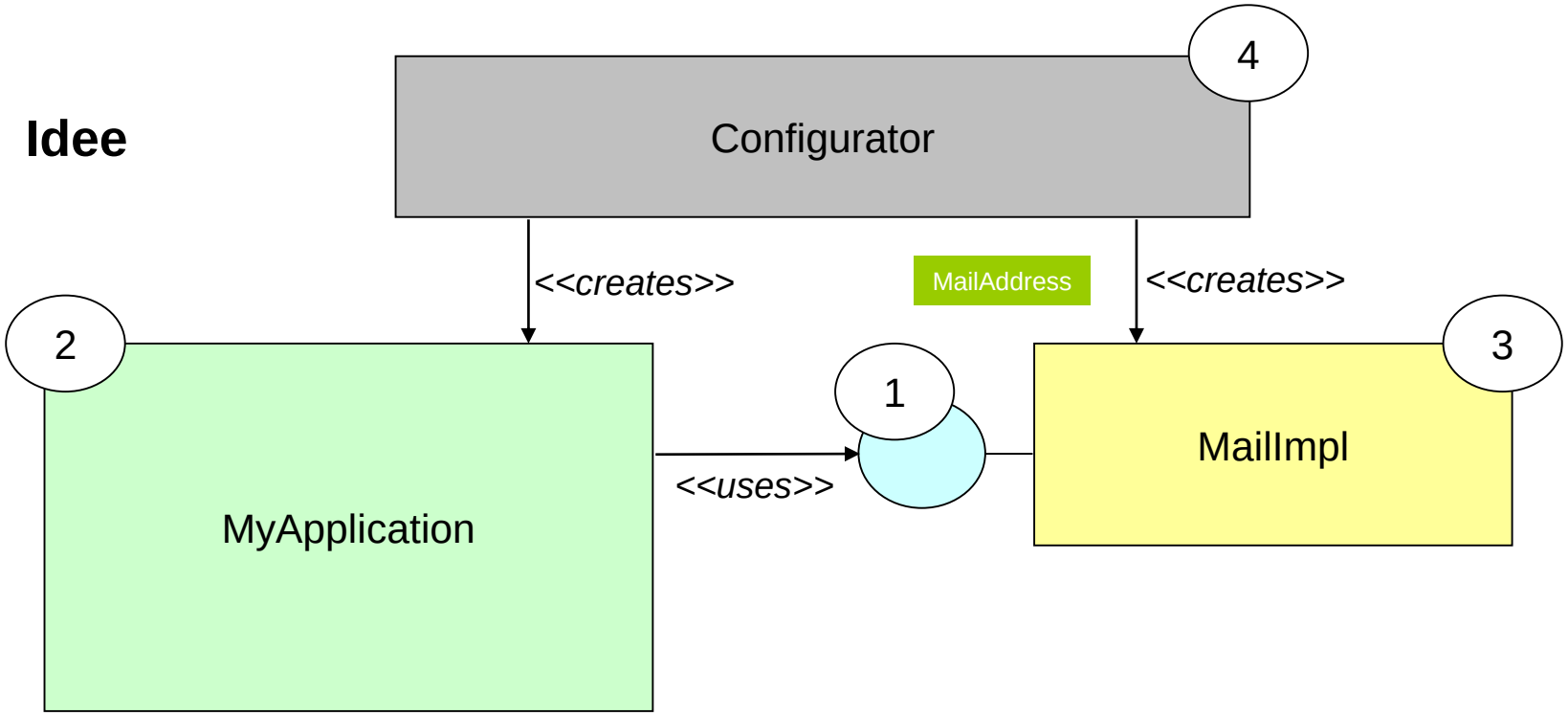
- Artefakt
  - physische Informationseinheit (Modell, Quellcode, Skript, E-Mail, Dokument, …)
  - Stereotypen: <<file>>, <<document>>, <<executable>>, <<source>> <library>>
  - Bsp: XML Deployment Deskriptor
- Port
  - Kanal über den Kommunikation ausgeführt wird
  - physikalische Verbindung zwischen den Einheiten (z.B. Kabel, Infrarot, Video-out)
  - kann Zusatzaufgaben wie Filterung, Caching, Protokollierung übernehmen
- Realisierungsbeziehung <<realize>>
  - Verbindung zwischen Spezifikation/Schnittstelle und Implementierung
- Verwendungsbeziehung <<use>>
  - abhängige Modellelemente die für die korrekte Funktion notwendig sind
- Implementierungsbeziehung <<manifest>>
  - Verbindet Artefakt mit realisierender Komponente

Mail – eine kleine Komponente

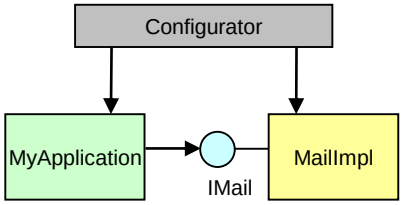
Anforderung

- Eine Anwendung soll Mails an eine feste Adresse senden.
- Keine Verteiler, kein Empfang, keine Anhänge, keine Verwaltung der versandten Mails.

Idee



Mail: Schnittstelle und Aufruf



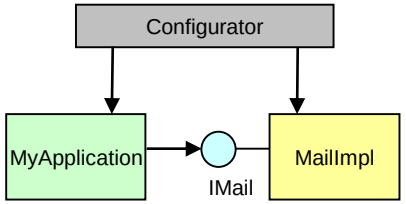
1

```
public interface IMail {
        void send(String subject, String content);
}
```

2

```
import mail.IMail;
public class MyApplication {
        private IMail mail;
        public MyApplication( .., IMail m, ..) {
                mail = m;
                ...
        }
        ..
        public void start() {
                ...
                mail.send("Hi Otto", "alles klar?");
        }
}
```

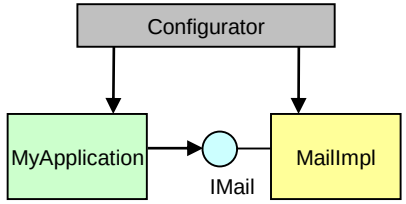
Mail: Implementierung



3

```
import mail.IMail;
import mail.MailAddress;

// import spezieller Mail-Klassen

public class MailImpl implements IMail {
        private MailAddress mailAddress;

        public MailImpl(MailAddress address) {
                mailAddress = address;
        }

       public void send(String subject, String content) {
                // komplizierte Aufrufe an die echte Mail
        }
}
```

Mail: Konfiguration

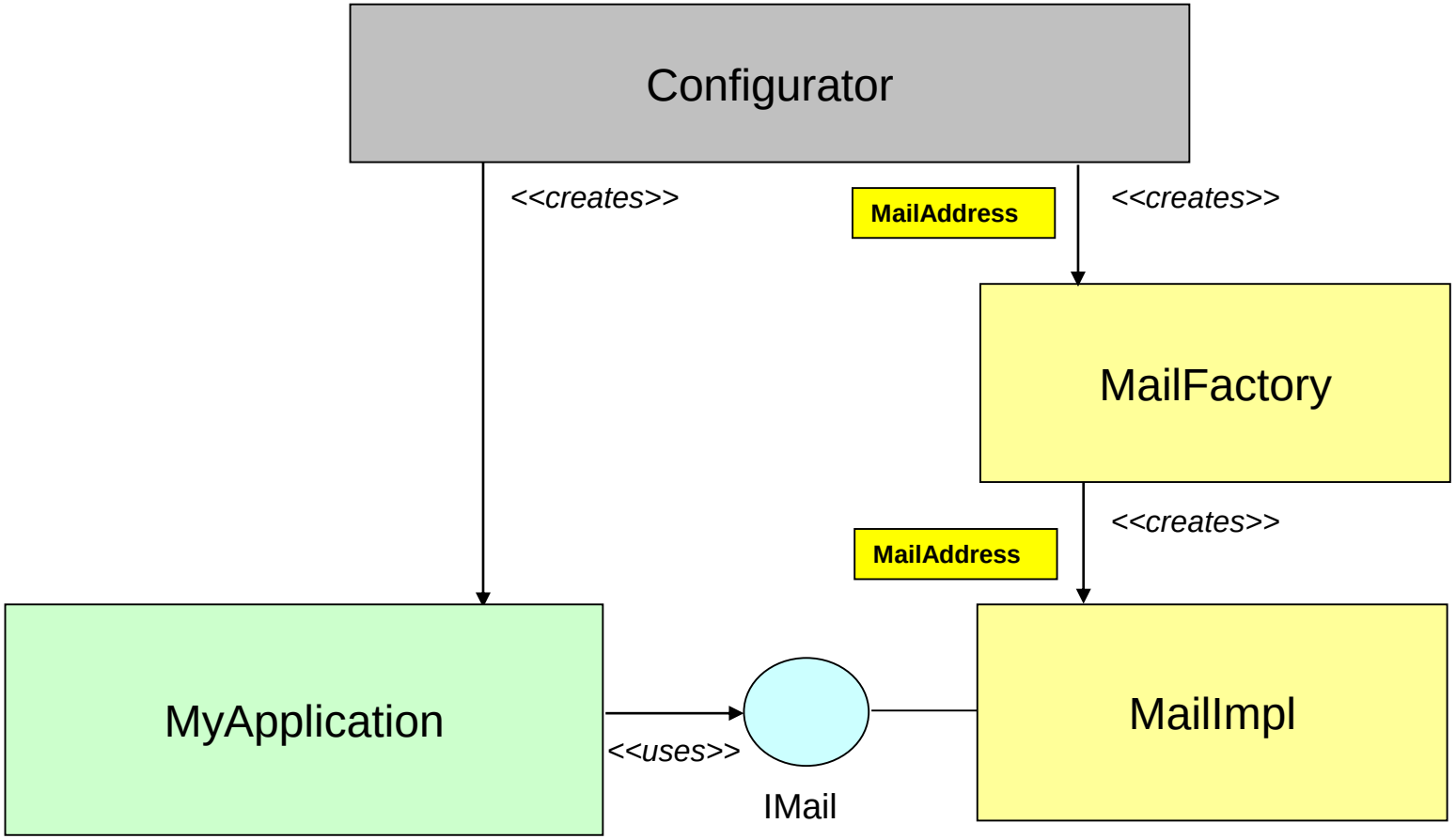



```
import ...mail.MailImpl;
import ...MyApplication;

public class Configurator {
    public Configurator( .. ) {
        IMail mail = new MailImpl(...);
        MyApplication appl = new MyApplication(mail);
        appl.start();
    }
    ..
}
```

4

- **Was geschieht, wenn eine andere Mail-Implementierung genutzt werden soll?**
  - Bei einer Applikation: Änderung in einer Konfiguration
  - Bei mehreren Applikationen: Änderung in mehreren Konfigurationen
  - Konfiguration gehört jedoch nicht zur Komponente Mail

Mail mit konkreter Fabrik



Eine konkrete Mail-Fabrik

4

```
import ...mail.MailImpl;
import mail.MailAddress;

public class MailFactory {
    private MailAddress address;
    public MailFactory(MailAddress address) {this.address = address;}
    public IMail getMail( ) {
        return new MailImpl(address);
   }
    ..
}
```

Wozu ist das gut?

a) Fabrik ist Indirektion zwischen Caller und Implementierung:
Wahl der Implementierung nur an einer Stelle

b) Caller importiert nur Fabrik, nicht die Implementierung

c) Konstruktoren funktionieren nicht über RMI, CORBA.

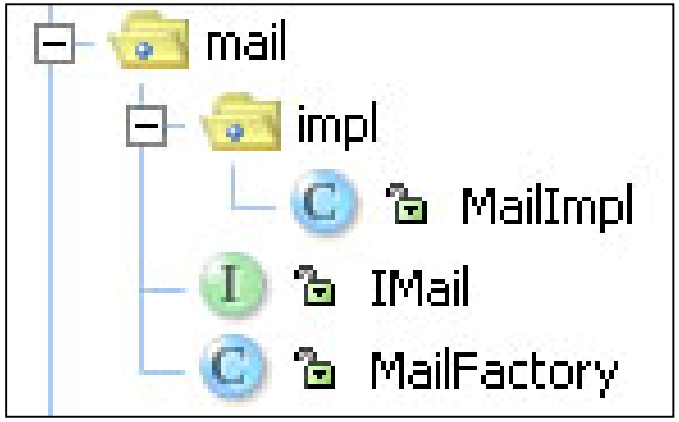
Konfiguration mit konkreter Fabrik

```
import mail.IMail;
import mail.MailFactory;
import mailapplication.impl.MyApplication;

public class Configurator {

    public Configurator(...) {
        MailFactory mailFactory = new MailFactory(mailAdress);
        IMail mail = mailFactory.getMail( );
        MyApplication application = new MyApplication(mail);
        application.start( );
    }
}
```

Was geschieht, wenn zur Compilezeit noch nicht bekannt ist, wer die Objekte erzeugt?

Paketstruktur

Mail



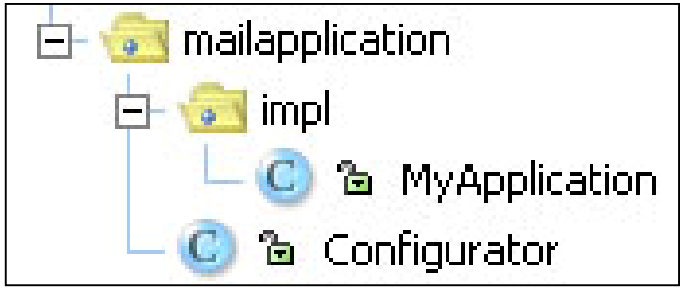
Außensicht der Komponente:

Oberste Ebene (`mail`)

Innensicht der Komponente:

Zweite Ebene (`impl`)

Anwendung



Unterscheide:
Was braucht der Importeur (Benutzer) und was verwirrt ihn nur?

Komponenten und Pakete - Außensicht

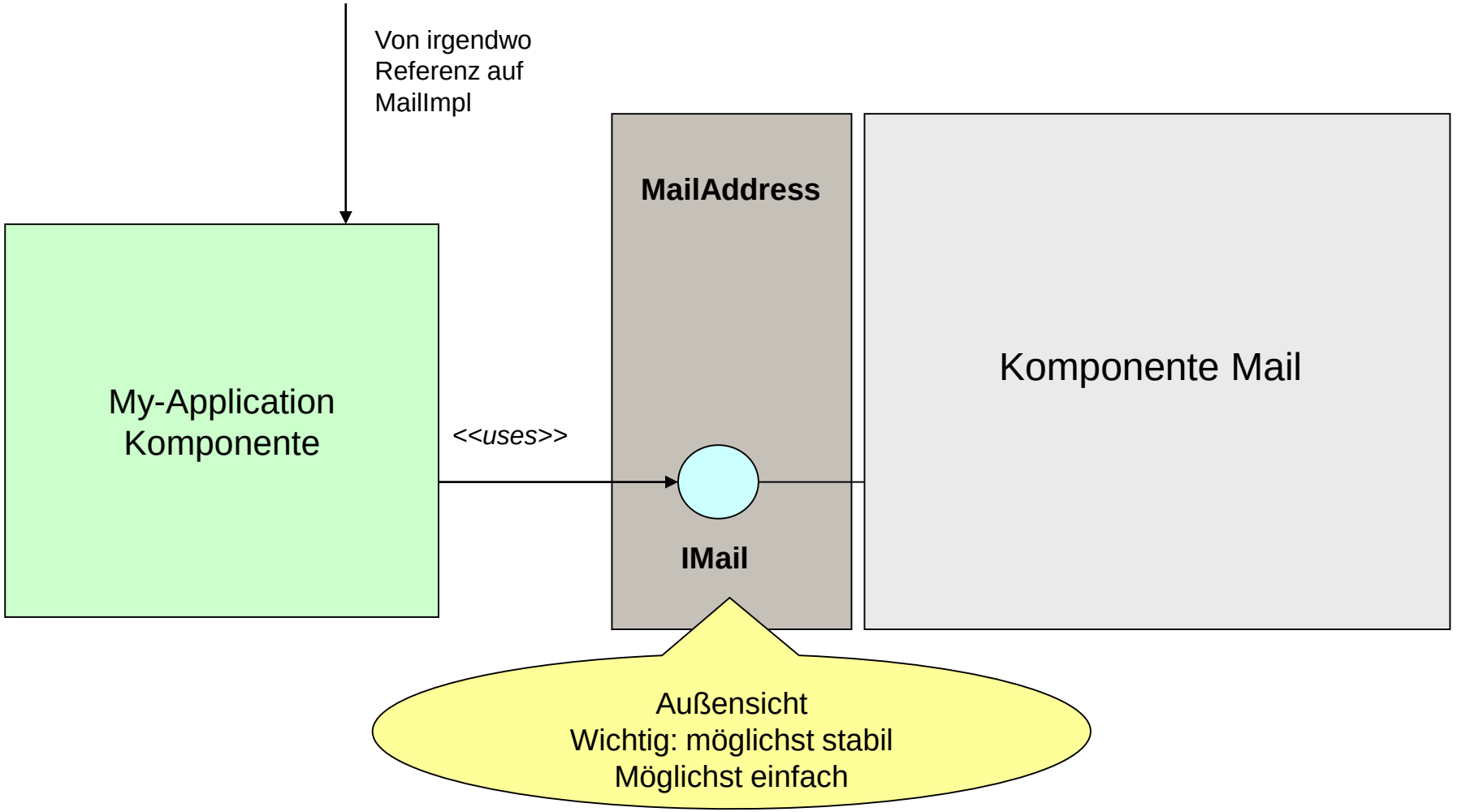
- Mail ist eine Komponente: Ein zusammengehöriges Ganzes.
- Sie hat eine **Außensicht**. Das ist die oberste Ebene der Paketstruktur. Außensicht = Armaturenbrett.
- Dort stehen alle Schnittstellen, die der Importeur braucht, sowie eine Zugangsklasse (Factory).
- Oft braucht man auf der obersten Ebene noch Datentypklassen (hier MailAddress): Die braucht man zur Benutzung der Schnittstellen und somit Teil der Schnittstelle.
- Die Konfiguration liefert die Implementierung zu den benötigten Schnittstellen (in diesem Fall IMail)
- Die Anwendung (hier: MyApplication) sehen nur die Schnittstellen, die sie wirklich brauchen (und keine Konfiguration oder Factories)

Komponenten und Pakete - Innensicht

- Die Mail-Komponente hat eine **Innensicht**. Die befindet sich in den unteren Ebenen der Paketstruktur
- Dort stehen alle diejenigen Implementierungsklassen, die der Importeur nicht sieht.
- Die Innensicht einer Komponente kann unbemerkt ausgetauscht werden.

Komponente Mail aus verschiedenen Perspektiven (1)

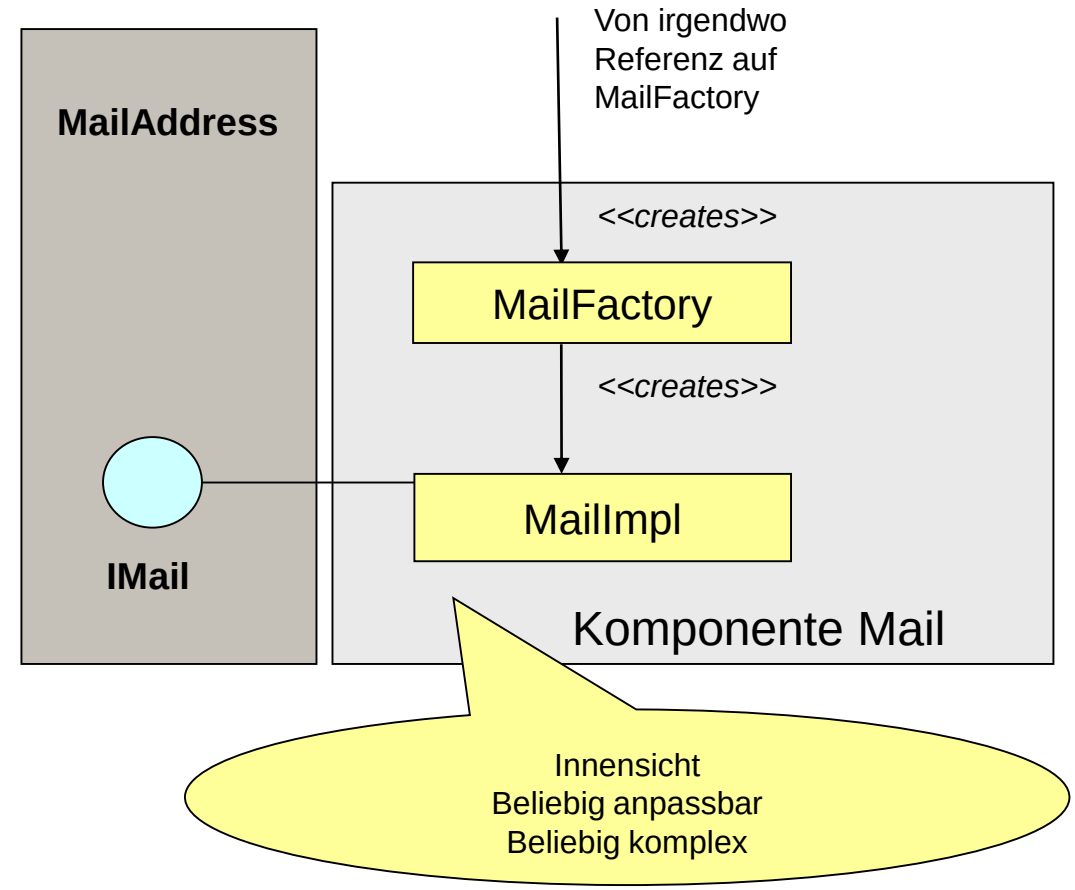
Mailing Komp. aus der Sicht des Nutzers (Team, das MyApplication baut)



Komponente Mail aus verschiedenen Perspektiven (2)

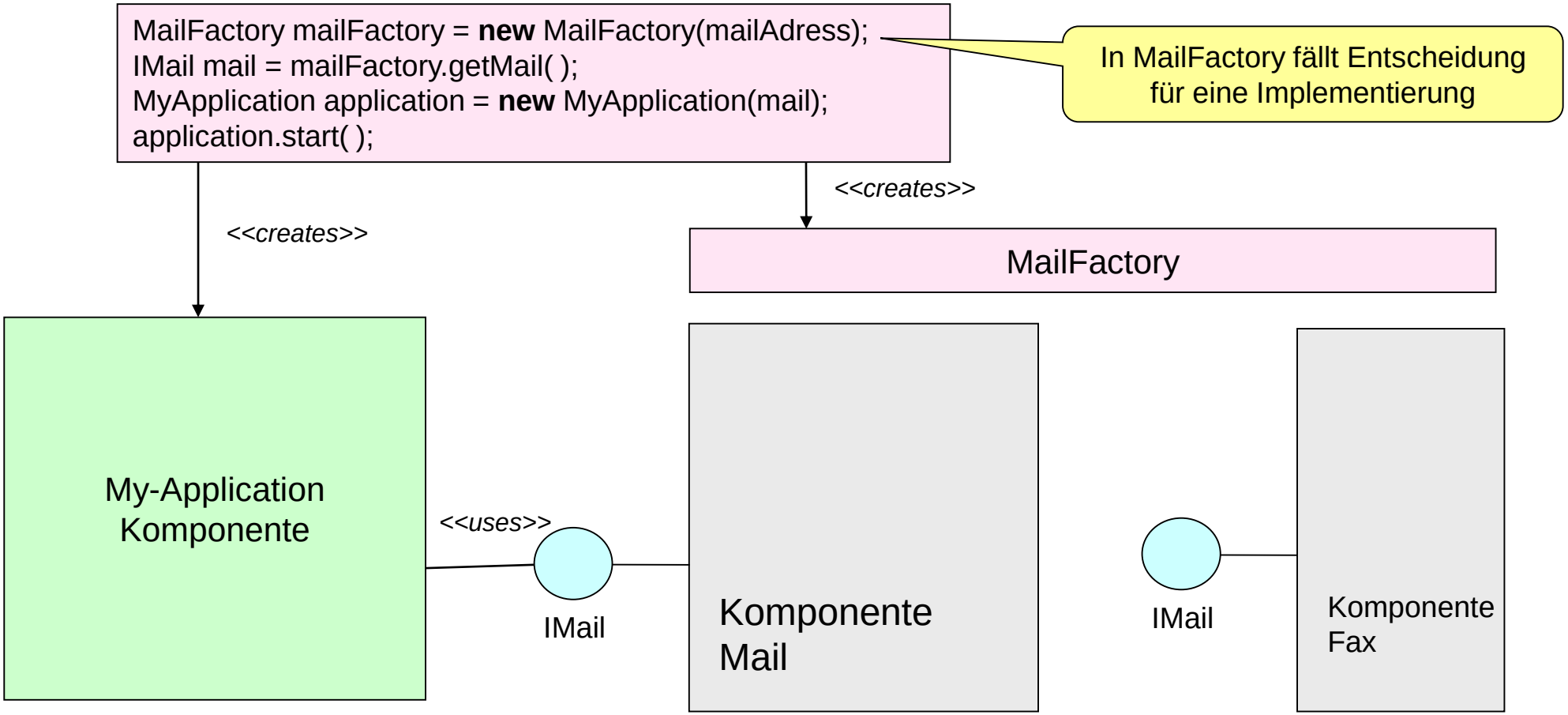
Mailing Komp. aus der Sicht des Erstellers / Experten

(Team, das Mailing-Komponente baut)



Komponente Mail aus verschiedenen Perspektiven (3)

Mailing Komp. aus der Sicht des Zusammenbauers



Schnittstellen (Interfaces)

- Was legt eine Schnittstelle fest?
  - Die Syntax: Methoden, Parameter, Rückgabewerte.
  - Mögliche Fehler. Keine Ausnahmen.
  - Die Semantik ihrer Methoden:
    - Vorbedingungen, Invarianten, Nachbedingungen.
    - Unter welchen Voraussetzungen darf man eine Methode aufrufen?
    - Was bewirkt die Methode (Seiteneffekte!)?
    - Was bedeutet das Ergebnis einer Methode?
  - zur Schnittstelle gehören die verwendeten Typen und ggfs. abstrakte Hilfsklassen

- Vollständigkeit einer Schnittstelle
  - Kann ich per Abfrage nachsehen, was eine Operation angerichtet hat?
  - Kann ich alle Vorbedingungen per Abfrage prüfen?
  - Kann ich eine gegebene Operation rückgängig machen (bewusste Entscheidung!)

Komponenten, Schnittstellen und Konfiguration

- Schnittstelle definiert Operationen und beschreiben das beobachtbare Verhalten einer Komponente
- Schnittstellen sind alleine nicht lauffähig, aber lebensfähig
- Jede Komponente exportiert (implementiert) eine oder mehrere Schnittstellen
- Jede Schnittstelle kann durch beliebig viele Komponenten implementiert werden
- Jede Komponenten importiert beliebig viele Schnittstellen (NICHT Komponenten)
- Konfiguration versieht eine Komponente mit Implementierungen der importierten Schnittstellen

Wie beschreiben wir Komponenten?

- **1. Idee: Worum geht´s?**
  - Management-Sicht: Komponente kostet Geld, muss Nutzen bringen
  - Ohne Idee keine Komponente
- **2. Außensicht: Fachliches Modell, Benutzerhandbuch, Konfiguration**
  - normaler Benutzer
  - Anwendungsprogrammierer (sieht die Schnittstelle)
  - Administrator
  - Arbeitsvorbereitung
  - *Semantik der Operationen*
- **3. Innensicht: Technisches Modell (UML, DB-Entwurf)**
  - Entwickler
  - Administrationsprogrammierer
  - Wartungsprogrammierer
  - *Abhängigkeiten: Unter welchen Annahmen läuft die Komponente? Wen ruft sie auf?*
- **4. Variabilitätsanalyse**
  - Welche Änderungen/Erweiterungen sind absehbar/wahrscheinlich/ausgeschlossen?
  - Was ist jeweils zu tun?

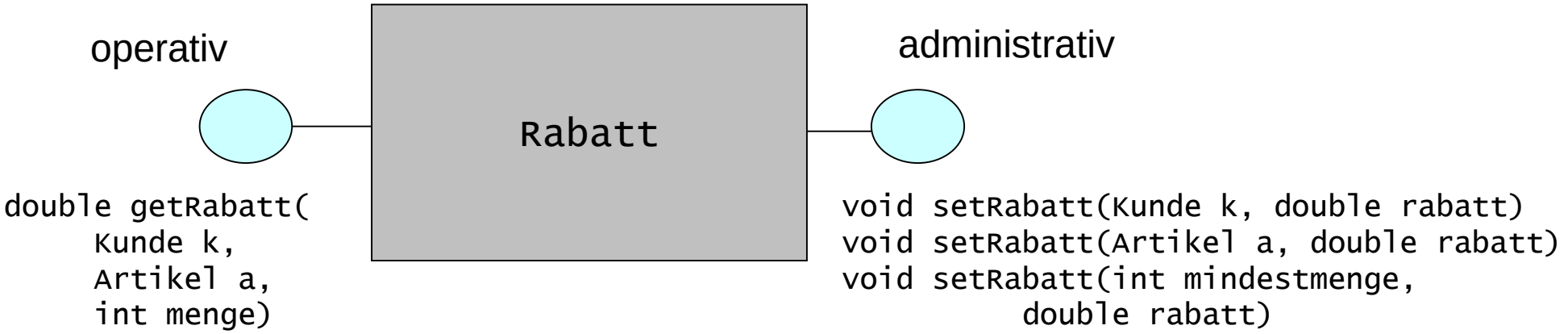
Weitere Beispiele für Komponenten

- **Versichertenauskunft**
  - Seit wann ist das Mitglied versichert?
  - Lebt das Mitglied noch (Abgangsart 99)?
  - Wo wohnt das Mitglied?
  - Weitere Infos ...
- **Druckmanager**
  - Individualdruck
  - Textbausteine
  - Massendruck
  - Portooptimierung
- **Regelmanager**
  - Prädikat = Aussage über ein fachliches Objekt
  - Regel = Verknüpfung von Prädikaten/Regeln (UND, ODER, Negation)
  - Nutzschnittstelle: Trifft die Regel zu?
  - Admin-Schnittstelle: Pflege der Regeln
  - ...

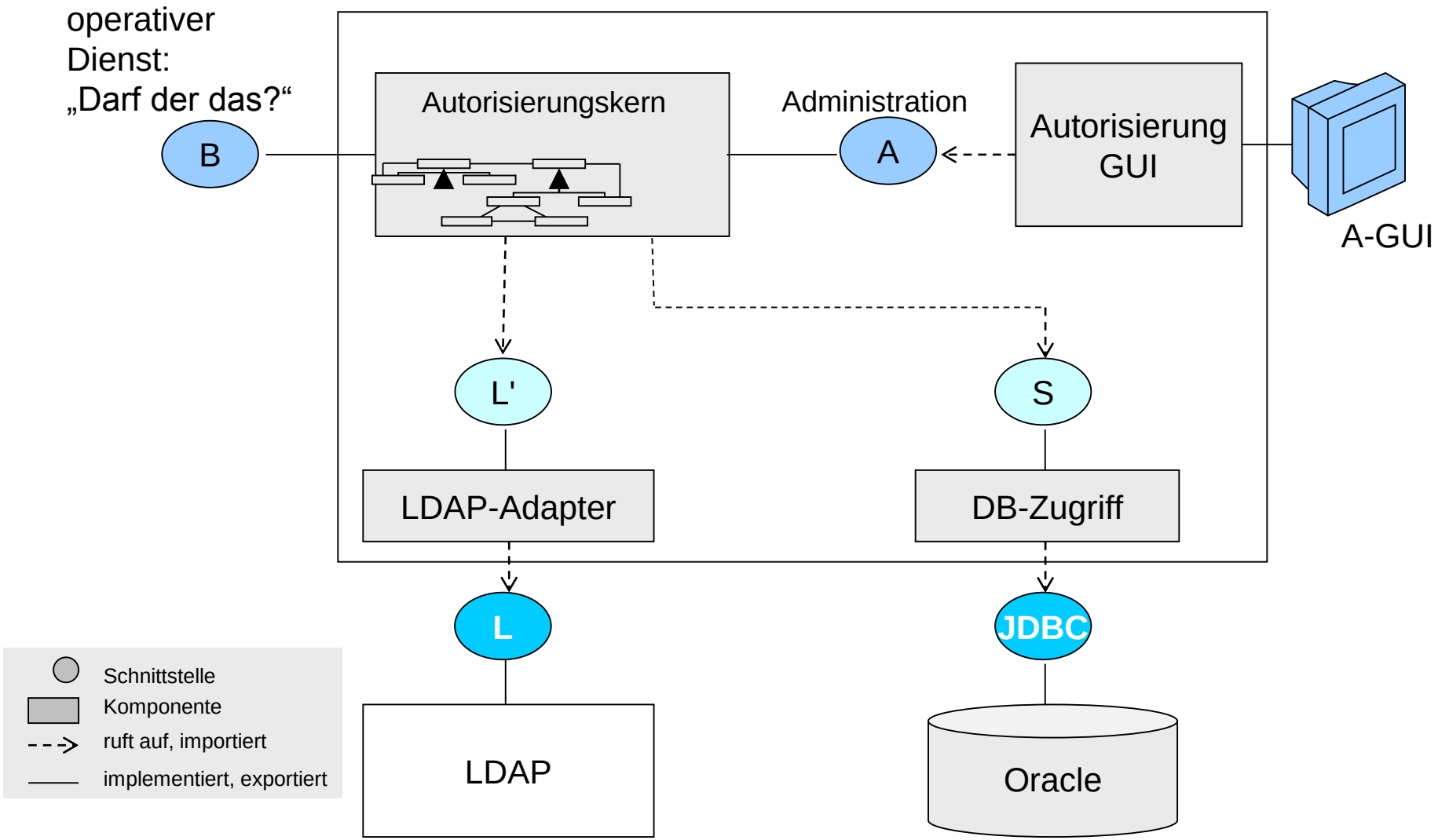
- **Komponenten sind implementiert**
  - prozedural(C, Cobol, PL/SQL)
  - objektorientiert (mehrere Klassen)
- **Komponenten sind vielfältig:**
  - mit/ohne GUI
  - mit/ohne DB
  - klein/groß

Beispiel: Das Rabattproblem.

- Rabatte werden gewährt abhängig von Kunde, Artikel und Menge.
- Problem: Welchen Rabatt bekommt der Kunde bei der Bestellung eines bestimmten Artikels in einer bestimmten Menge tatsächlich?
- Lösung:
  - Algorithmus steckt in Kasten
  - Einfache Schnittstellen
  - Datenmodell für das Rabattproblem erübrigt sich



Beispiel: Berechtigungskomponente



Rollen: Wer weiß was?

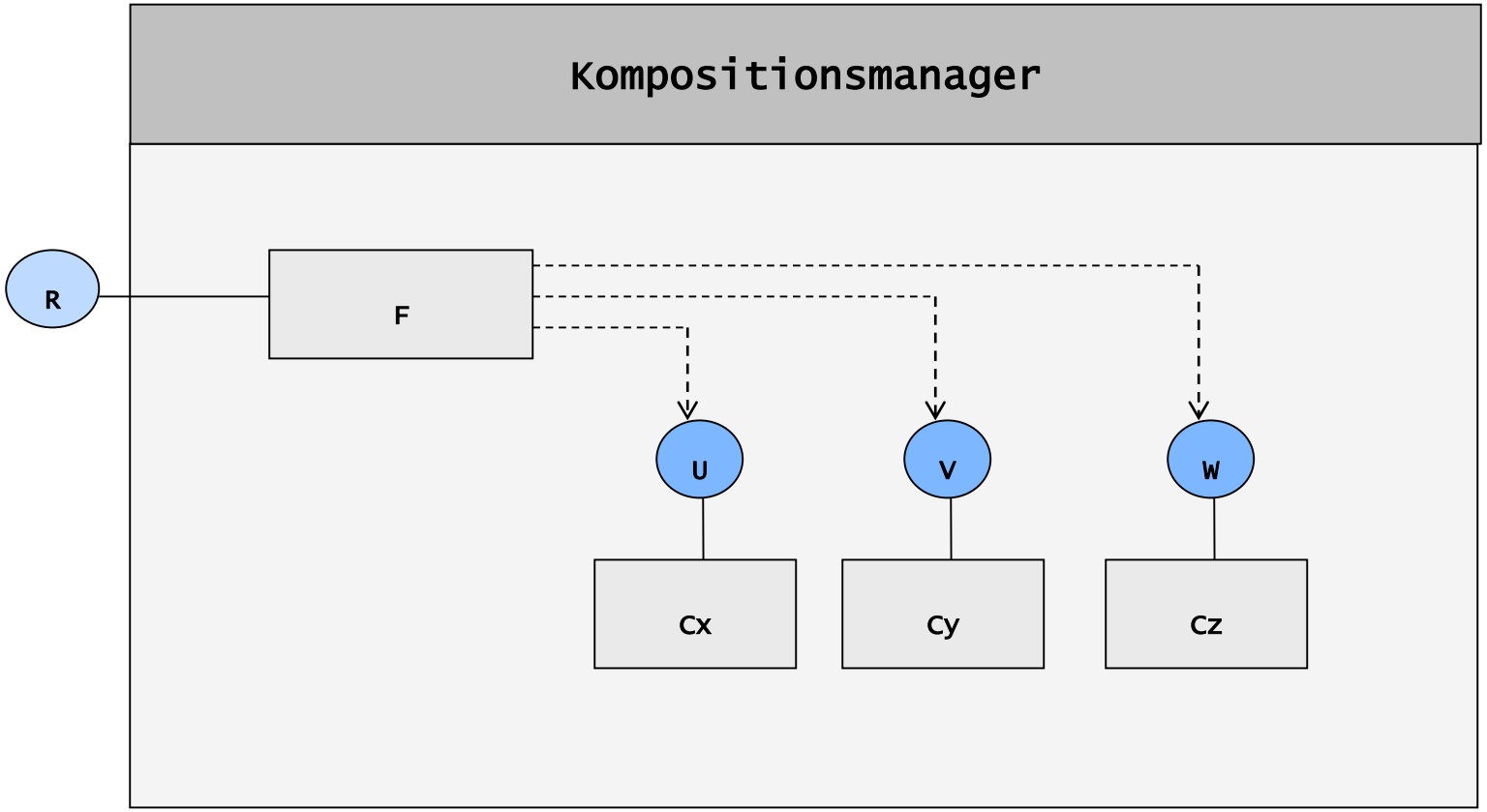
- Anwendungsprogrammierer (viele): **B**
- Administratoren (wenige, abhängig von der Organisation): **A-GUI**
- Autorisierungsexperten (2): **A, B, L', S**
- LDAP-Experte (1): **L, L'**
- DB-Experte (1): **S, JDBC**
- GUI-Programmierer (1): **A, A-GUI**

Daumenregel:

Für jede Schnittstelle S: komplexität(S) * anzahlNutzer(S) = konst.

Beispiel für die Komposition von Komponenten

Die **Fassade** reduziert die komplexen Schnittstellen U, V, W auf eine einfache Schnittstelle R.



Zusammenfassung Komponenten

- Mail ist eine Komponente - ein zusammengehöriges Ganzes.
- Sie hat eine **Außensicht**. Dort stehen alle Schnittstellen und Datentypen, die der Importeur / Nutzer braucht
- Eine Zugangsklasse liefert den Zugang zu den benötigten Schnittstellen (in diesem Fall MailFactory).
- Die Anwendungen (MyApplication) sehen ***nur*** die Schnittstellen, die sie wirklich brauchen (und keine Zugangsklasse). Damit hängt die Anwendung auch ***nur*** von diesen Schnittstellen ab!
- Eine Komponente hat drei Perspektiven: Sicht des Nutzers (Anwendung), Sicht des Erstellers (Implementierung), Sicht des Zusammenbauers (Konfiguration)
- Die Trennung von Admin- und Nutzungsschnittstelle von Komponenten ist oft sinnvoll.
- Beim Entwurf von Komponenten kommt erst die Außensicht, dann die Innensicht. Was mute ich meinem Aufrufer zu?