HSLU Hochschule Luzern

Verteilte Systeme und Komponenten

Komponentenbegriff und Schnittstellendesign

Martin Bättig

Letzte Aktualisierung: 29. September 2022

FH Zentralschweiz

Inhalt

- Komponenten
- Nutzen von Komponenten
- Entwurf mittels Komponenten
- Exkurs: Rolle von Komponenten in Softwarearchitekturen
- Konzept der Schnittstelle
- Dienstleistungsperspektive
- Spezifikation von Schnittstellen

Lernziele

- Sie kennen das Konzept der Software-Komponenten.
- Sie können Komponenten gemäss Spezifikation erstellen, dokumentieren, testen und überarbeiten.
- Sie kennen die Kriterien für gute Schnittstellen im Software-Entwurf und können solche Schnittstellen entwerfen.
- Sie können verschiedene Arten von Schnittstellen angemessen dokumentieren.

Komponenten

Definition des Komponentenbegriffs

(es gibt mehrere Definitionen, nachfolgend zwei gebräuchliche)

- "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties." *(European Conference on Object-Oriented Programming (ECOOP), 1996*

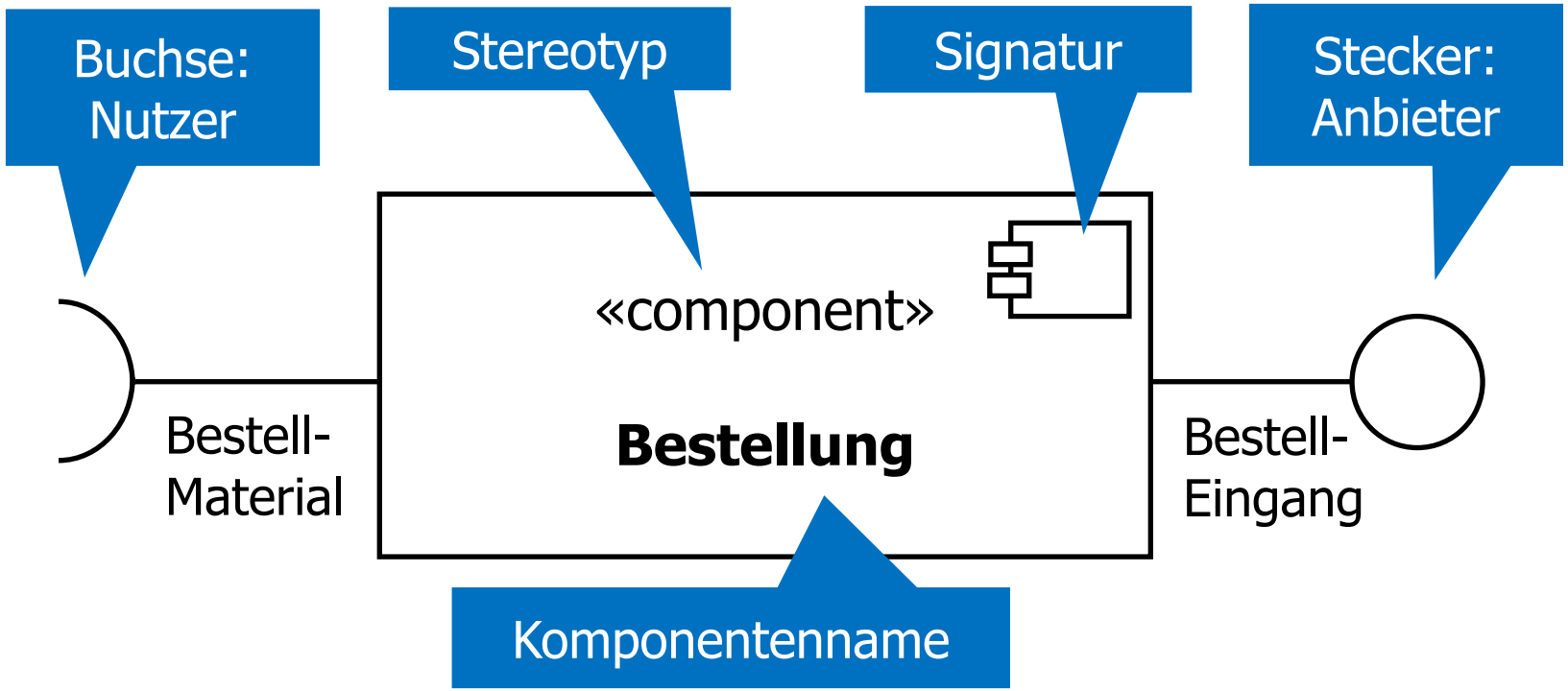
- "Eine Software-Komponente ist ein Software-Element, das zu einem bestimmten Komponentenmodell passt und entsprechend einem Composition-Standard ohne Änderungen mit anderen Komponenten verknüpft und ausgeführt werden kann." *Councill, Heineman: Component-Based Software Engineering, Addison-Wesley, 2001*

Eigenschaften von Komponenten

- Eigenständige, ausführbare Softwareeinheiten (Laufzeit-Sicht) (*).
- Über ihre Schnittstellen in Ihrer Umgebung austauschbar definiert.
- Lassen sich unabhängig voneinander entwickeln.
- Kunden- / anwendungsspezifische, bzw. wiederverwendbare Software sowie Components-off-the-shelf (COTS).
- Können installiert bzw. deployed werden.

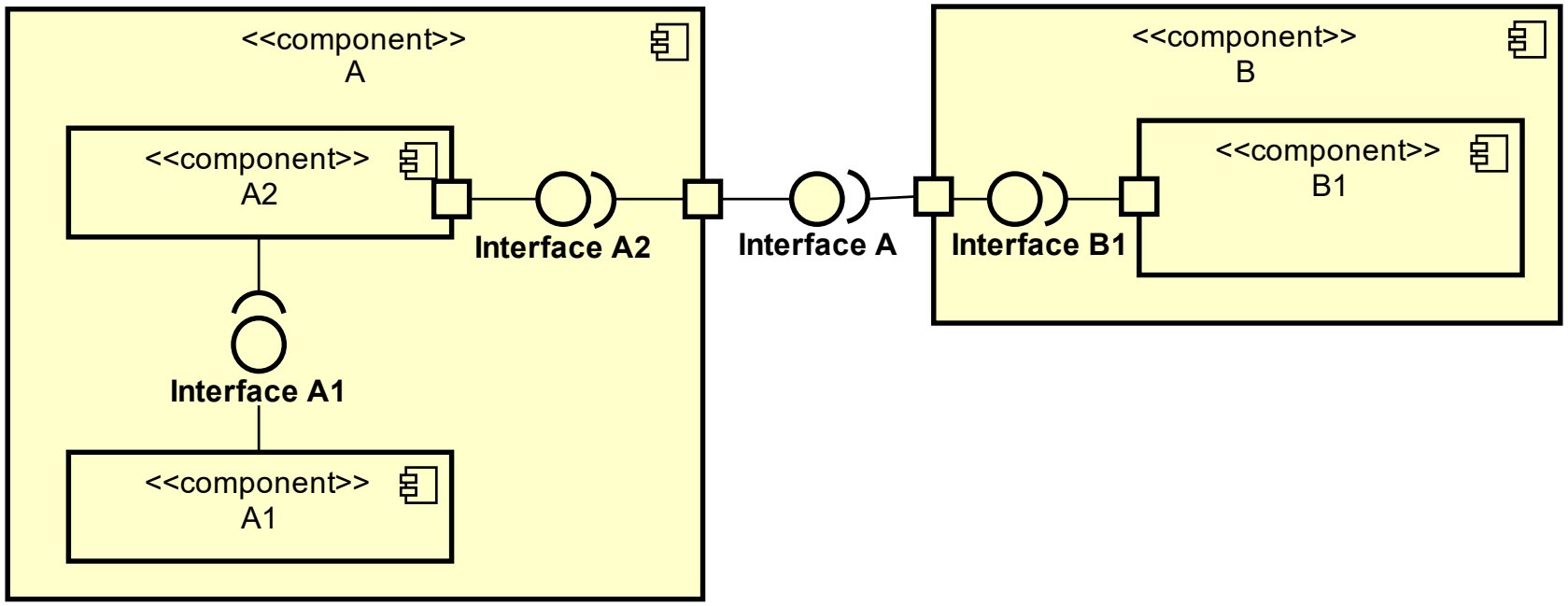
(*) Nicht zwingend in einem eigenen Prozess / Thread.

Modellierung von Software Komponenten in UML



Komponentenbegriff ist hierarchisch

– Modellierung vom "Subsystemen" in UML mittels Verschachtelung:



– UML kennt auch den Stereotyp <<subsystem>>, welcher aus Sicht Modellierung identisch zu <<component>> ist.

– Applikation kann wiederum Komponente eines grösseren Systems sein (ggf. mit anderem Komponentenmodell).

Komponentenmodelle

- Komponentenmodelle sind konkrete Ausprägungen des Paradigmas der Komponentenbasierten Entwicklung.
- Neben der genauen Form und den Eigenschaften einer Komponente muss das Komponentenmodell einen Interaction-Standard und einen Composition-Standard festlegen.
- Komponentenmodelle können ausserdem Implementierungen verschiedener Hersteller besitzen.

Beispiele:

- Enterprise Java Beans (Java).
- (Distributed) Component Object Model (Microsoft, C++).
- CORBA Component Model (Sprachunabhängig).
- OSGi (Java).
- Viele proprietäre Komponentenmodelle (Eigenentwicklungen).

Interaction-Standard

- Beschreibt den Schnittstellenstandard, also wie Komponenten innerhalb eines Komponentenmodells miteinander kommunizieren.
  Beispiele:
  - verteilt oder lokal.
  - verteilte Objekte, Remote-Procedure-Calls, Unix-Pipes (Streams).
  - konkrete Technologien wie SOAP / REST (HTTP).

- Legt fest wie innerhalb einer Komponente die Schnittstelle festgelegt wird.
  Beispiele:
  - Interface Definition Language (CORBA).
  - WSDL (SOAP).

Composition-Standard

– Beschreibt wie der Entwickler Komponenten zusammensetzt um grössere Einheiten zu bilden.
– Beschreibt wie Komponenten ausgeliefert werden.

Beispiele:

- Unix-Prozesse: Zusammenfügen via Pipes, Auslieferung als Binaries.
- Webservices: Zusammenfügen via Domainname / IP-Adresse, Auslieferung als WAR.
- Microservices: Zusammenfügen via Domainname / IP-Adresse, Auslieferung als Service (Package / ZIP / Docker / etc.).

Nutzen von Komponenten

Wiederverwendung

- Verpackung versteckt Komplexität (divide and conquer).
- Reduzierte Auslieferzeit (des eigenen Produkts):
  - Wiederverwenden ist schneller als selbst bauen.
  - Weniger Tests notwendig.
- Grössere Konsistenz: Verwendung von "Standard"-Komponenten.
- Möglichkeit die Beste von verschiedenen Komponenten zu verwenden: Wettbewerb und Markt.

Erbringt vereinbarte Dienstleistung

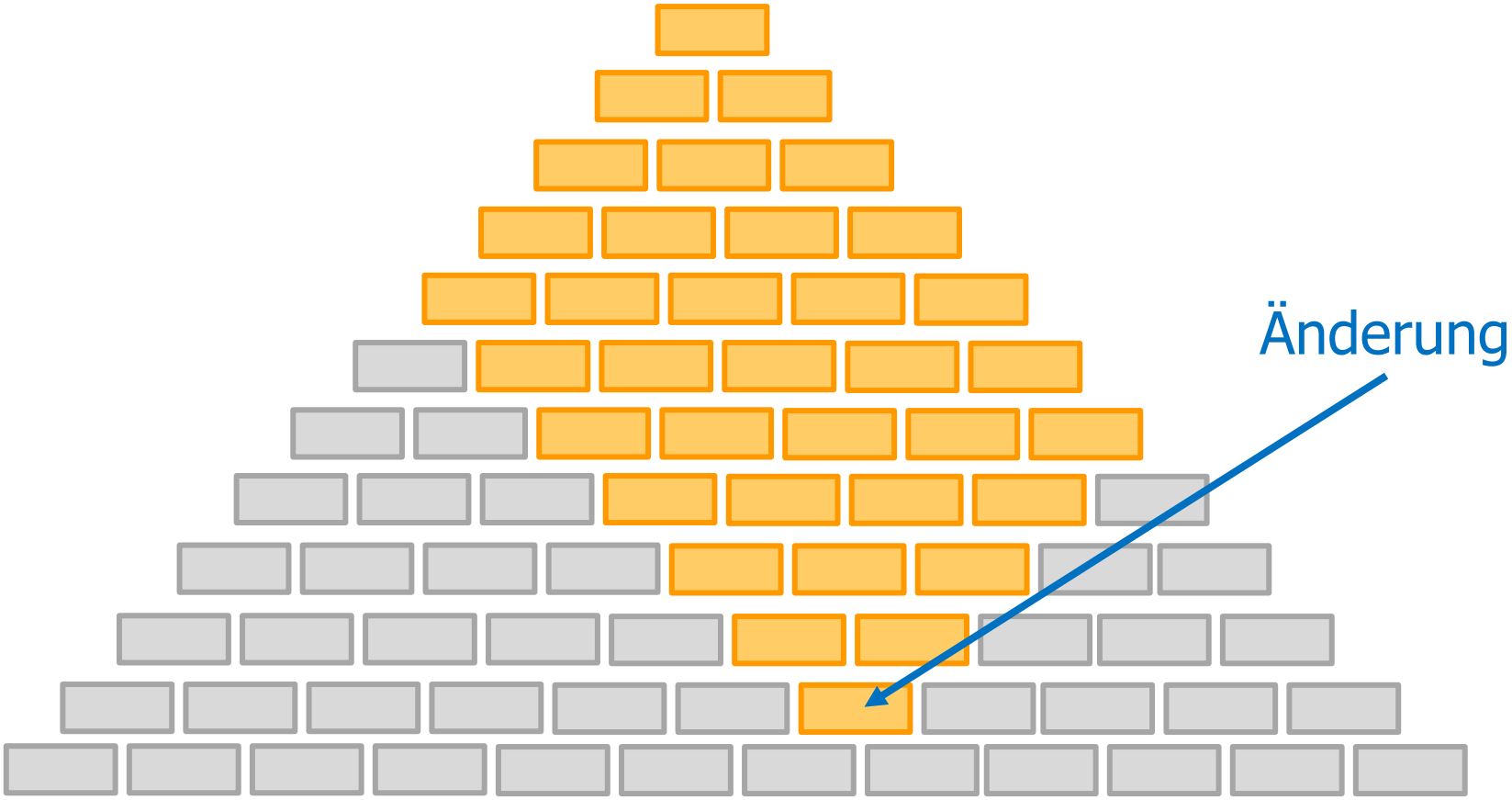
- Erhöhte Produktivität: Existierende Komponenten zusammenfügen.
- Höhere Qualität: Vorgetestet.

Vollständigkeit

- Komponente als ganzes ersetzbar.
- Parallele und verteilte Entwicklung.
  - Präzise Spezifikationen.
  - Verwaltete Abhängigkeiten.
- Verbesserte Wartung.
  - Kapselung limitiert den Auswirkung von Veränderung.

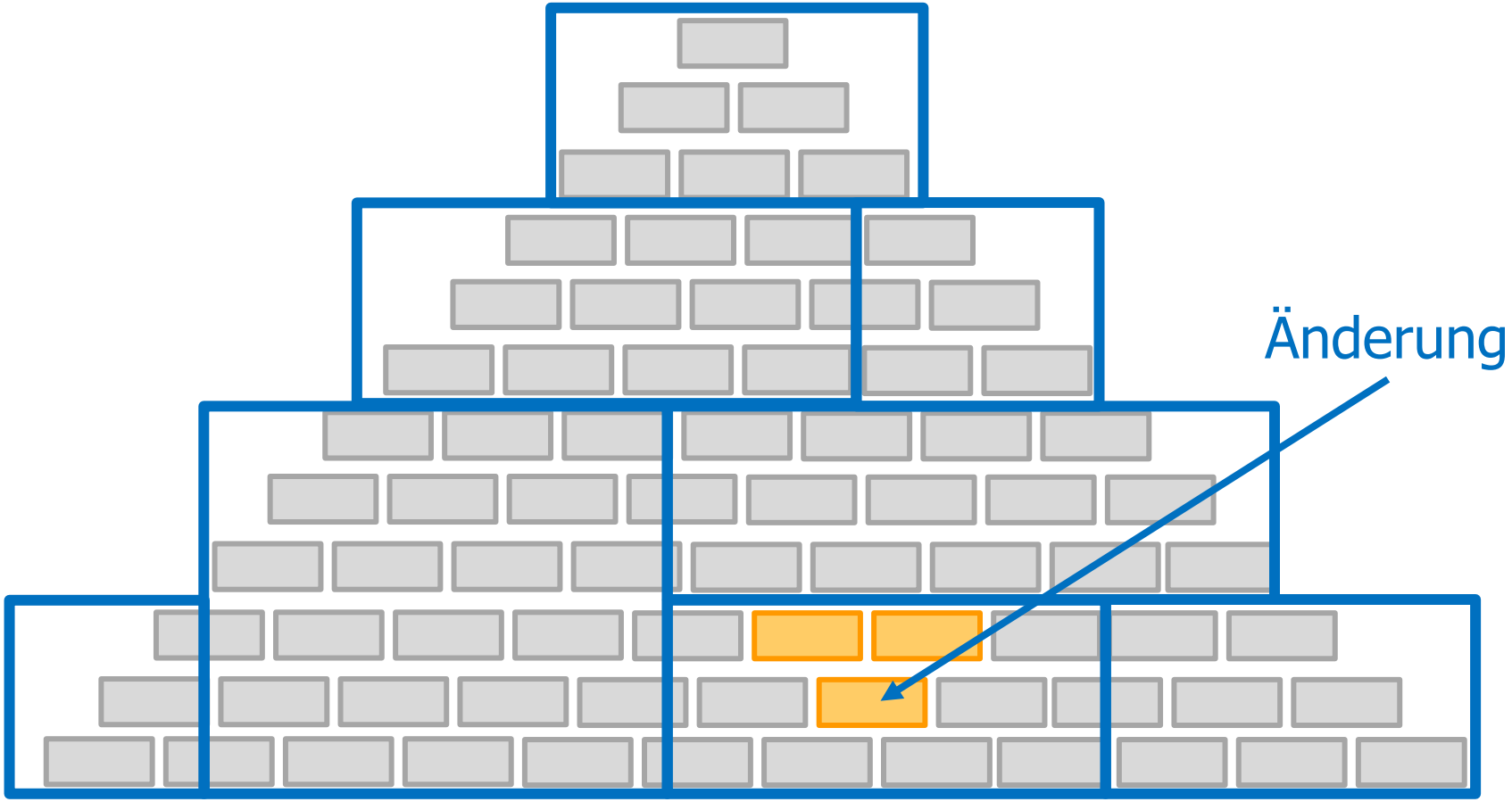
Auswirkung von Änderungen

– Monolithisches System (keine Komponenten):



Auswirkung von Änderungen (forts.)

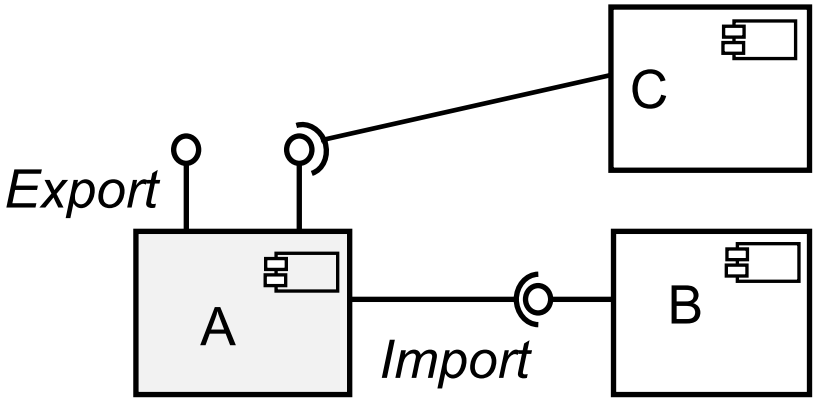
– Komponentenbasiertes System:



Entwurf mittels Komponenten

Spezifikation von Komponenten

- **Export:** unterstützte Interfaces, die andere Komponenten nutzen können.
- **Import:** benötigte / benutzte Interfaces von anderen Komponenten.
- **Verhalten:** Verhalten der Komponente.
- **Kontext:** Rahmenbedingungen im Betrieb der Komponente.



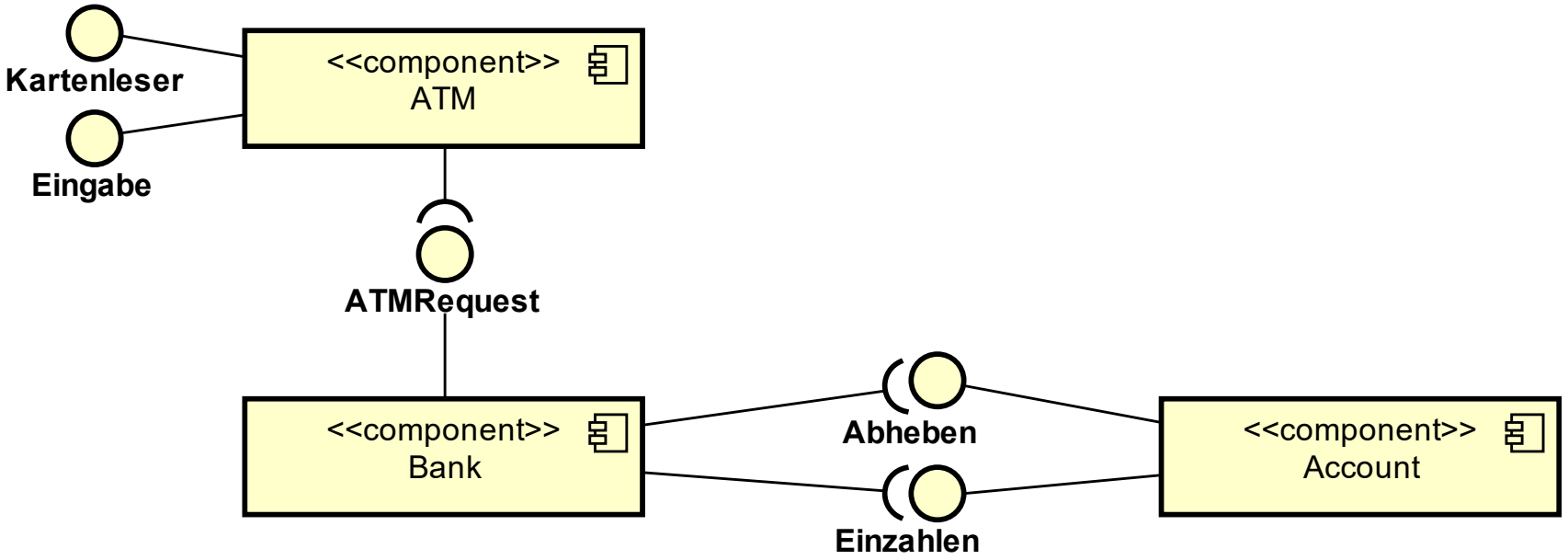
Verhaltenssicht

Weil Komponenten ausführbare SW-Einheiten sind, lässt sich mit ihrer Hilfe das Systemverhalten auf höherer Flughöhe gut darstellen:

- **Components & Connectors:** ausführbare Einheiten und gemeinsame Daten
- **Datenfluss:** Datenfluss zwischen Komponenten.
- **Kontrollfluss:** Wird angestossen von …
- **Prozess:** Welche Komponenten laufen parallel?
- **Verteilung:** Zuordnung der Komponenten zur HW.

UML-Komponentendiagramm: "Verdrahtung"

Beispiel: Bankomat (ATM)



- Sichtbar sind Komponenten und Konnektoren
- Wie sieht es mit Daten- und Kontrollfluss sowie Verteilung aus?

Exkurs: Rolle von Komponenten in Architekturen

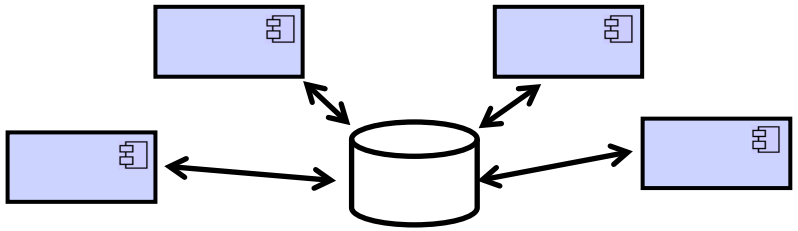
Softwarearchitektur

- Softwarearchitektur enthält Informationen über die Struktur eines Software-Systems:
  - Aus welchen Komponenten besteht ein System?
  - Wie kommunizieren die einzelnen Komponenten?
- Muss relativ früh zu Beginn der Softwareentwicklung stattfinden
  - Spätere Änderung u.U. sehr teuer.
- Architekturmuster für häufig wiederkehrende Architekturen.

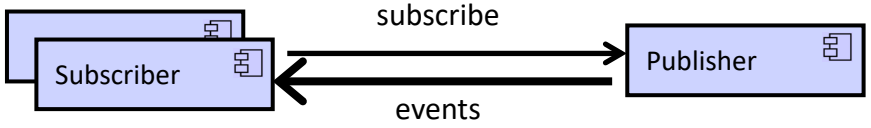
Typische Architekturmuster

– **Hinweis:** Pfeile zeigen den Datenfluss

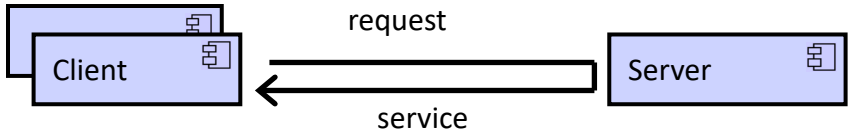
Pipe and Filter



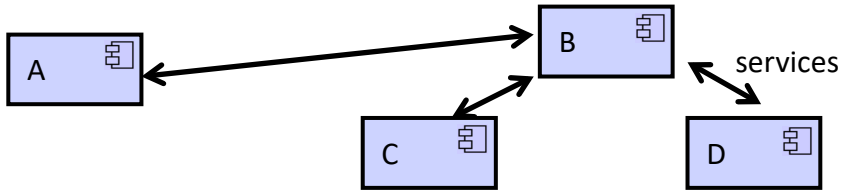
Shared Data



Publish-Subscribe



Client-Server



Peer-to-Peer

Begriff und Konzept der Schnittstelle

Schnittstellen: Begriff und Konzept

- Wo Komponenten kooperieren oder zusammengefügt werden sollen, müssen sie zueinander passen.
- Wir konstruieren Verbindungsstellen, Festlegungen, welche die Kombinierbarkeit sicherstellen.
- Eine Schnittstelle tut nichts und kann nichts.
- Schnittstellen verbinden:
  - Komponenten untereinander: Programmschnittstellen oder kurz Schnittstelle.
  - Komponenten mit dem Benutzer: Benutzerschnittstellen.

Bedeutung des Schnittstellenkonzepts

- **Verständlichkeit:** Schnittstellen machen Software leichter verständlich, denn es genügt, die Schnittstelle (*) zu betrachten.
- **Reduktion von Abhängigkeiten:** Schnittstellen gestatten es, die Abhängigkeiten in der Schnittstelle zu konzentrieren und jede Abhängigkeit von der Implementierung zu vermeiden.
- **Wiederverwendung:** Schnittstellen erleichtern die Wiederverwendung von bewährten Implementierungen und sparen damit Arbeit.

(*) Damit sind nicht nur die Signaturen der Methoden gemeint, sondern auch deren Verhalten und Zusammenspiel.

Bedeutung des Schnittstellenkonzepts (forts.)

- Die Java-Implementierungen der gängigen Behälter (`HashSet`, `TreeSet`, `HashMap`, `TreeMap`, usw.) sind mit Sicherheit leistungsfähiger als alles, was der beste Programmierer in seinem Projekt unter Zeitdruck fertig bringt.
- Achtung: Schnittstellen entfalten ihren Nutzen nur dann, wenn wirklich ohne Bezug auf die Implementierung nur gegen Schnittstellen programmiert wird.

Breite einer Schnittstelle

Bestimmt über:

- Anzahl Operationen (mehr ist breiter).
- Anzahl Funktionsüberschneidungen (mehr ist breiter).
- Anzahl von Parametern (mehr ist breiter).
- Anzahl globaler Daten (mehr ist breiter).
- Typ der Parameter und Rückgabewerte (generisch ist breiter).

Schmälere Schnittstellen haben weniger Abhängigkeiten!

Kriterien für gute Schnittstellen

- Schnittstellen sollen schmal sein.
- Schnittstellen sollen einfach zu verstehen sein.
- Schnittstellen sollen gut dokumentiert sein.

Beispiel: Modem-Schnittstelle

Reduktion von Abhängigkeiten erzielt durch Aufteilung:

Variante 1 (schmal):

```
interface Modem {
    void dial(String number);
    void hangup();
    void send(char data);
    char receive();
}
```

Variante 2 (noch schmaler):

```
interface Connection {
    void dial(String number);
    void hangup();
}

interface Transmit {
    void send(char character);
    char receive();
}
```

Weitere Beispiele

- Breit:
  https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html
- Schmal:
  https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

Schnittstellen: Design by Contract

Dienstleistungsperspektive

Konsument **Anbieter**

stimmt zu arbeitet gemäss

Vertrag

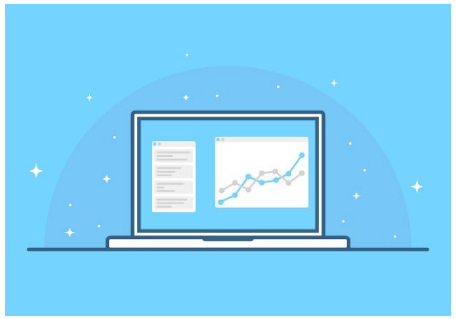
Dienstleistung

Bildquellen: Pixabay

Serviceperspektive

Konsument **Anbieter**

stimmt zu arbeitet gemäss



Schnittstelle

Dienstleistung

Bildquellen: Pixabay

Design by Contract

Das Zusammenspiel der Komponenten wird durch einen „Vertrag“ erreicht, dieser besteht aus:

- Preconditions: Zusicherungen, die der Aufrufer einzuhalten hat.
- Postconditions: Nachbedingungen, die der Aufgerufene garantiert.
- Invarianten: Bedingung, die Instanzen einer Klasse ab der Erzeugung erfüllen müssen (kann während der Ausführung einer Funktion/Methode verletzt sein).

Der Vertrag kann sich auf Variablen, Parameter und Objektzustände beziehen.

Verantwortlichkeiten bei "Design by Contract"

	Nutzer	Anbieter
Precondition	Nutzer muss sicher stellen, dass Vorbedingungen vor Ausführung einer Methode gelten	Prüfen. Aussagekräftige Fehlermeldung, falls inkorrekt.
Postcondition	Während Entwicklung: Doppelcheck mit assert (Defensives Programmieren)	Anbieter muss sicher stellen, dass Nachbedingungen nach Ausführung einer Methode gelten.
Invariante		Anbieter muss sicher stellen, dass Invarianten vor- und nach Ausführung jeder Methode gelten. Während Entwicklung: Doppelcheck mit assert (Defensives Programmieren)

Beispiel mit Pre- and Postconditions

```
public class LinkedList {
    private static class Node {
        int value;
        Node prev, next;
        LinkedList list;
    }

    private Node first, last;
    private int size;
}


// Inserts a new element with content value after node.
// PRE:  Node is element of list (and not null).
// POST: Returns node containing new element,
//       new element is part of list, size of list increased by 1.
public Node insert(Node node, int value) {
    if (node.list != this) throw new IllegalArgumentException(...);
    ...
}
```

Klasseninvarianten

```
// INV: either both null or both not null.
// INV: size >= 0
```

Pre- und Postconditions

Klassenraumübung: Pre- and Postconditions

Gegeben sei folgende Methode:

```
static int sum(int[] arr, int fromIndex, int toIndex) {
    int sum = 0;
    for (int i = fromIndex; i < toIndex; ++i) {
        sum += arr[i];
    }
    return sum;
}
```

Aufgabe:

- Bestimmen Sie Pre- and Postconditions.
- Würde Sie die Eingabeparameter prüfen? Falls ja, wie?

Spezifikation von Schnittstellen

Dokumentation von Schnittstellen

- Zur Programmschnittstelle gehört alles,
  - was für die Benutzung der Komponente wichtig ist und
  - was der Programmierer verstehen und beachten muss.

- Jede Programmschnittstelle definiert eine Menge von Methoden mit den folgenden Eigenschaften:
  - **Syntax** (Rückgabewerte, Argumente, in/out, Typen).
  - **Semantik** (Was bewirkt die Methode?).
  - **Protokoll** (z.B. synchron, asynchron).
  - **Nichtfunktionale Eigenschaften** (Performance, Robustheit, Verfügbarkeit, bei Web-Anwendungen möglicherweise auch Kosten).

Dokumentation von Schnittstellen (forts.)

- Die Syntax einer Programmschnittstelle notieren wir in der verwendeten Programmiersprache.
- Die Semantik lässt sich weniger leicht darstellen. Obwohl Schnittstellen so wichtig sind, gibt es bis heute keine allgemein akzeptierte Art der Dokumentation für deren Semantik.
- Beispiel für die Dokumentation einer Schnittstelle ist die Spezifikation des StringPersistor:

  ILIAS: I.BA_VSK_MM.H2201 » Projekt » VSK_stringpersistor-api-6.0.2

Angaben aus der Sicht des Schnittstellenanbieters

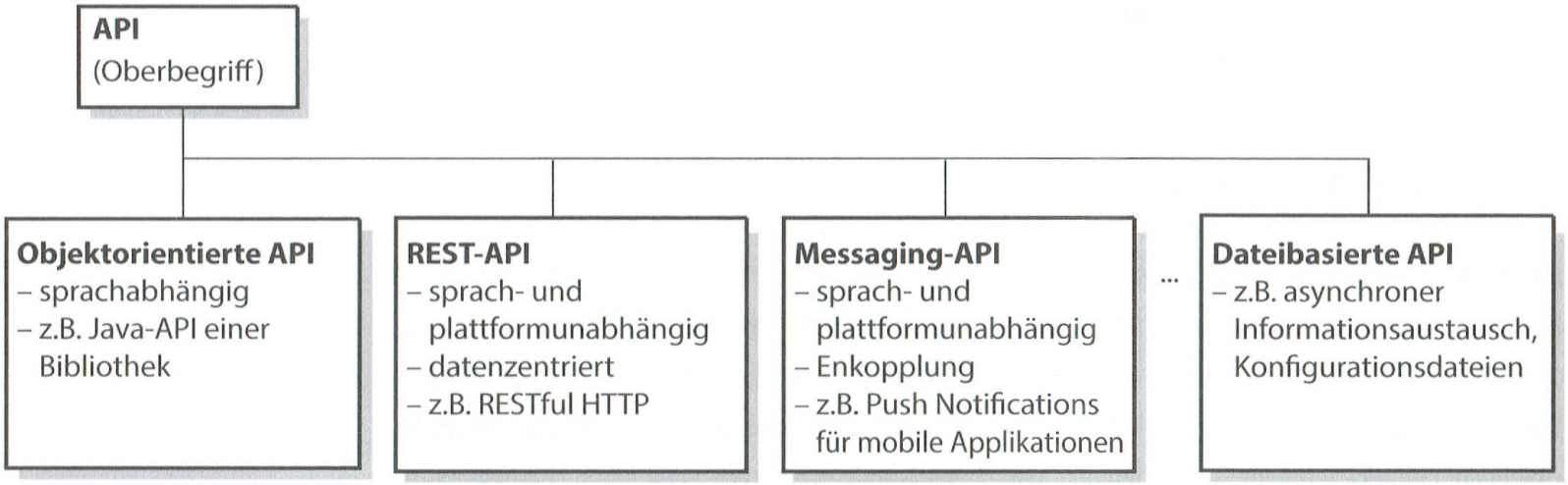
- Name der Schnittstelle.
- Eingabeparameter: Welche Informationen werden an die Komponente weitergeleitet.
- Ausgabeparameter: Welche Informationen die Komponenten zurückliefert.
- Zustandsänderung der Komponenten.
- Spezifikation inwiefern sich Eingabeparameter auf die Zustandsänderung und die Ausgabeparameter auswirken.

Schnittstellen in Java

- In Java werden Schnittstellen mit Java-Interfaces deklariert.
- Schnittstellen werden (wie Klassen) zu class-Dateien kompiliert.
- Schnittstellen können somit (wie Klassen) in JAR-Dateien verpackt und verteilt werden.
- Sinnvollerweise dokumentiert man Schnittstellen besonders gut mit JavaDoc (woraus eine Dokumentation im HTML-Format erzeugt werden kann).
- Schnittstellen an der Systemgrenze und zwischen Subsystemen sind architekturrelevant und werden in der Architekturbeschreibung dokumentiert.
- öffentliche Schnittstellen bezeichnet man häufig als API: Application Programmer Interface (können auch mehrere Java-Interfaces sein).

Application Programmer Interface (API)

- Eine API spezifiziert die Operationen sowie die Ein- und Ausgaben einer Softwarekomponente.
- Hauptzweck besteht darin, eine Menge an Funktionen unabhängig von ihrer Implementierung zu definieren.
- Die Implementierung kann variieren ohne die Benutzer der Softwarekomponente zu beeinträchtigen.



Zusammenfassung

- Komponente: Softwareelement passend zu einem bestimmten Komponentenmodell, eigenständig ausführbar und über Schnittstellen austauschbar definiert.
- Nutzen erzielt durch einfache Wiederverwendung, Erbringung der vereinbarten Dienstleistung sowie vollständiger Ersetzbarkeit.
- Komponenten limitieren Auswirkung von Änderungen auf das Gesamtsystem.
- Entwurf mittels Komponenten durch Betrachtung von importierten und exportierten Schnittstellen, sowie durch die Beschreibung des Verhaltens und des Ausführungskontextes.
- Darstellung in UML mittels Komponentendiagramm.

Zusammenfassung (forts.)

- Begriff und Konzept der Schnittstelle:
  - Schnittstellen machen Software leichter verständlich.
  - Schnittstellen helfen Abhängigkeiten zu reduzieren.
  - Schnittstellen erleichtern die Wiederverwendung.
  - Kriterien: schmal / einfach zu verstehen / gut dokumentiert.
- Dienstleistungsperspektive / Design by Contract.
  - Service Provider / Service Consumer.
  - Preconditions / Postconditions / Invarianten.
- Spezifikation von Schnittstellen
  - Dokumentation: Syntax / Semantik / Protokoll / nicht-funktionale Eigenschaften.
  - UML-Notationsmöglichkeiten.
  - API dokumentiert öffentliche Schnittstellen.

Literatur und Quellen

- Die UML-Kurzreferenz für die Praxis von B. Oestereich, Oldenbourg Wissenschaftsverlag, 2014.
- Moderne Software-Architektur von Johannes Siedersleben, Dpunkt Verlag, 2004.
- Component Software: Beyond Object-Oriented Programming von Clemens Szyperski, Addison-Wesley Professional, 2002.
- Effektive Software-Architekturen von Gernot Starke, Hanser Fachbuch, 2002.
- Software Engineering von Jochen Lodewig & Horst Richter, Dpunkt Verlag, 2013.

Fragen?