

Verteilte Systeme und Komponenten

Zusammenfassung Frühlingssemester 2018

Patrick Bucher

18.05.2018

Inhaltsverzeichnis

1 Komponenten	1
1.1 Begriffe und Architekturen	1
1.1.1 Der Komponentenbegrif	1
1.1.2 Der Nutzen von Komponenten	2
1.1.3 Der Entwurf mit Komponenten	3
1.1.4 Komponenten in Java	4

1 Komponenten

Herkunft: *componere* (lat.) = zusammensetzen

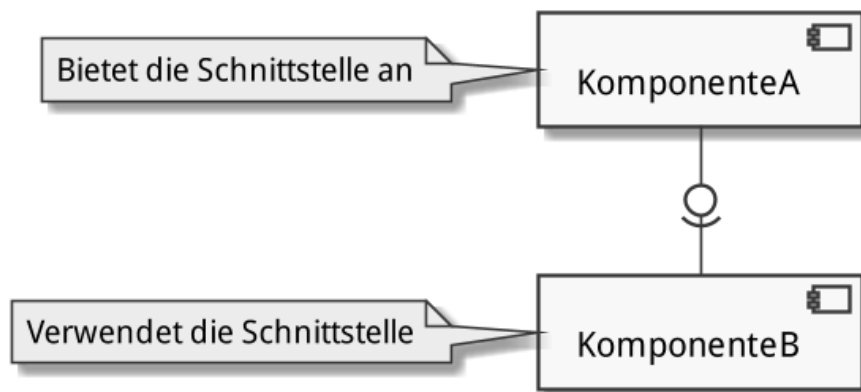


Abbildung 1: Komponentendiagramm (UML2)

1.1 Begriffe und Architekturen

1.1.1 Der Komponentenbegrif

- Definition: Eine Software-Komponente
 1. ist ein Software-Element
 2. passt zu einem bestimmten Komponentenmodell
 3. folgt einem bestimmten *Composition Standard*
 4. kann ohne Änderungen mit anderen Komponenten verknüpft und ausgeführt werden
- Eigenschaften: Software-Komponenten
 1. sind eigenständig ausführbare Softwareeinheiten
 2. sind über ihre Schnittstellen austauschbar definiert
 3. lassen sich unabhängig voneinander entwickeln
 4. können kunden- und anwendungsspezifisch oder anwendungsneutral und wiederverwendbar sein
 - COTS (Commercial off-the-shelf): Software «von der Stange»
 5. können installiert und deployed werden
 6. können hierarchisch verschachtelt sein
- Komponentenmodelle
 - sind konkrete Ausprägungen des Paradigmas der komponentenbasierten Entwicklung
 - definieren die genaue Form und Eigenschaften einer Komponente
 - definieren einen *Interaction Standard*
 - * wie können die Komponenten miteinander über Schnittstellen kommunizieren (Schnittstellenstandard)
 - * wie werden die Abhängigkeiten der Komponenten voneinander festgelegt
 - von der Komponente verlangte Abhängigkeiten: *Required Interfaces*
 - von der Komponente angebotene Abhängigkeiten: *Provided Interfaces*
 - definieren einen *Composition Standard*
 - * wie werden die Komponenten zu grösseren Einheiten zusammengefügt
 - * wie werden die Komponenten ausgeliefert (Deployment)
- Beispiele verbreiteter Komponentenmodelle:
 - Microsoft .NET
 - EJB (Enterprise Java Beans)
 - OSGi (Open Services Gateway Initiative)
 - CORBA (Common Object Request Broker Architecture)
 - DCOM (Distributed Component Object Model)

1.1.2 Der Nutzen von Komponenten

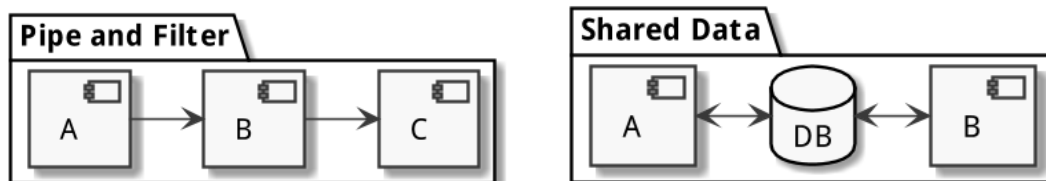
- Packaging: *Reuse Benefits*
 - Komplexität durch Aufteilung reduzieren (*Divide and Conquer*)
 - Wiederverwendung statt Eigenentwicklung spart Entwicklungszeit und Testaufwand
 - erhöhte Konsistenz durch Verwendung von Standardkomponenten

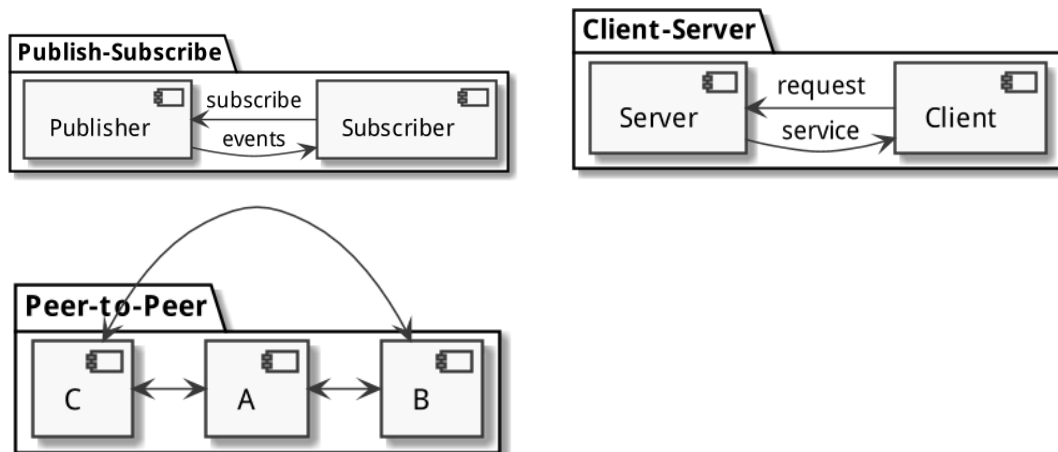
- Möglichkeit zur Verwendung bestmöglicher Komponente auf dem Markt
- Service: *Interface Benefits*
 - erhöhte Produktivität durch Zusammenfügen bestehender Komponenten
 - erhöhte Qualität aufgrund präziser Spezifikationen und vorgetesteter Software
- Integrity: *Replacement Benefits*
 - erweiterbare Spezifikation durch inkrementelle Entwicklung und inkrementelles Testing
 - parallele und verteilte Entwicklung durch präzise Spezifizierung und Abhängigkeitsverwaltung
 - Kapselung begrenzt Auswirkungen von Änderungen und verbessert so wie Wartbarkeit

1.1.3 Der Entwurf mit Komponenten

- Komponentenbasierte Entwicklung
 - steigende Komplexität von Systemen, Protokollen und Anwendungsszenarien
 - Eigenentwicklung wegen Wirtschaftlichkeit und Sicherheit nicht ratsam
 - Konstruktion von Software aus bestehenden Komponenten immer wichtiger
 - Anforderungen (aufgrund mehrmaliger Anwendung) an Komponenten höher als an reguläre Software
- Praktische Eigenschaften
 - Einsatz einer Komponente erfordert nur Kenntnisse deren Schnittstelle
 - Komponenten mit gleicher Schnittstelle lassen sich gegeneinander austauschen
 - Komponententests sind Blackbox-Tests
 - Komponenten lassen sich unabhängig voneinander entwickeln
 - Komponenten fördern die Wiederverwendbarkeit
- Komponentenspezifikation
 - Export: angebotene/unterstützte Interfaces, die von anderen Komponenten genutzt werden können
 - Import: benötigte/verwendete Interfaces von anderen Komponenten
 - Kontext: Rahmenbedingungen für den Betrieb der Komponente
 - Verhalten der Komponente

Architekturmuster:





1.1.4 Komponenten in Java

- Komponenten in Java SE
 - Komponenten als normale Klassen implementiert
 - Komponenten können, müssen sich aber nicht an die *Java Beans Specification* halten
 - * Default-Konstruktor
 - * Setter/Getter
 - * Serialisierbarkeit
 - * PropertyChange
 - * Vetoable
 - * Introspection
 - Weitergehende Komponentenmodelle in Java EE
 - * Servlets
 - * Enterprise Java Beans
- Austauschbarkeit
 - Die Austauschbarkeit von Komponenten wird durch den Einsatz von Schnittstellen erleichtert.
 - Schnittstellen werden als Java-Interface definiert und dokumentiert (JavaDoc).
 - Eine Komponente implementieren eine Schnittstelle als Klasse.
 - * mehrere, alternative Implementierungen möglich
 - * Austauschbarkeit über Schnittstellenreferenz möglich
 - Beispiel: API von JDBC (Java Database Connectivity)
 - * von Sun/Oracle als API definiert
 - * von vielen Herstellern implementiert (JDBC-Treiber für spezifische Datenbanksysteme)
 - * Datenbanktausch auf Basis von JDBC möglich
- Deployment
 - über .jar-Dateien (Java Archive): gezippte Verzeichnisstrukturen bestehend aus

- * kompilierten Klassen und Interfaces als `.class`-Dateien
- * Metadaten in `META-INF/manifest.mf`
- * optional weitere Ressourcen (z.B. Grafiken, Textdateien)
- Deployment von Schnittstelle und Implementierung zum einfacheren Austausch häufig in getrennten `.jar`-Dateien mit Versionierung, Beispiel (fiktiv):
 - * `jdbc-api-4.2.1.jar` enthält die Schnittstelle
 - * `jdbc-mysql-3.2.1.jar` enthält die MySQL-Implementierung
 - * `jdbc-postgres-4.5.7.jar` enthält die PostgreSQL-Implementierung
 - * Versionierung idealerweise im Manifest und im Dateinamen (Konsistenz beachten!)