

# **Vorlesung Software aus Komponenten**

## 4. Moderne Komponentenkonzepte

Prof. Dr. Hans-Gert Gräbe  
Wintersemester 2012/13

### Das Spring Framework

- Leichtgewichtiges Open Source Applikationsframework für die Java-Plattform, um Entwicklungen innerhalb der J2EE zu vereinfachen
  - Alternative zu den Sun J2EE Blueprints als Architekturempfehlung für J2EE-Anwendungen, dem ein strenges Schichtenkonzept (Web-Schicht, Business-Schicht auf EJB-Basis, Daten-Schicht) zu Grunde liegt.
- Verwirklicht beispielhaft das Zusammenspiel zwischen POJO's auf der Anwendungsebene und verschiedenen konditionierbaren Framework-Kontexten (Entwicklung, Testumgebung, Produktivumgebung) durch weitere Einbettung der Kontexte
- Auf der Basis existieren weitere Projekte, welche die Anbindung von Spring an verschiedene häufig anzutreffende Entwicklungsumgebungen und -anforderungen realisieren.

### Ansatz und Community

- Oktober 2002: Grundlagen werden von Rod Johnson im Rahmen seiner Buchreihe „Expert One-On-One J2EE Design and Development“ entwickelt, der auch den Sourcecode des Framework-Gerüsts als freien Download zur Verfügung stellt.
- Juni 2003: erstes Release, unter der Apache 2.0 Lizenz
- März 2004: Spring 1.0 – seitdem fand das Framework sehr schnelle Verbreitung. Entwicklung wird von SpringSource koordiniert
  - <http://www.springsource.org>
- Gemeinsame Entwicklungsbemühungen um einen leichtgewichtigen Nachfolger der J2EE
  - Bibliotheken stehen unter der Apache 2.0 Lizenz frei zur Verfügung
  - *Spring Community Forums* als themenspezifische Kommunikationsplattformen
  - *SpringSource Enterprise Bundle Repository* als eine Sammlung von Basiskomponenten

### Mission Statement

We believe that:

- J2EE should be easier to use
- It is best to program to interfaces, rather than classes. Spring reduces the complexity cost of using interfaces to zero.
- JavaBeans offer a great way of configuring applications.
- OO design is more important than any implementation technology, such as J2EE.
- Checked exceptions are overused in Java. A platform shouldn't force you to catch exceptions you're unlikely to be able to recover from.
- Testability is essential, and a platform such as Spring should help make your code easier to test.

We aim that:

- Spring should be a pleasure to use
- Your application code should not depend on Spring APIs
- Spring should not compete with good existing solutions, but should foster integration. (For example, JDO, Toplink, and Hibernate are great O/R mapping solutions. We don't need to develop another one.)

(Quelle <http://www.springsource.org/about>)

### Geschichte

- Oktober 2006: Version 2.0
- November 2007: Version 2.5
- September 2009: SpringSource wird von VMWare übernommen
- Dezember 2009: Version 3.0
  - Neue *Spring Expression Language*, mit der Ausdrücke zur Laufzeit ausgewertet werden können. So lassen sich dynamische Elemente des Systems implementieren, ohne komplexe Techniken wie eine Programmiersprache nutzen zu müssen.
  - Konfiguration mit Annotationen und Meta-Annotationen
  - REST-Unterstützung (Representational State Transfer) im MVC-Framework. Erlaubt den standardisierten Austausch von Zustandsinformationen zwischen verschiedenen Komponenten
  - *Abhängigkeitsmanagement* aus der Plattform ausgelagert, da neue Konzepte und Werkzeuge wie Maven, ivy oder OSGi existieren, die nicht Spring spezifisch sind

## 4.2. Spring Konzeption

- Komponenteneinsatz auch auf der Ebene der Applikationsentwicklung
  - Applikationsdienst wird im Zusammenwirken mehrerer Komponenten (diese werden **Applikationsobjekte** genannt) erbracht, die **innerhalb** der Applikation als Kontext konfiguriert werden.
  - dabei Rückkehr zu plain old Java Objekten (POJO)
- Schichtenmodell mit Standardisierung auf verschiedenen Abstraktionsebenen in eigenen Teilprojekten
  - Konfiguration und Basisarchitektur: Spring Core
  - Kommunikation und Datenaustausch: Spring Integration
  - Prozessmodellierung: Spring Web Flow
  - Benutzerschnittstellen (UI)-Entwicklung: Spring Faces, Spring Web
  - Verteilung und Performance: Spring JEE
  - Datenbankschicht: Spring ORM, Spring DAO
  - Querschnittsfunktionen: Spring AOP

- Komponenten **und** Kontext (als leichtgewichtiger Container) sind Gegenstand der Entwicklung
  - Lösung der Applikation von der inhärenten Bindung an einen J2EE-Server
  - Komponente trifft keine unnötigen Annahmen über den Kontext
    - Applikationsobjekte werden zur Laufzeit von außen konfiguriert.
    - Konfiguration der Komponente transparent für den Anwender, er hat Zugriff auf die Dienstschnittstelle, nicht aber auf die Zuordnung der Ressourcen. Entsprechende Konfigurationsmethoden sind nur auf der Implementierungsklasse bekannt.
    - Explizites Deployment wie bei EJB wird damit von vornherein vermieden.
- Nutzung von Dependency Injection (Inversion of Control) und Annotationen zur Kommunikation zwischen Komponente und Kontext

- Spring bietet speziellen Support für typische Laufzeitumgebungen an
  - Spring MVC – Framework für Webanwendungen
  - Spring Web Flow – Framework für Abläufe auf einer Web-Site
  - Spring Security (vormals Acegi) – Framework für Sicherheit
  - Spring Rich Client – Framework zur Verteilung der Dienst-erbringung auf Server und Client
- Unterstützung aspektorientierte Ansätze
  - Idee: Basisdienste (cross cutting concerns) werden vom Kontext angeboten und über entsprechende deklarative Schnittstellen (Interzeptoren) in die Komponente eingebunden.
  - Spring bietet eigene Annotationen u.a. im Bereich Transaktionen und Bindung an die Persistenzschicht.
- Fazit: Ein und dasselbe Komponentenmodell kann sowohl für grob granulare Fassadenkomponenten (etwa EJB) wie auch für fein granulare Applikationsobjekte verwendet werden.

### Das CORBA Komponentenmodell (CCM)

- mit CORBA 3.0 endgültig spezifizierte ambitionierte (logische) Erweiterung des EJB-Ansatzes
  - Aktuell CCM 4.0 (April 2006)
- CCM-**Anwendung** besteht aus CCM-**Komponenten**
  - EJB erfüllen die CCM-Komponenten-Spezifikation
- CCM-Komponenten sind in **Komponentenpaketen** zusammengefasst
- CCM-**Sammlungen** (CCM assemblies) enthalten Komponentenpakete zusammen mit einer **Beschreibung** der Abhängigkeiten und der Montage-Beschreibung im XML-Format
- Eine CCM-Komponente kann aus mehreren **Segmenten** bestehen
  - CCM-**Laufzeitumgebungen** laden Anwendungen segmentweise

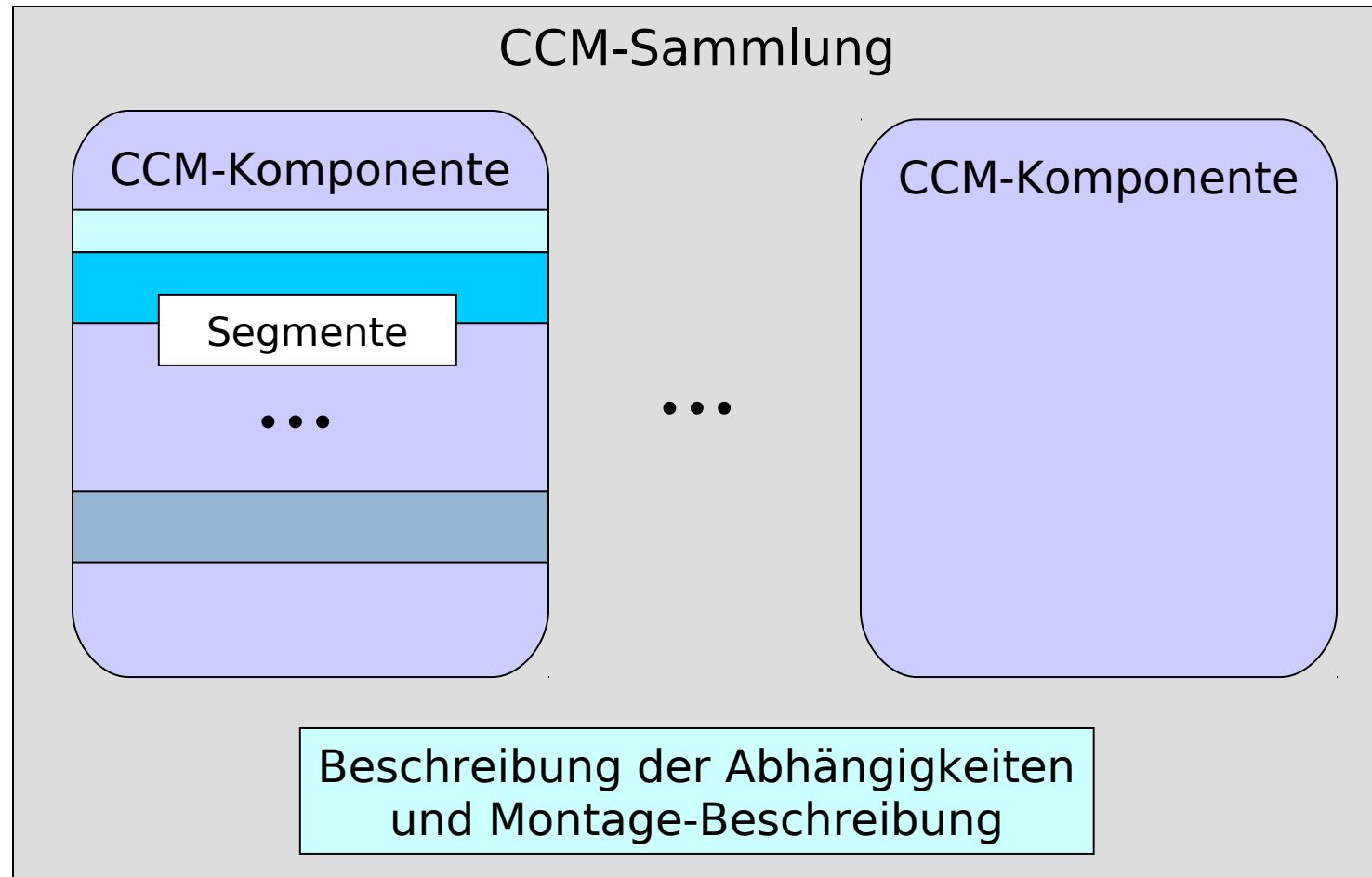
## Das CORBA Komponenten-Entwicklungsmodell

- Mit Komponentenmodell wird auch ein Entwicklungsmodell mit vier Ebenen spezifiziert
  - **Abstract Component Model:** Äußere Eigenschaften von CCM Komponenten und Definition von Komponententypen in IDL
  - **Component Implementation Framework:** Programmiermodell zur Erstellung von Komponentenimplementierungen sowie zur Beschreibung der Relationen zu Implementierungen anderer Komponenten
  - **Container Programming Model:** Beschreibung der Container-Architektur und der Schnittstellen zum ORB und zu den Komponenten.
  - **Packaging and Deployment:** Verpacken von Komponenten und Assemblies, Spezifikation von Deskriptoren sowie des Deployment-Vorgangs

### CCM in der Praxis

- Im Gegensatz zu Spring ist CCM auf grob granulare Komponentenstrukturen der Anwendungsschicht ausgerichtet und nicht auf Zusammenarbeit von leichtgewichtigen Komponenten innerhalb eines Dienstes
- CCM-Anwendungen laufen nur mit CORBA-3-konformen ORBs
  - wird auch auf der Client-Seite benötigt, wenn die ganze CCM-Funktionalität (etwa Navigation) ausgenutzt werden soll
  - CCM-Standard unterstützt aber abgerüstete Klienten auf pre-CORBA-3-Plattformen (component-unaware clients)
- Es gibt im Gegensatz zu J2EE/EJB und .NET bisher kaum Plattformen, ORBs oder Applikationsserver, die CCM vollständig unterstützen.

### Grundstruktur einer CCM-Sammlung



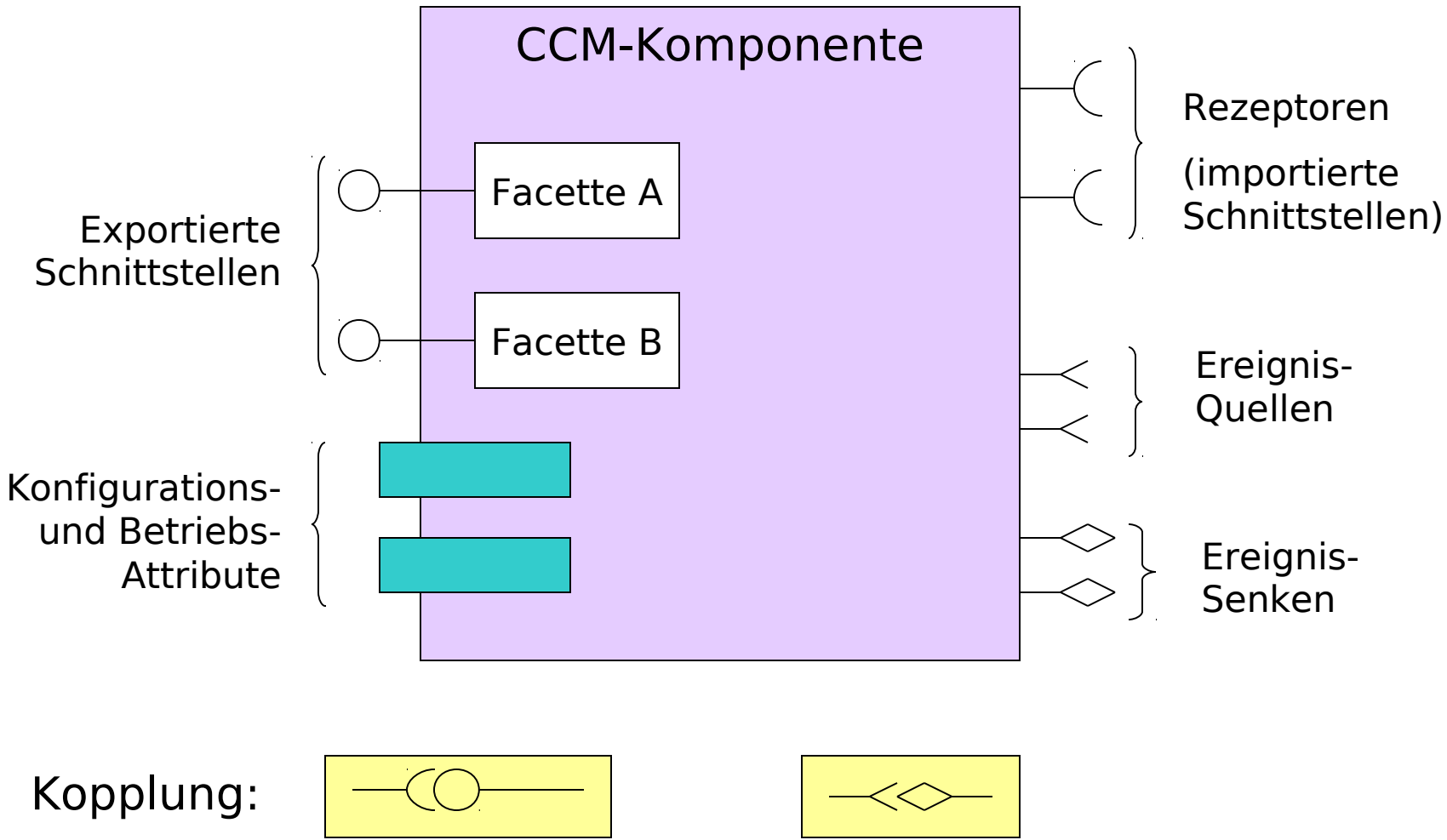
### CCM-Kategorien

- CCM-Komponenten werden (ähnlich EJB) in **Kategorien** eingeteilt
- **Service-Komponenten**
  - Instanzen sind Aufrufen zugeordnet und speichern keine Zustände über Aufrufgrenzen hinweg
- **Session-Komponenten** ( = stateful session EJB)
  - Verwaltung des Zustands innerhalb eines Transaktionszyklus (transactional session)
- **Entity-Komponenten** ( = entity EJB)
  - Instanzen haben persistenten Zustand, entsprechen Datenbankeinträgen
  - können über Primärschlüssel aus einer Datenbank gefunden werden
- **Prozess-Komponenten**
  - persistent, Lebensdauer an die des Prozesses gebunden, der bedient wird
- CCM-Anwendung enthält deklarative Informationen über Komponentenkategorien und Komponentenaufgaben

# 4.3. Das CORBA-Komponentenmodell

## Aufbau einer CCM-Komponente

Aufbau einer CCM-Komponente



### Ports von CCM-Komponenten

- **Facetten** (facets)
  - exportierte Schnittstelle, gewöhnlich einem Teilobjekt der Komponente zugeordnet
- **Rezeptoren** (receptables)
  - importierte Schnittstellen, intern Referenzen auf externe Objekte, die zum Komponentenbetrieb benötigt werden
  - connect / disconnect Operationen
  - können explizit in der Montage-Beschreibung gefordert oder zur Laufzeit eingebunden werden
- **Ereignisquellen** (event sources) und **Ereignissenken** (event sinks)
  - durch Ereigniskanäle zu verbindende Ports
- **Primärschlüssel** (nur Entity-Komponenten)
- **Konfigurations-** und **Betriebs-Attribute**
  - benannte Werte, die über **Zugriffsfunktionen** (accessor) oder **Modifizierer** (mutator) nach außen sichtbar sind

## 4.3. Das CORBA-Komponentenmodell

### Ports

- **Home-Schnittstelle**, über welche die Komponenten-Factory erreicht werden kann
  - in der Komponenten-Klasse implementiert
  - also Komponentenbegriff verschieden von dem in der Vorlesung
  - Management des Lebenszyklus von Komponenten-Instanzen
- Spezielle Facette **E-Schnittstelle** (equivalent interface), über die zwischen den Facetten der Komponente navigiert werden kann
  - Clienten müssen CORBA-3 unterstützen, um diese Navigationsmöglichkeiten auszunutzen
- **Konfigurations-Schnittstelle** (configuration interface)
  - Unterstützung der initialen Konfiguration neuer Komponenten
  - spezielles call-Signal schließt die Konfigurationsphase ab
  - erst danach sind Aufrufe der operationalen Schnittstellen möglich, Aufrufe der Konfigurations-Schnittstelle dagegen untersagt