

Vorlesung Software aus Komponenten

3. Komponenten-Modelle

apl. Prof. Dr. Hans-Gert Gräbe
Wintersemester 2007/08

Allgemeine Konzepte

- Genauere Trennung und Spezifikation von
 - Entwicklungsmodell
 - Interaktionsmodell
 - Plattformmodell
 - Packaging and Deployment
- Fokus auf serviceorientierte Architekturen
 - stärkere Trennung von **Geschäftslogik** innerhalb der Komponenten und **Interaktionslogik** im Framework
- Fokus auf eng gekoppeltes Zusammenspiel der Interaktionslogik auf einer gemeinsamen BS-Basis (Applikationsserver, Client-PC)
 - Spring-Framework
 - CCM – Corba Component Model
 - OSGi – Open Service Gateway Initiative

Spring Framework

- Spezielles Java-Framework auf der Enterprise-Ebene
- Geschichte:
 - Okt. 2002: Grundlagen werden von Rod Johnson im Rahmen seiner Buchreihe „Expert One-On-One“ entwickelt, der auch den Sourcecode des Framework-Gerüsts als freien Download zur Verfügung stellt.
 - Febr. 2003: Code als Sourceforge-Projekt
 - Juni 2003: erstes Release, unter der Apache 2.0 License
 - März 2004: Milestone Release 1.0
 - seitdem fand das Framework sehr schnelle Verbreitung
 - aktuelle Version 2.5 (November 2007)
- 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.

- 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)
- 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
 - „Don't call us, we call you.“

Spring Framework

- Spring bietet speziellen Support für typische Laufzeitumgebungen an
 - Spring MVC – Framework für Webanwendungen
 - Spring Web Flow – Framework für Abläufe auf einer Web-Site
 - Acegi – Framework für Sicherheit
 - Spring Rich Client – Framework zur Verteilung der Dienstleistung 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

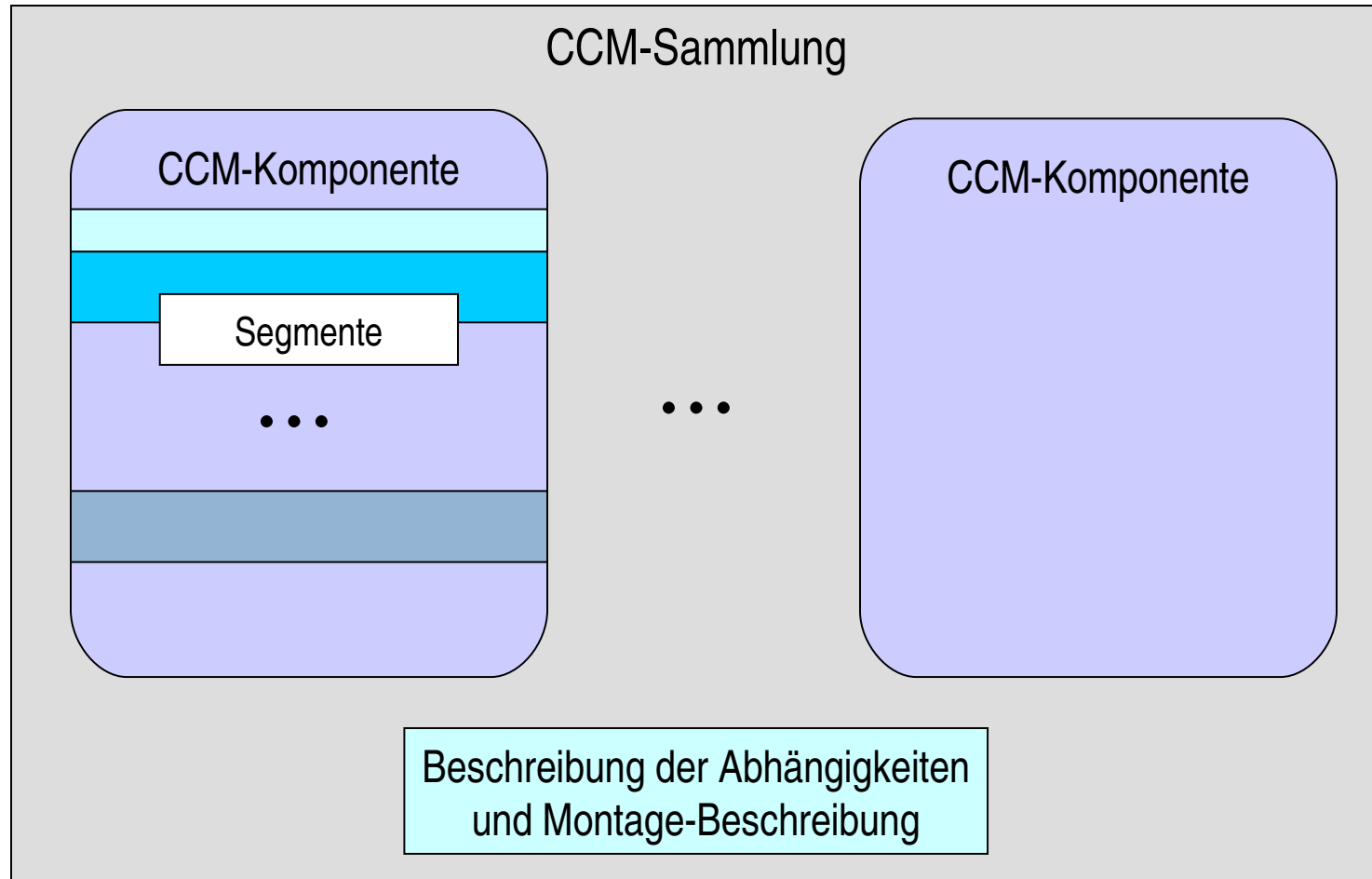
Das CORBA Komponenten-Entwicklungsmodell

- 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

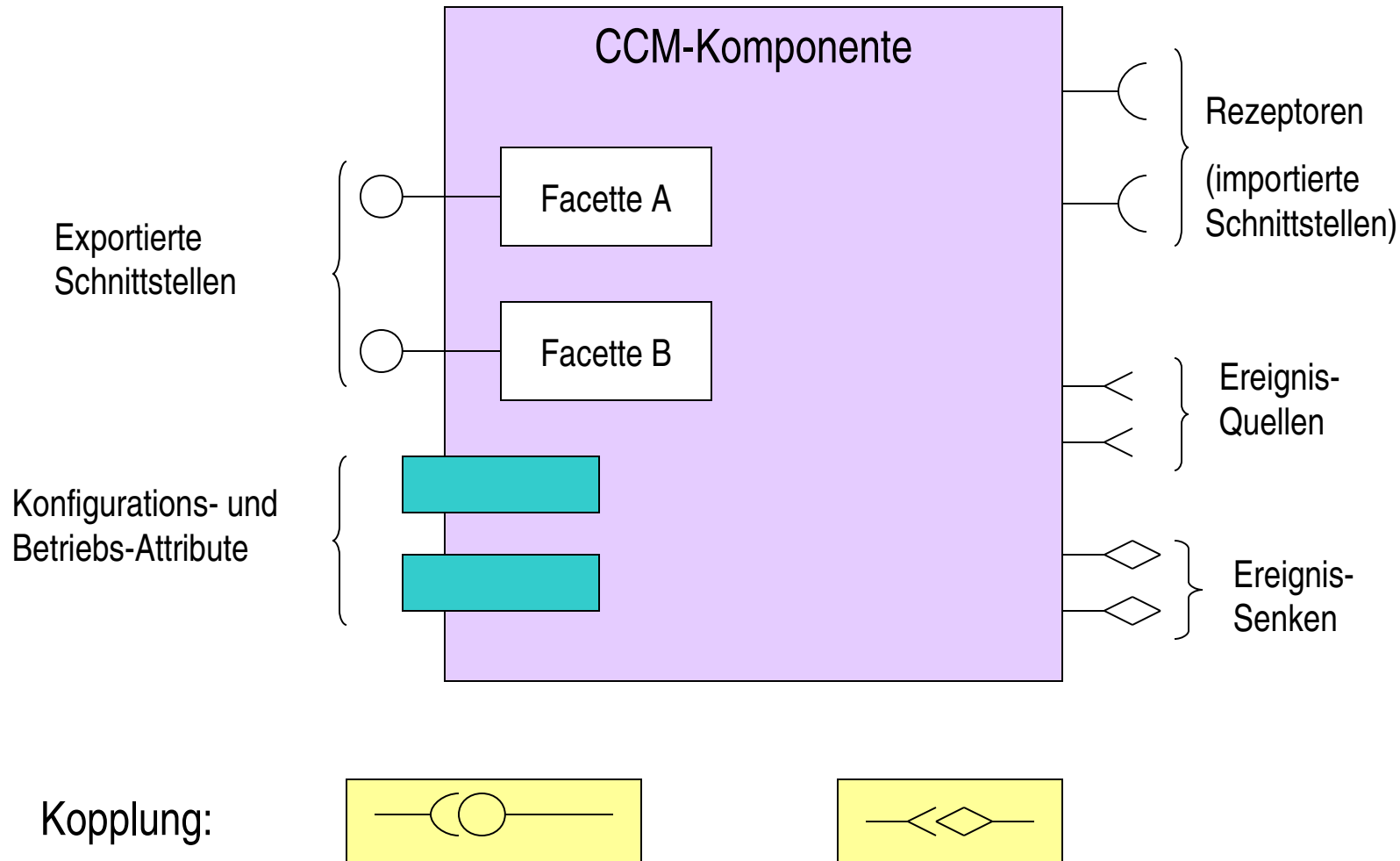


CCM-Kategorien

- 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

Aufbau einer CCM-Komponente

Aufbau einer CCM-Komponente



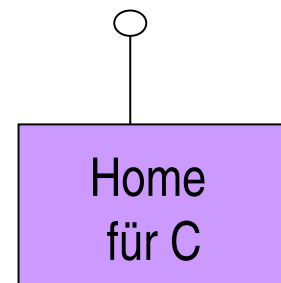
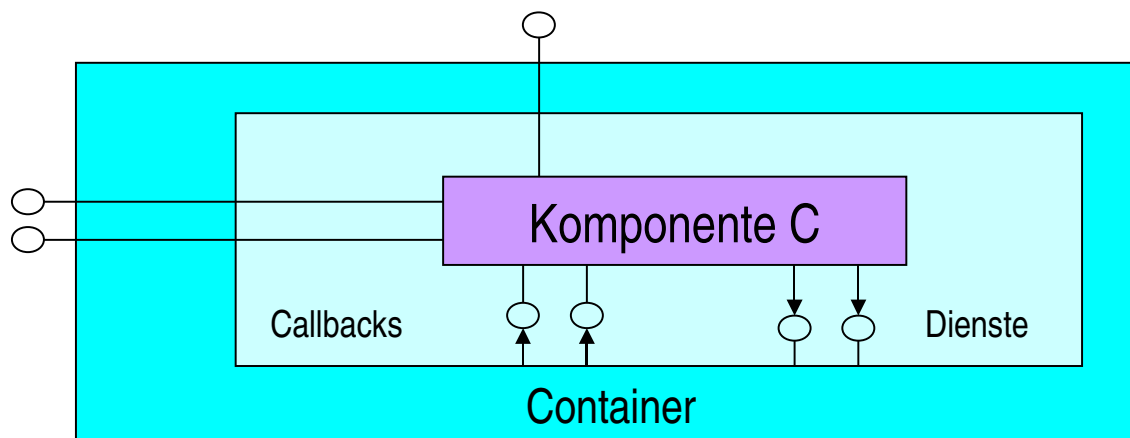
Ports von CCM-Komponenten

- **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

- **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
 - analog der IUnknown-Schnittstelle im COM-Konzept
 - 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

CCM-Container

- CORBA 3 definiert ein **Komponenten-Implementierungs-Gerüst** (component implementation framework, CIF)
 - Generatoren erzeugen aus Eingaben im **CIDL-Format** (component implementation description language) Code, der den Komponentencode ergänzt
- Jede Komponenten-Instanz ist in einem **CCM-Container** untergebracht, über den die Anbindung der Facetten und Rezeptoren erfolgt. Rezeptoren und Dienste können in einem solchen Container per Callback gebunden sein.



CCM-Container

Navigation zwischen den Schnittstellen aller Komponenten im Container

- Container hat Kontrolle über exportierte und importierte Schnittstellen seiner Komponenten
- kann nur von CORBA-3-konformen Clients genutzt werden

CCM-Container ist ein spezieller POA

- vorgefertigten Basisdienste (pre-packaged object services)
- **Transaktionsdienst:** durch Container oder selbst
 - Komponente: Beschreibung enthält die Transaktionsanforderungen (supported, required, required new, not supported)
 - Container: Ausführung der Transaktionen entsprechend der Spezifikation der einzelnen Komponenten
- **Persistenzdienst:** durch Container oder selbst
 - Komponente: Beschreibung der Anforderungen im PSDL-Format (persistent state description language)

CCM-Container

- **Sicherheitsdienst:** Zugriffsrechte können im CIDL-Format beschrieben und durch den Container geprüft werden
- **Benachrichtigungsdienst:** Aufbau und Verwaltung von Ereigniskanälen