Vorlesung
Component Ware und Web-Services

- Komponentenkonzepte -

10. Komponentenmodelle

Prof. Dr. Hans-Gert Gräbe, F. Schumacher
Wintersemester 2003/2004

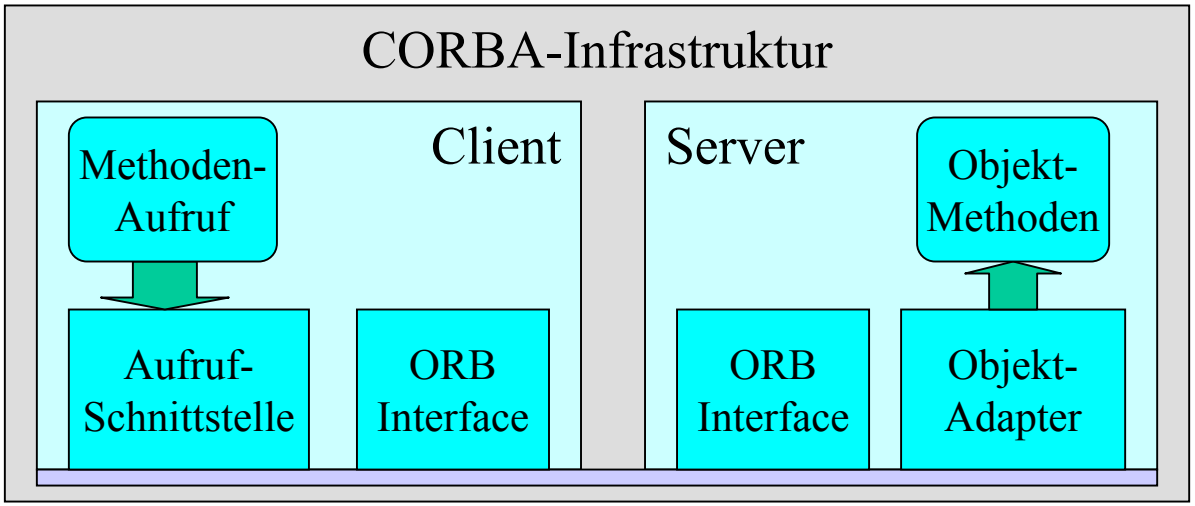
Inhalt

- Der portable Objektadapter (portable object adapter, POA)
  - Aufrufgenerierung über Stummel, ORB und POA
  - Generierung von Stummel- und Skelettklassen aus Schnittstellenbeschreibung in CORBA-IDL
- Ein Beispiel
- Das CORBA Komponenten-Modell (CCM)
  - CCM-Sammlungen
  - CCM-Kategorien
  - Aufbau einer CCM-Komponente
  - CCM-Container

POA

Portable Object Adapter (POA)

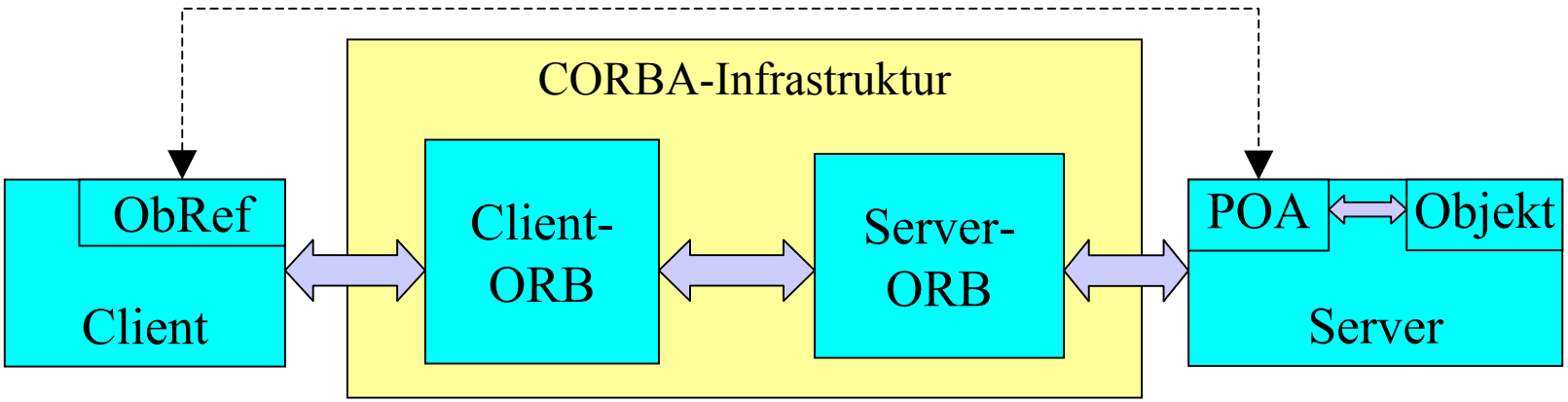
- Vermittlungseinheit zwischen ORB und aktueller Implementierung eines Objekts, die eingehende Aufrufe und zurückgegebene Resultate verwaltet
- Eine POA-Instanz bedient eine **Gruppe von Objekten**
- Jeder von einem ORB bediente Serverprozess hat wenigstens eine POA-Instanz, kann aber zu jedem Objekt eine eigene POA-Instanz haben
- zuständig für Aktivierung, Deaktivierung, Ressourcenverwaltung, Servantenmapping, Skalierung



POA

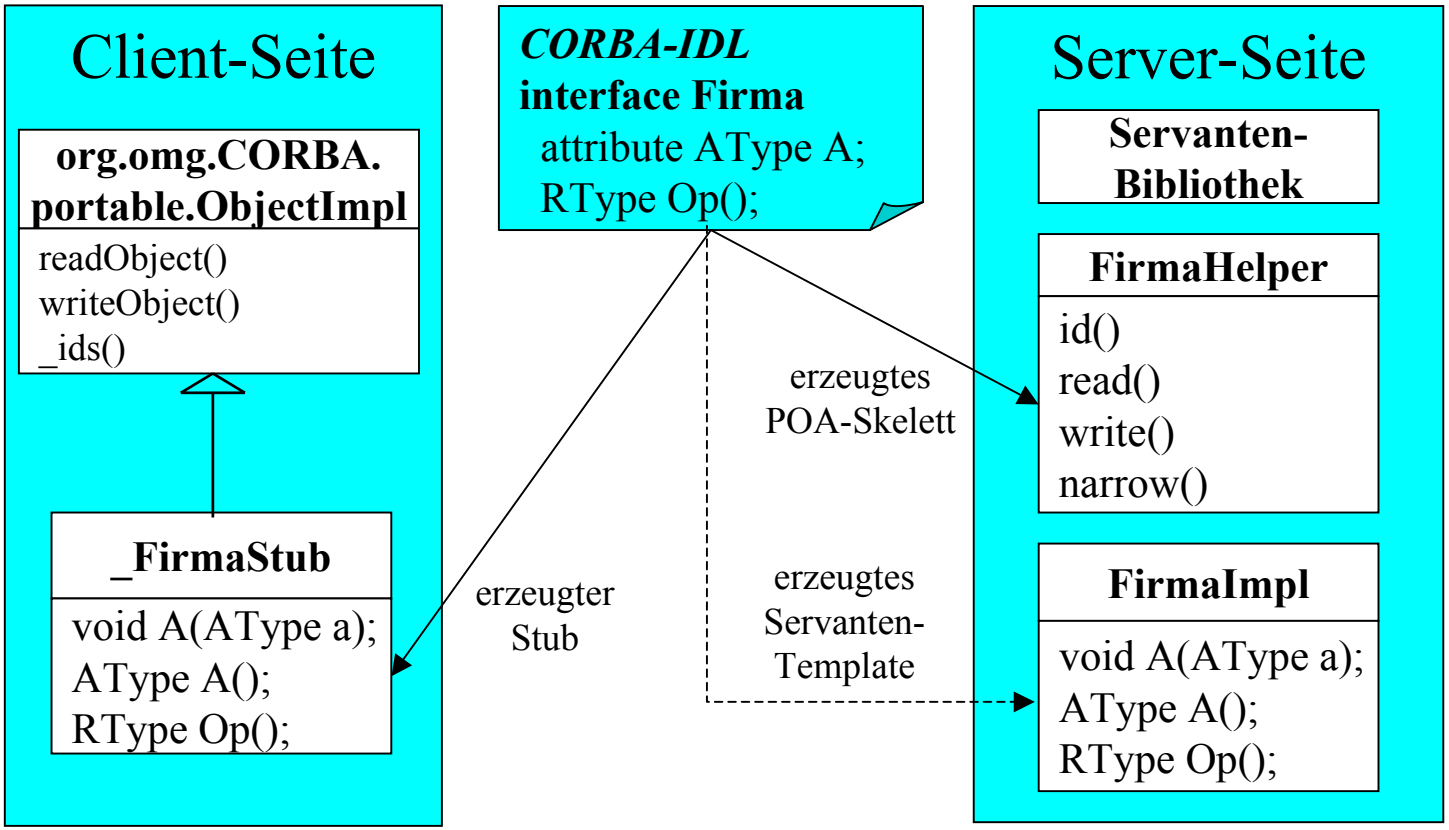
Aufrufzyklus

1. POA-Name wird während Serverinitialisierung **generiert** und beim Server-ORB **registriert**
2. POA generiert für Client-Stub, der Dienst in Anspruch nehmen will, eine lokale Objektreferenz **ObRef** mit Information über Server-ORB und POA
3. Client setzt **Aufruf** lokal an ObRef ab
4. Client-ORB findet über ObRef den **Server-ORB** heraus und übermittelt POA-Namen, der ObRef erzeugt hat, die ID aus ObRef und den Aufruf
5. Server-ORB findet über Registrierung die **POA-Instanz** heraus und übergibt Objekt-ID und Aufruf
6. Objekt-ID wird im POA-Kontext aufgelöst und Aufruf an geeigneten **Servanten** weitergeleitet.



POA

Erstellung von Client-Stub und POA-Skelett aus IDL-Beschreibung



Beispiel

Entwicklung einer CORBA–Anwendung unter JAVA
(aus "Lehrbuch der Softwaretechnik" von Helmut Balzert)

Definition der Schnittstelle mittels CORBA-IDL

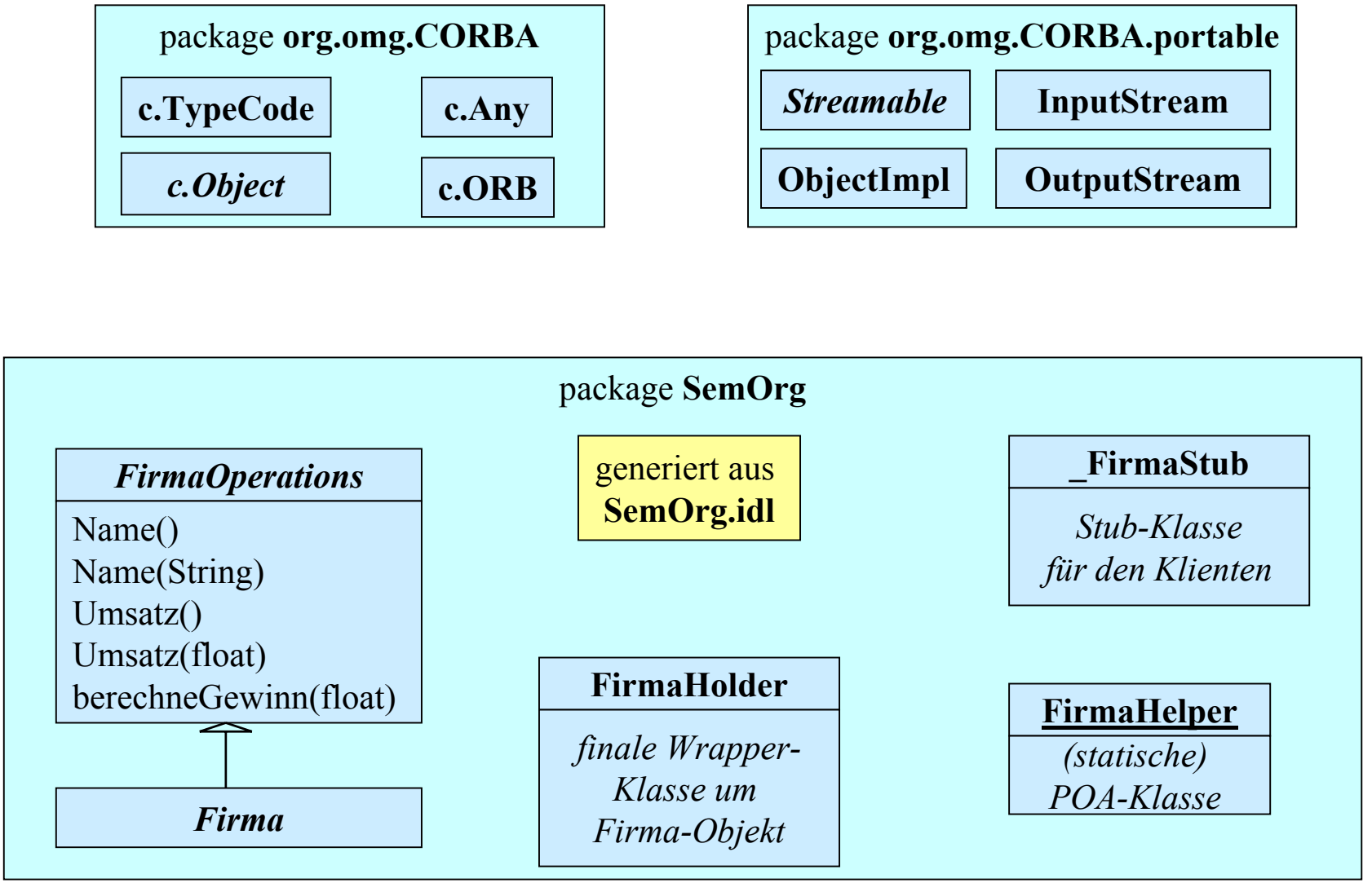
```
//SemOrg.idl
module SemOrg {
    //Schnittstelle der Klasse Firma
    interface Firma {
        attribute string Name;
        attribute float Umsatz;
        //Operationssignatur
        float berechneGewinn( in float Kosten);
    };
};
```

Beispiel

Datei mit IDL–Compiler übersetzen

- IDL-to-Java Compiler **idlj** 1.3 generiert aus SemOrg.idl ein **package SemOrg** mit den Klassen
  - **interface FirmaOperations**
    - generierte Schnittstelle zum Servanten
  - **interface Firma extends FirmaOperations**
    - (vorerst leere) Erweiterung von FirmaOperations
  - **final class FirmaHolders**
    implements org.omg.CORBA.portable.Streamable
    - Wrapper um ein Firma-Instanz, die Serialisierung implementiert
  - **abstract class FirmaHelper**
    - die (Methoden für die) POA-(Ober)-Klasse
  - **class _FirmaStub**
    extends org.omg.CORBA.portable.ObjectImpl
    - die (Ober)-Klasse für den Client-Stub

Beispiel


package org.omg.CORBA
c.TypeCode
c.Any
c.Object
c.ORB
package org.omg.CORBA.portable
Streamable
InputStream
ObjectImpl
OutputStream
package SemOrg
FirmaOperations
Name()
Name(String)
Umsatz()
Umsatz(float)
berechneGewinn(float)
Firma
generiert aus SemOrg.idl
_FirmaStub
Stub-Klasse für den Klienten
FirmaHolder
finale Wrapper-Klasse um Firma-Objekt
FirmaHelper
(statische) POA-Klasse

Beispiel

Implementierung der Servantenklasse

```
package SemOrg;
public class FirmaImpl implements Firma {
    private String derName;
    private float derUmsatz;
    public FirmaImpl() { ... }
        //Attribute initialisieren
    public float berechneGewinn(float Kosten)
        { return this.Umsatz – Kosten; }
    //get-/set-Operation für Attribut Name
    public String Name()
        { return this.derName }
    public void Name(String neuerName)
        { this.derName = neuerName; }
    //analog für Umsatz ...
}
```

Beispiel

Initialisierung der Serverseite (1/2)

- Server-ORB initialisieren (c = **org.omg.CORBA**)
  - c.ORB orb = c.ORB.init(args, null);
- vom ORB Referenz auf einen Objektadapter holen, RootPOA ist fest definierter Name, den ORB immer kennt
  - c.Object tempPOA = orb.resolve_initial_references("RootPOA");
- Umwandlung der Referenz nach Typ ops.POA (ops = **org.omg.PortableServer**)
  - ops.POA poa = ops.POAHelper.narrow(tempPOA);
- Objektadapter aktivieren, damit er Anfragen entgegennehmen kann
  - poa.the_POAManager().activate();
- einen Servanten aktivieren
  - FirmaImpl tempFirma = new FirmaImpl();
- Erzeugen einer CORBA-Objektreferenz auf den Servanten
  - c.Object dieFirma = poa.servant_to_reference(tempFirma);

Beispiel

Initialisierung der Serverseite (2/2)

- Verbindung der Objektreferenz mit einem Namensdienst
  - c.Object tempNameServer =
    orb.resolve_initial_reference("NameService");
  - NamingContextExt nameServer =
    NamingContextExtHelper.narrow(tempNameServer);
- Firmen-Objekt unter dem Namen "eineFirma" bekannt machen.
  - nameServer.bind(nameServer.to_name("eineFirma"), dieFirma);

Beispiel

Initialisierung der Client-Seite

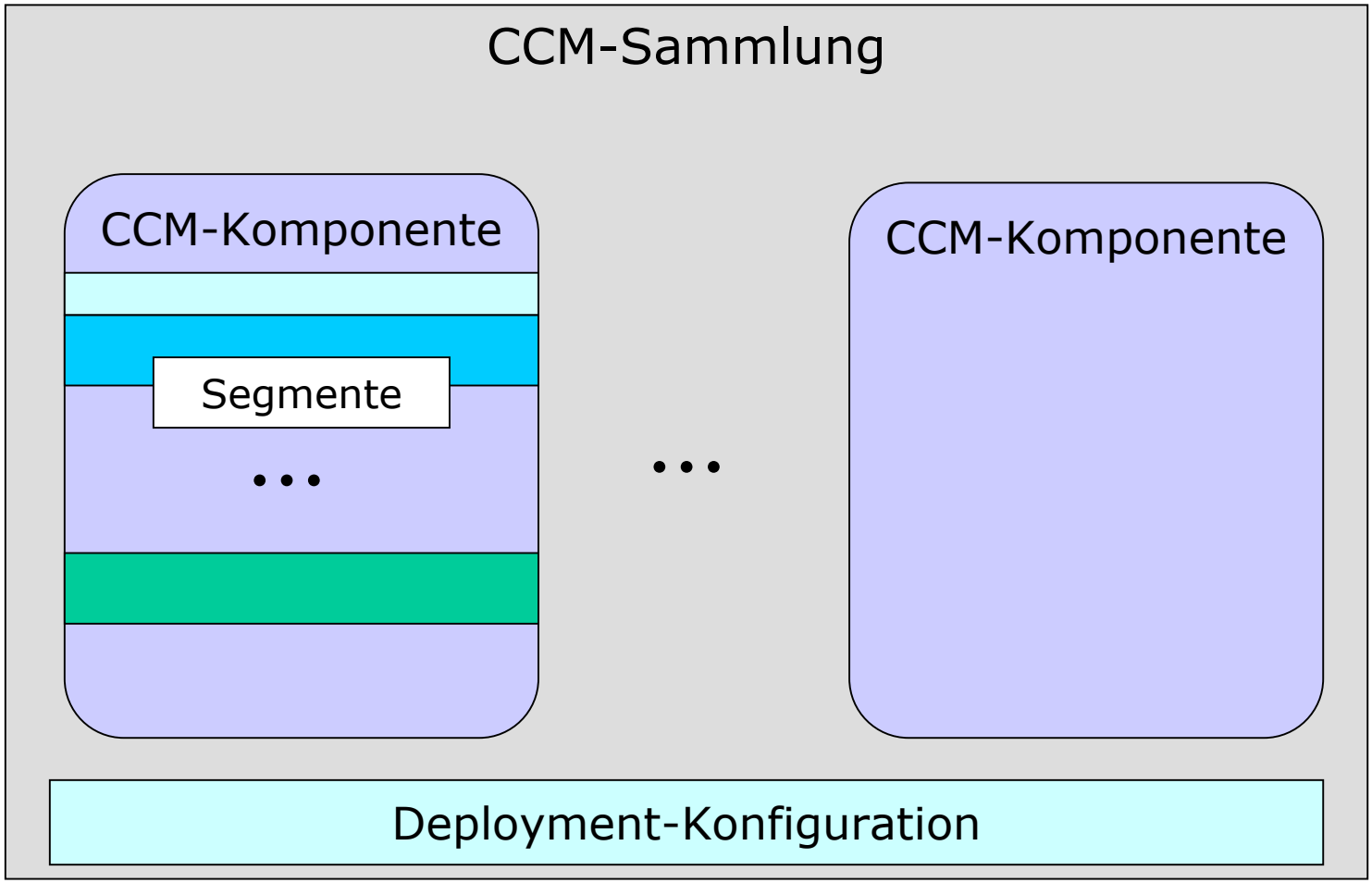
- den Client-ORB initialisieren
  - c.ORB orb = c.ORB.init(args, null);
- Namensdienst initialisieren
  - c.Object tempNameServer = orb.resolve_initial_reference("NameService");
  - NamingContextExt nameServer = NamingContextExtHelper.narrow(tempNameServer);
- Objekt-Referenz vom Namensdienst anfordern (die von POA dort abgelegt wurde)
  - c.Object tempFirma = nameServer.resolve(nameServer.to_name("eineFirma"));
  - eineFirma = FirmaHelper.narrow(tempFirma);
- Aufrufe durchführen
  - etwa System.out.println(eineFirma.Name());

Das CORBA Komponentenmodell (CCM)

- mit CORBA 3.0 endgültig spezifiziert
- Ambitionierte (logische) Erweiterung des EJB-Ansatzes
  - derzeit allerdings vor allem auf dem Papier
- 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 Deployment-Konfiguration im XML-Format
- Eine CCM-Komponente kann aus mehreren **Segmenten** bestehen
  - CCM-**Laufzeitumgebungen** laden Anwendungen segmentweise
- 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)

CCM

Grundstruktur einer CCM-Sammlung



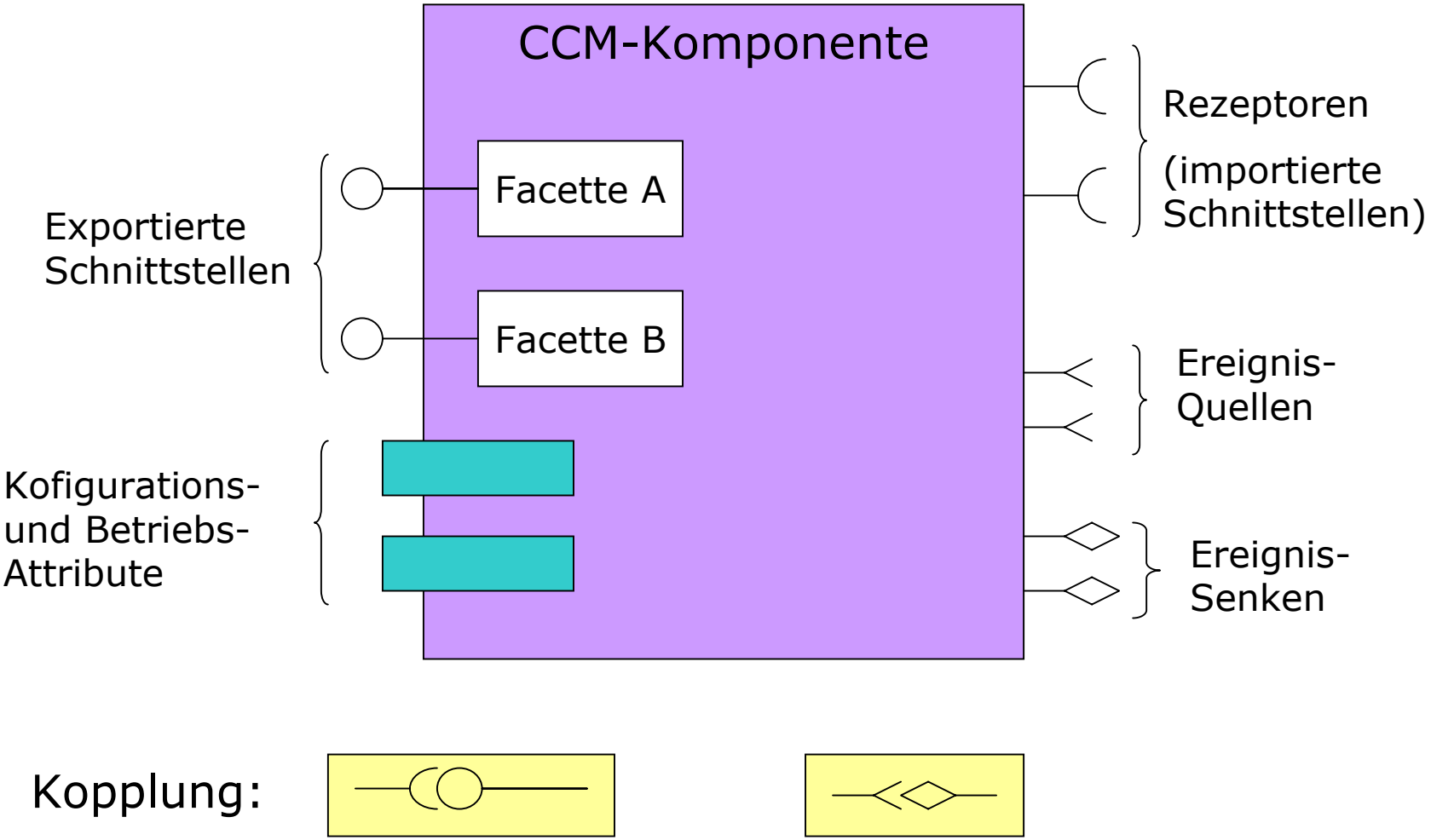
CCM

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

CCM

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 Deployment-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

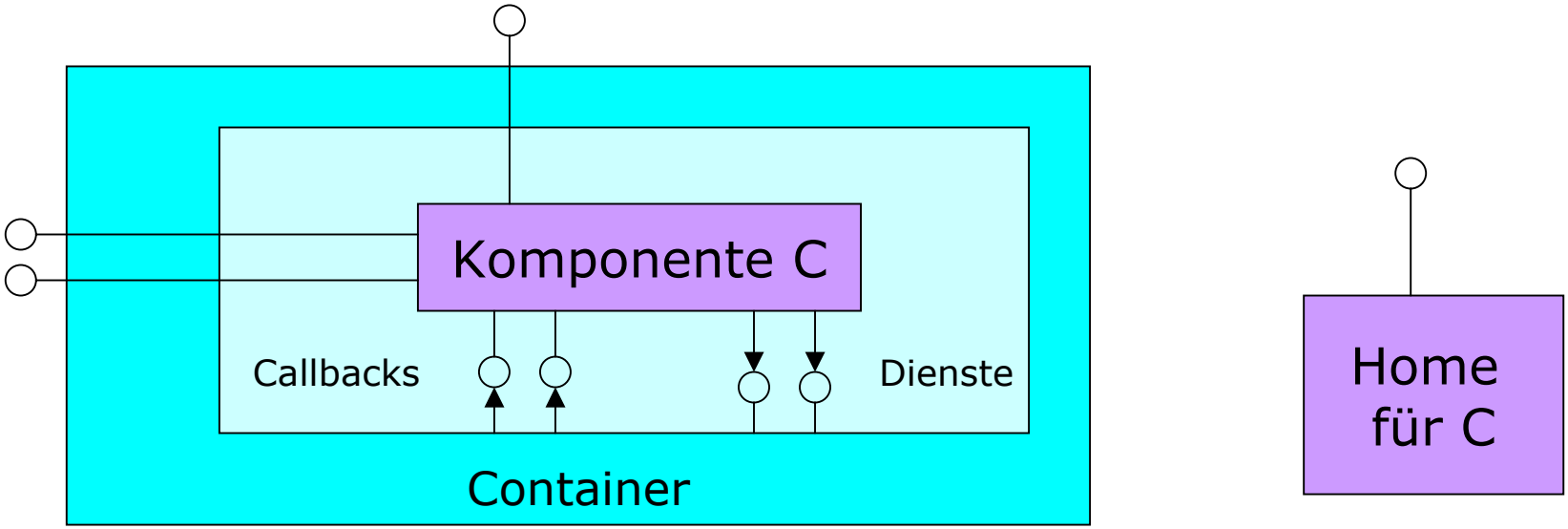
Ports von CCM-Komponenten (Fortsetzung)

- **Home-Schnittstelle**, über welche die Komponenten-Factory erreicht werden kann
  - in der Komponenten-Klasse implementiert
  - 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

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

- Navigation zwischen den Schnittstellen aller Komponenten im Container
  - Container hat Kontrolle über exportierte und importierte Schnittstellen seiner Komponenten
  - kann nur von CORBA-3-konforme Clients genutzt werden

- CCM-Container ist ein spezieller POA mit vorgefertigten Basisdiensten (pre-packaged object services)
  - **Transaktionsdienst**: durch Container oder selbst
    - Komponente: Beschreibung enthält die Transaktionsanforderungen (supported, requireed, 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 (pesistent state description language)
  - **Sicherheitsdienst**: Zugriffsrechte können im CIDL-Format beschrieben und durch den Container geprüft werden
  - **Benachrichtigungsdienst**: Aufbau und Verwaltung von Ereigniskanälen

Zusammenfassung

- Entwicklungsumgebung, die auf Programmierer von Geschäftsanwendungen ausgerichtet ist
  - Umgebung, in welcher diese ihre speziellen Erfahrungen bestmöglich entfalten können
  - Konzentration auf die Logik auf Geschäftsprozess-Ebene möglich
- Stellt eine strukturierte CORBA-Laufzeit-Infrastruktur (packaged CORBA-computing infrastructure) zur Verfügung
  - höhere Abstraktionen für Persistenz, Transaktionalität, Sicherheit, Ereignisbehandlung
  - Zusammenbau durch visuell orientierte oder Scripting-Werkzeuge (CORBA scripting language)