Vorlesung
Software aus Komponenten

4. Moderne Komponentenkonzepte

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2011/12

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- 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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- 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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- 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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

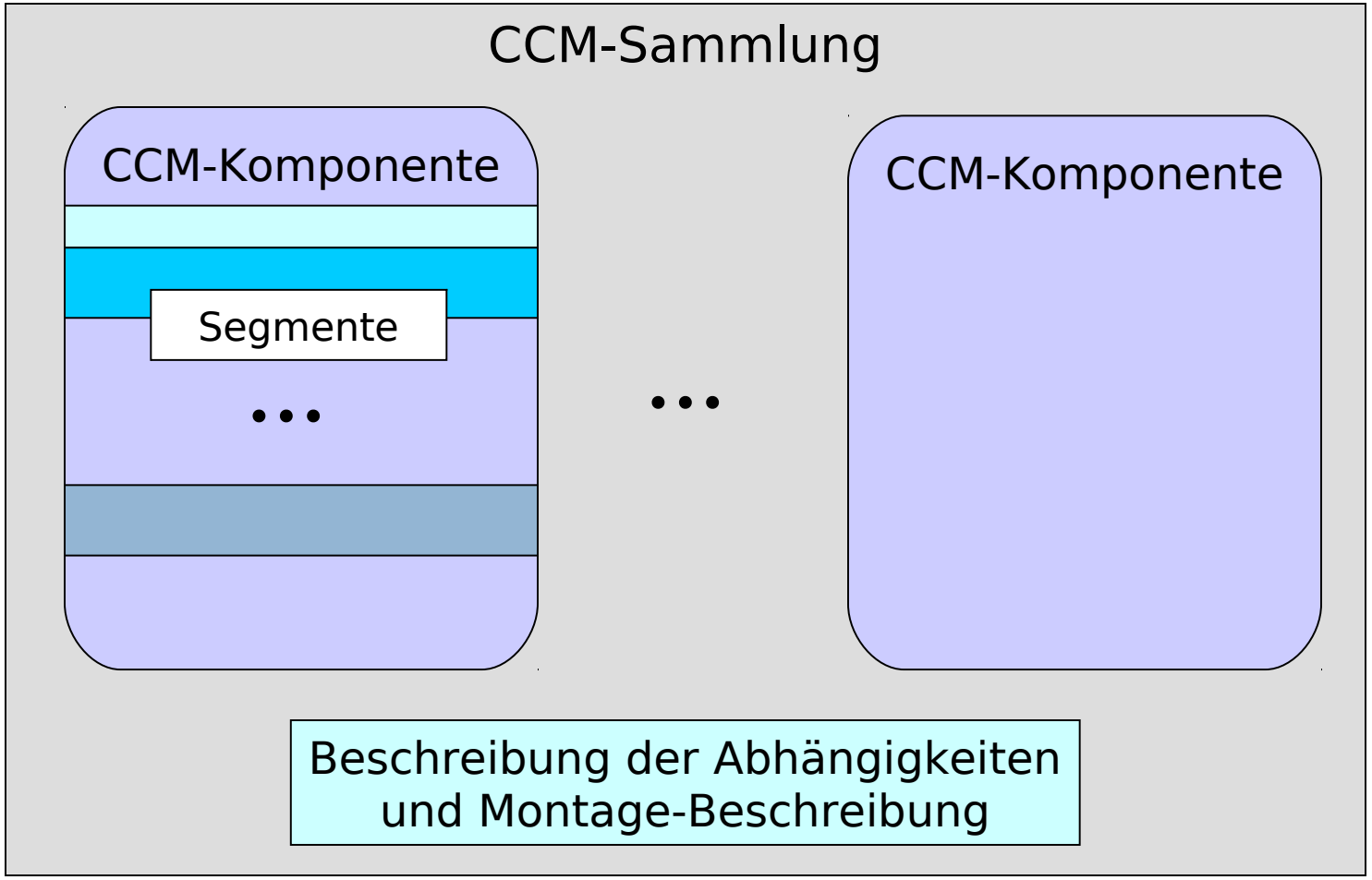
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Grundstruktur einer CCM-Sammlung

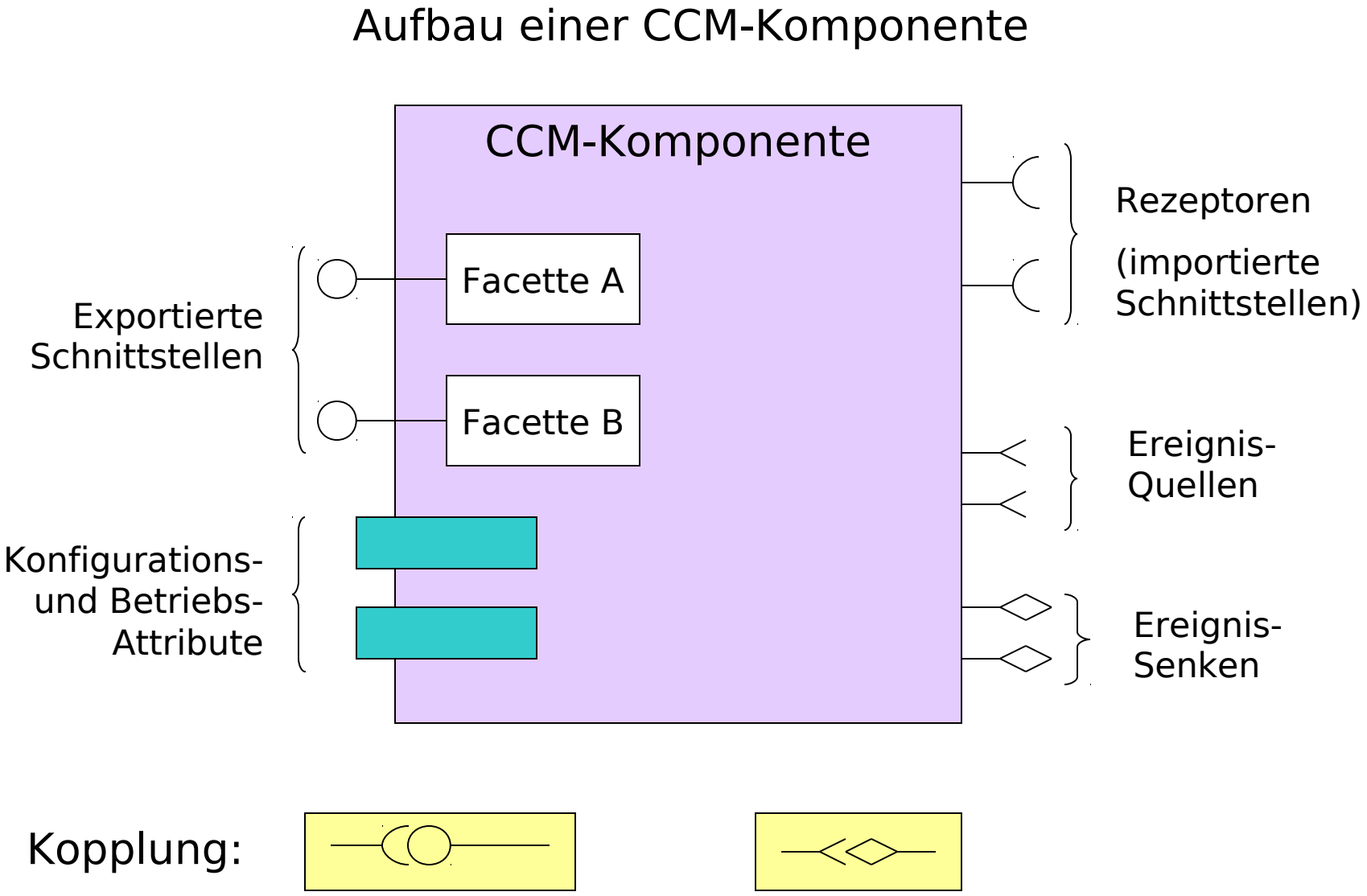


CCM-Kategorien

- 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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

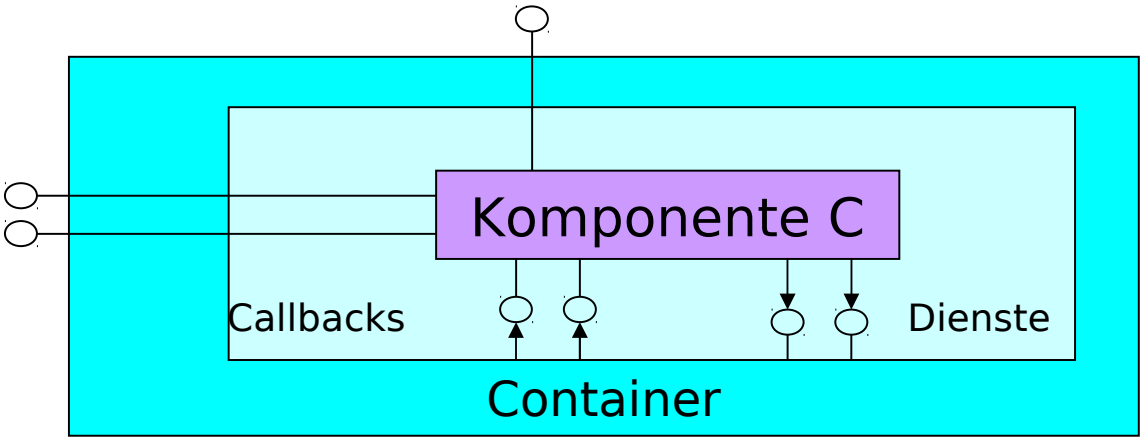
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

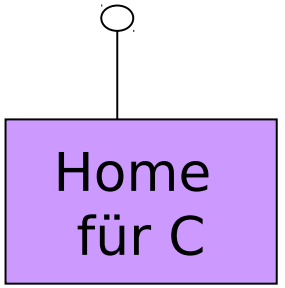
- **Home-Schnittstelle**, über welche die Komponenten-Factory erreicht werden kann
  - in der Komponenten-Klasse implementiert
  - also Komponentenbegriff verschieden von dem in der Vorlesung
  - Management des Lebenszyklus von Komponenten-Instanzen
- Spezielle Facette **E-Schnittstelle** (equivalent 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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.





- Der CCM-Container ist ein spezieller POA

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

Vorlesung
Software aus Komponenten

5. Komponentenkonzepte im Vergleich

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2011/12

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentenkonzepte und Anforderungen im Vergleich

Eine Zusammenfassung

- Komponententechnologie und Softwaretechnik
- Komponentenkonzepte im Vergleich
- Konvergenz der Konzepte
- Differenzen der Konzepte
- Komponenten und Objekte
- Kontraktspezifikationen für Komponenten
- Komponenten und Softwaretechnik
- Komponenten-Montage
- Komponenten und Berufsprofile

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Softwaretechnik als Ingenieurtechnik

Ingenieurtechnik

- Standards, Vorgehensweisen und Zusammenhänge, die beim Bearbeiten einer Aufgabenstellung aus dem jeweiligen Gebiet von einer qualifizierten Fachkraft zu berücksichtigen sind.
- technologische Einbettung der für das jeweilige Gebiet verfügbaren Technik

Softwaretechnik ist eine ingenieurtechnische Disziplin

- Lehre von Planung, Erstellung, Einsatz, Wartung und Weiterentwicklung von komplexen Software-Systemen in einem arbeitsteiligen Prozess
- und den dabei zweckmäßig zum Einsatz kommenden Prinzipien, Methoden und Werkzeugen. [Balzert]

Im Zentrum steht dabei die **Beherrschung der Komplexität** der Anforderungen aus dem Lebenszyklus von Software-Systemen.

Als typische Arbeitsschritte haben sich bewährt

- Anforderungsanalyse, Entwurf, Modellierung, Realisierung, Montage, Einsatz

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponententechnologie aus ingenieurtechnischer Sicht

Die **Nutzung von Komponenten** ist ein Charakteristikum jeder entwickelten Ingenieurtechnik

- Neue Produkte werden aus vorgefertigten, standardisierten, dem Stand der Technik entsprechenden Bestandteilen nach allgemein anerkannten Standards und eigener Kreativität zusammengebaut.
- Form der Komplexitätsreduktion

Komponententechnologie hat zum Gegenstand das Zusammenspiel von Komponentenentwicklung und Komponenteneinsatz

- Rolle: **Komponentenentwickler**, Perspektive: Zulieferer-Sicht
  - Komponenten für möglichst breites Einsatzfeld entwickeln
- Rolle: **Komponentenmonteur**, Perspektive: Dienstleister-Sicht
  - Komposition von Anwendersystem aus geeigneten Komponenten

Ansatz findet über mehrere hierarchische Ebenen der Komposition statt

- Treiber – Betriebssysteme
- Laufzeitbibliotheken – Hochsprachen-Programme
- der in dieser VL besprochene Komponentenbegriff

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponententechnologie und Softwaretechnik

Ziel: Montage eines IT-Systems, das als **verteilte Anwendung** auf einem System von mehreren miteinander verbundenen Rechnern aus **Komponenten unterschiedlicher Hersteller** konzipiert ist.

Anforderungen:

- formal fundiertes **Komponentenkonzept** als Basis
- **Beschreibungstechniken** für derartige Komponenten
- Entwicklung eines **Prozessmodells** zur Entwicklung, Verwaltung und Zusammensetzung von Komponenten
  - Unterstützung der Zuweisung verschiedener Rollen
- **Werkzeuge**, welche die Beschreibung und das Prozessmodell unterstützen
  - zur Systemgenerierung selbst
  - zur Dokumentation
  - zur Verifikation und Sicherung wichtiger und kritischer Systemeigenschaften

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zwei grundlegende Herangehensweisen

Eng gekoppelte Architektur (CORBA, Java, EJB, .NET, Spring)

- Laufzeitsystem als Infrastruktur, in der Informationen über Objektinstanzen ausgetauscht werden, in denen Zustand und Funktionalität des Gesamtsystems lokal gespeichert sind.
- fein granulares Konzept, Technik der Interaktion steht im Fokus
- Erweiterung objektorientierter Ansätze von einer Einzelplatzanwendung auf eine verteilte Umgebung
- grundlegendes Konzept: RPC und dessen Verallgemeinerungen

Lose gekoppelte Architektur (Webservices)

- hohe Autonomie der Rechner, die nachrichtengesteuert gegenseitige „Dienste“ erbringen
- grobgranulares Konzept auf höherer Abstraktionsstufe
- näher am Geschäftsprozess-Modell
- in dieser Vorlesung nicht besprochen

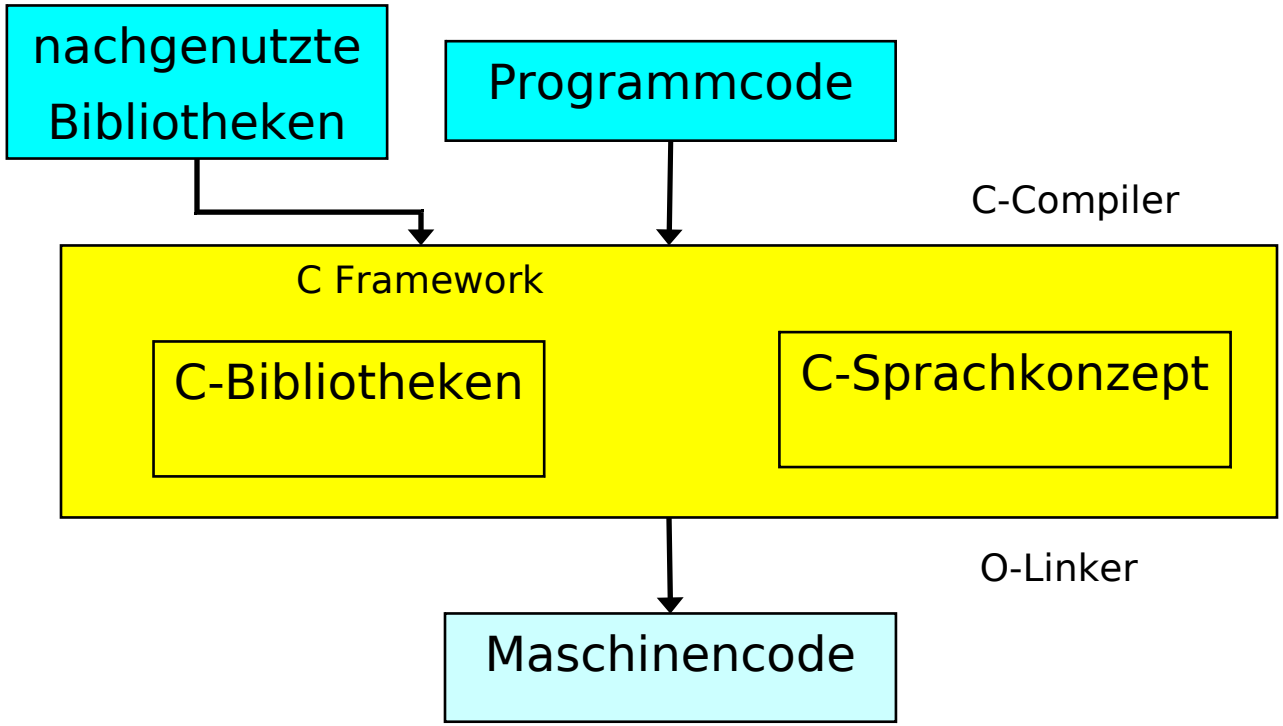
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentenmodelle auf Quellcode-Ebene:
Aufbau von Anwendungen aus Software-Bausteinen

Ziel: Sicherung plattform- und sprachübergreifender Kompilierungskompatibilität
Anwendungsbereich: Desktop, Basiskomponenten
Grundlage: Gemeinsame Designprinzipien
Beispiele: C, Java, .NET

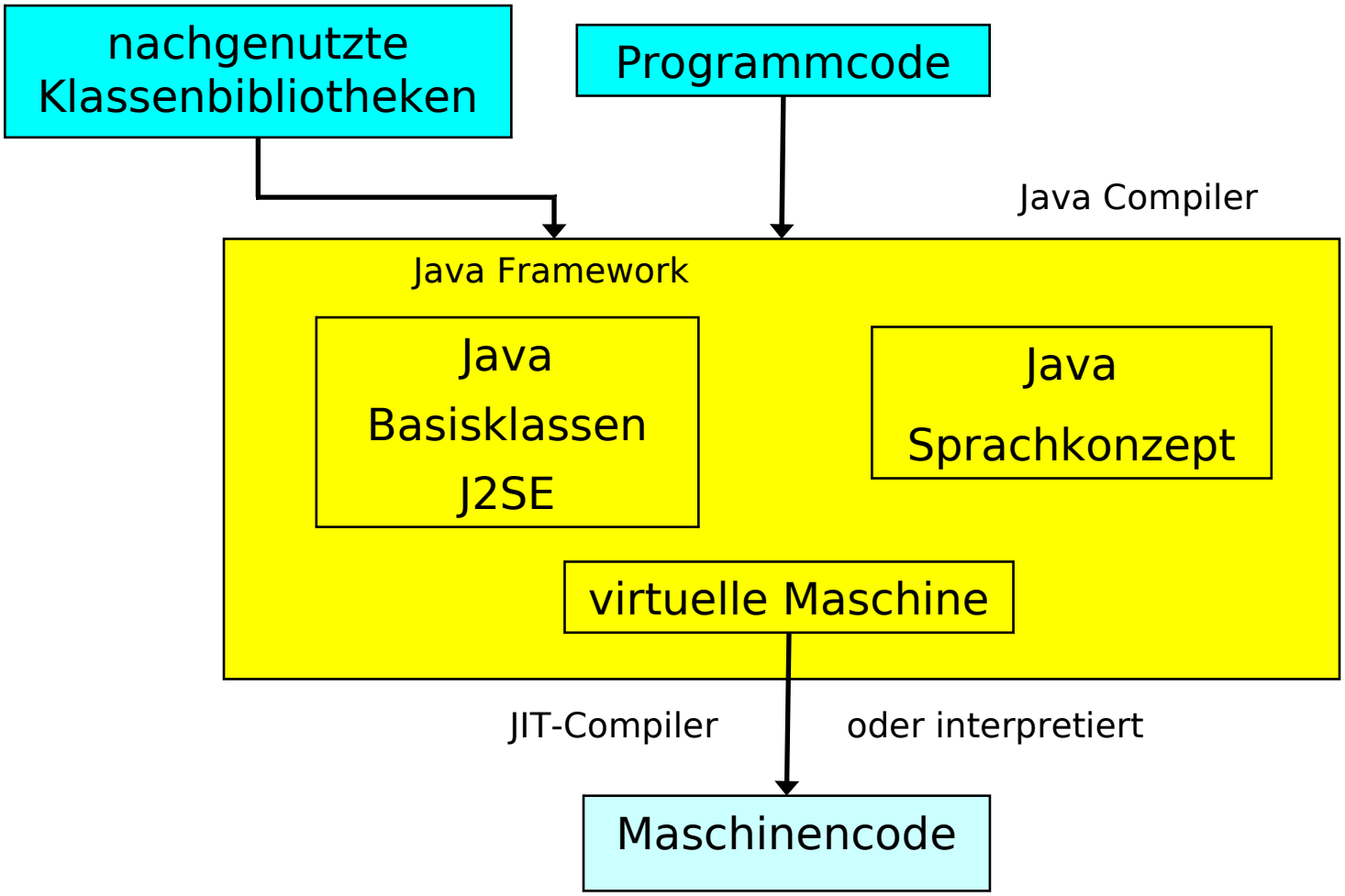
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Eine Sprache, eine Plattform: C



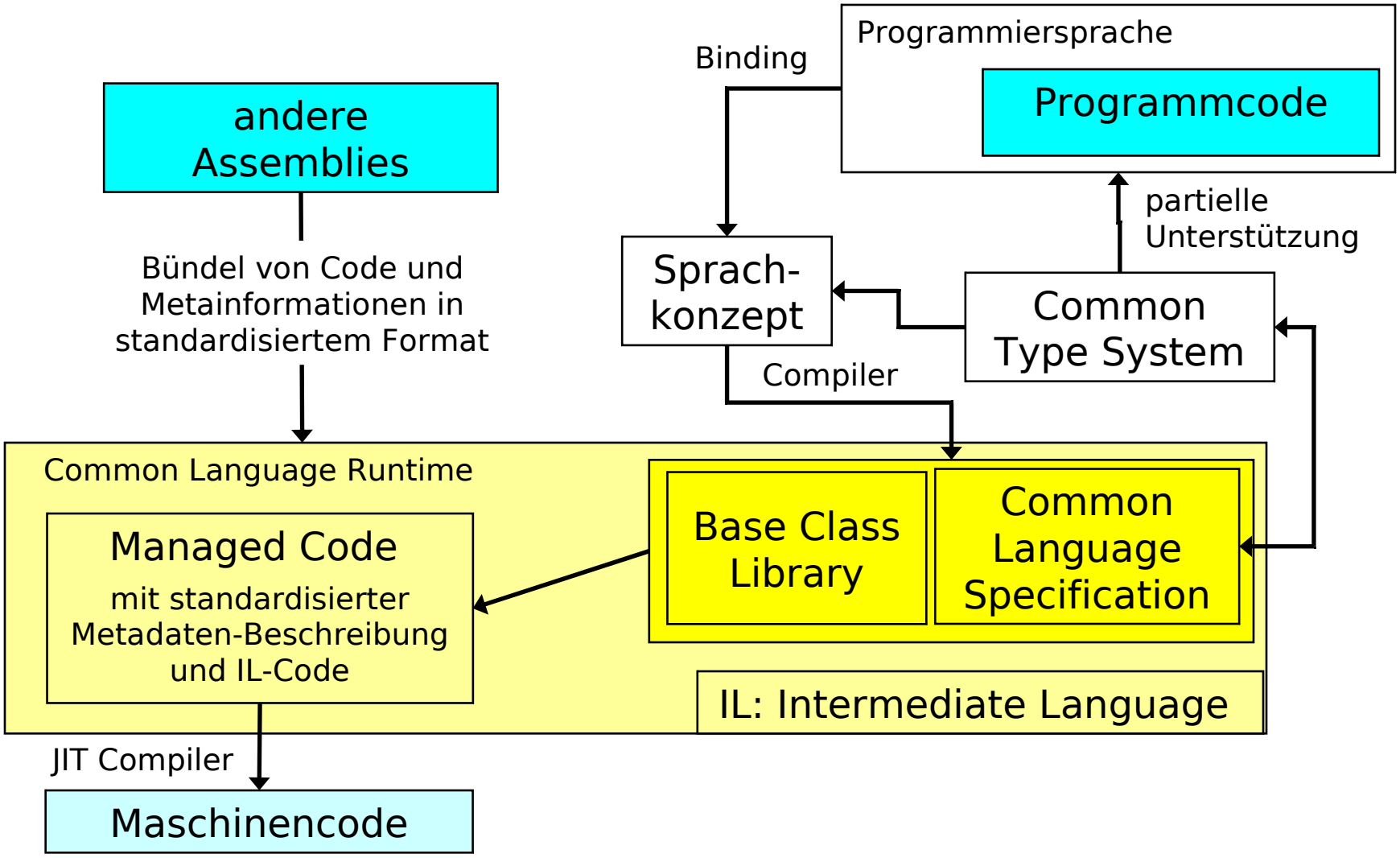
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Eine Sprache, mehrere Plattformen: Java



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Mehrere Sprachen, mehrere Plattformen: .NET



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentenmodelle für verteilte Anwendungen:

Aufbau von Anwendungen aus Komponenten

Ziel: Integration von Diensten in eine standardisierte verteilte Infrastruktur

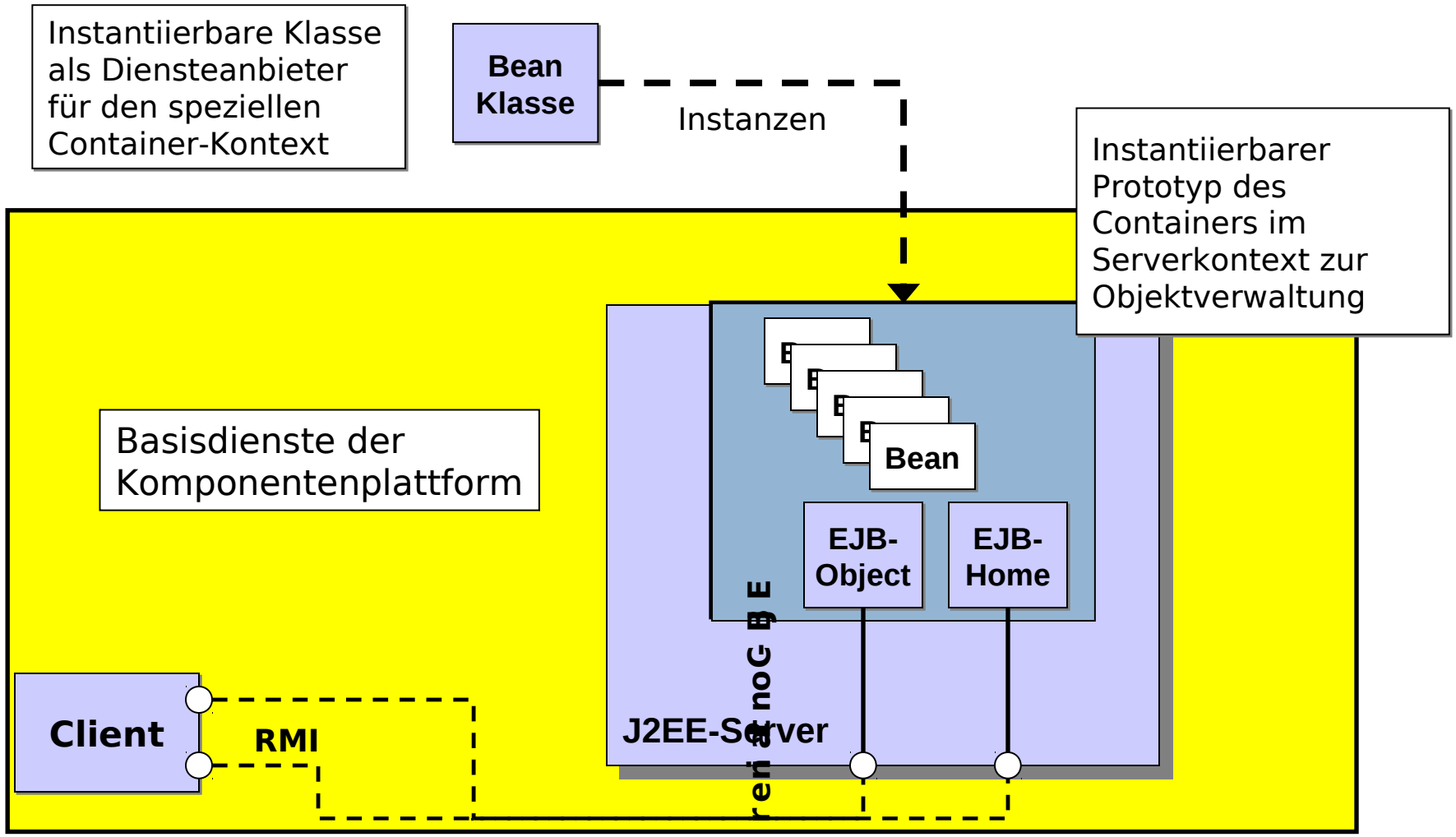
Anwendungsbereich: Middleware und verteilte Systeme

Grundlage: eng gekoppelte Client-Server-Architektur, gemeinsames Framework

Beispiele: CORBA, EJB, Spring, OSGi

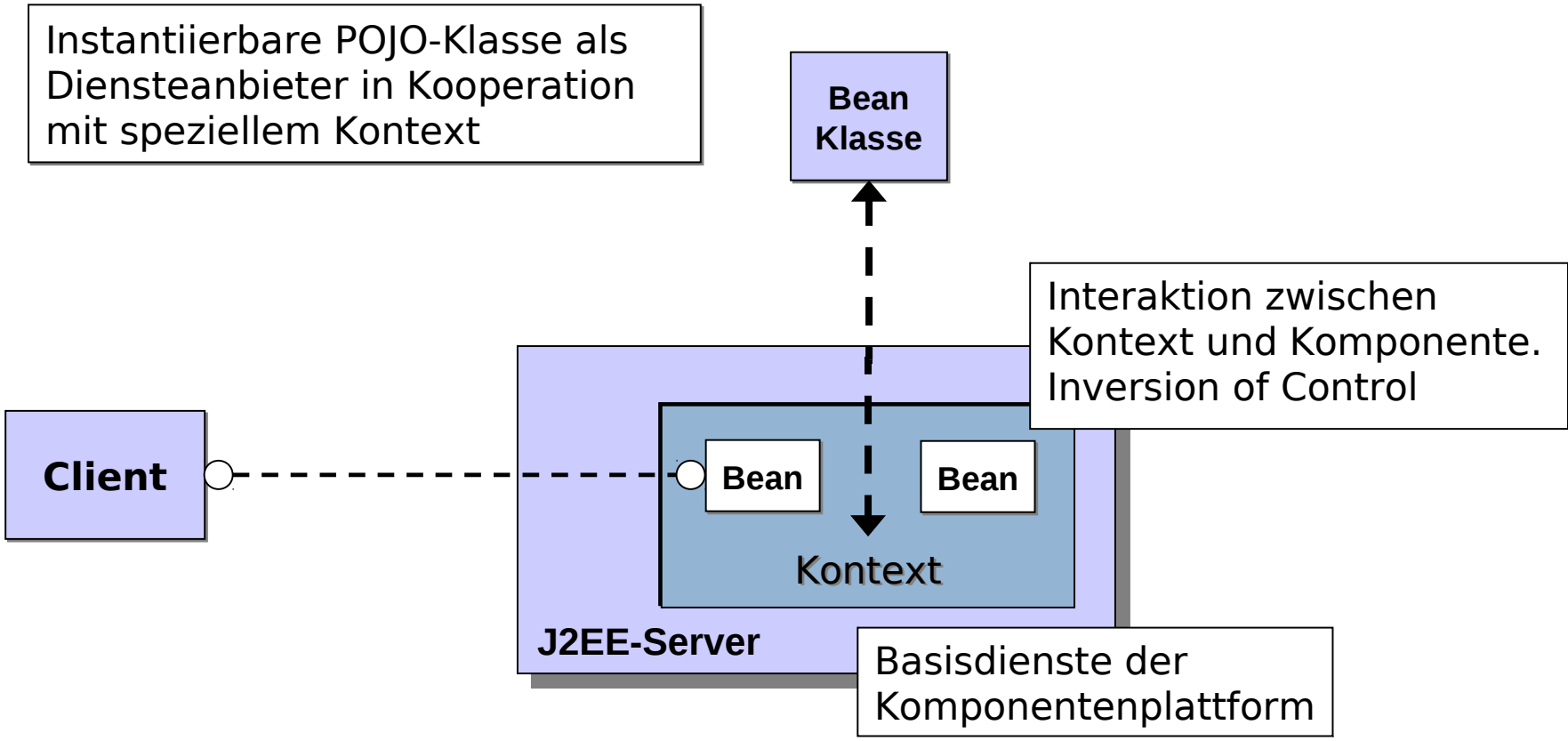
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Prototypischer Aufbau von CORBA und EJB 2



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Prototypischer Aufbau leichtgewichtiger Komponenten-Frameworks



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Konvergenz auf der Ebene der Konzepte

- Alle Zugänge unterstützen spätes Binden, Kapselung, dynamische Polymorphie, Vererbung auf Schnittstellenebene
- Standardisierte Komponenten-Transfer-Formats unterschiedlicher Leistungsfähigkeit
  - Java: *.jar, CLI: Assemblies
- Uniformer Datentransfer
  - einheitliche Konzepte der Serialisierung von Objekten
  - Entwicklung von Persistenzmechanismen auf dieser Basis
- Ereignis- und Ereigniskanal-Konzept
- Metainformationen können über Introspektion und Reflektion zur Laufzeit ausgelesen werden
- Einsatz von Konfigurationsinformationen
  - Montage-Beschreibungen
  - attributbasiertes Programmieren (CLI) – custom attributes
  - Annotationen, Übergang zu Beschreibungssprache für Abhängigkeiten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Verteilte Speicherverwaltung und Garbage Collection

- Komplizierte Aufgabe in Systemen mit verteilten Objekten
- explizites Management des Lebenszyklus: CORBA
- Referenzzähler-Konzept: COM/DCOM
  - verlangt Kooperation aller Komponenten
  - skaliert schlecht in offenen verteilten Umgebungen
- Object leasing = Objektreferenzen haben nur beschränkte Lebensdauer
  - Java: GC von Java-RMI mit sehr guter Performance in verteilter Umgebung.
  - CLR: verwendet ähnlichen Ansatz

Evolution und Versionsmanagement

Sehr wichtig, wenn man Software-Entwicklung als Prozess verstehen will. Wird aber bisher kaum unterstützt

Erster Ansatz: Schnittstellen und deren Spezifikation dürfen nach Veröffentlichung nicht verändert werden (immutable interfaces)

CORBA: Versionsnummern, die zur Objektinitialisierung geprüft werden

- Damit *dynamische* Versionsprüfung nicht möglich

Java: einige Regeln, aber inkonsistent

- Problem der vorübersetzten Konstanten bei Versionswechsel

CLI: Adressiert das Problem erstmals in voller Komplexität

- Jede Assembly trägt Versionsinformationen von sich und allen Import-Komponenten. Es kann festgelegt werden, welche Toleranzen der Versionen erlaubt sind.
- In einer Komponente können mehrere Versionen koexistieren
- Standard wird weder von .NET noch von der CLR voll unterstützt

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Kategorien

- erstmals von COM zur Klassifizierung von Software eingeführt
- Kategorie = Schnittstellenkontrakt auf Komponentenebene
  - Konkrete Komponente kann zu mehreren Kategorien gehören
  - Kategorie = abstrakte Zusicherung (high level assertion)
- Java, CORBA: kennen dieses Konzept nicht (aber: Marker-Interface)
- CLI: Unterstützung über Nutzerattribute

Montage / Konfigurierung

- EJB 2: Erstmals Abtrennung der Montage als eigenständiger Schritt und Beschreibung in einem deployment descriptor
- J2EE: erweitert dieses Konzept auf andere Komponentenmodelle
- .Net, EJB 3, Spring: Inversion of Control und aspektorientierte Programmierung
  - Trennung von Installations-Konfiguration und zur Laufzeit erforderlicher Kontextinformation
  - damit werden die Rollen des Komponentenentwicklers und des Komponentenmonteurs klarer unterschieden
- CLR: kennt sowohl XML-basierte Konfiguration als auch CLI-basierte custom attributes

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten und Objekte

Szyperski: „beyond object oriented programming.“

Grundprinzipien der Objektorientierung:

- Zerlegung des globalen Raums der Programmzustände in lokal abgrenzbare interagierende Einheiten (Objekte)
- Objekte kapseln Zustand und Verhalten
- Objekte sind Instanzen von Klassen und werden damit von einer überschaubaren Zahl prototypischer Generatoren erzeugt
- Diese Generatoren sind als Klassen durch das (traditionelle) Vererbungskonzept verbunden.

Weitere Fragen:

- Ortstransparenz der Objekte in verteilten Umgebungen
- Persistenz von Objektidentitäten

Java

- Alles ist aus Objekten (Ausnahme: ein paar primitive Typen)
- Fokus auf den Klassen und ihren Interaktionen, nicht den Interaktionen der Objekte. Damit steht das Verhalten im Vordergrund. Klassen als Kapselungseinheit, denn nur über diese können Instanzen verwaltet werden.
    - orthogonal dazu das Konzept der Pakete
- Ortstransparenz von Objekten in verteilten Umgebungen über eigenständige Konzepte wie RMI
- Persistenz von Objektidentitäten nur auf der Ebene von Basisdiensten, nicht per se.

CORBA

- Objekt als zentrales Konzept. Damit steht der Zustand im Vordergrund.
- Klasse = Objektimplementierung, hat aber nichts mit Vererbung zu tun. Abtrennung der Funktionalität in die Servanten, die aber über die Objekte aufzurufen ist.
- Kapselung durch Restriktion aller Interaktion auf Objektschnittstellen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

CORBA (Fortsetzung)

- CORBA-Objekte sind recht gewichtig
  - Unterschied zwischen lokalen Objektreferenzen (kennt nur POA) und CORBA-Objektreferenzen
  - keine Unterstützung „kleiner“ oder „serverloser“ Objekte
  - zu teuer für jegliche Kommunikation innerhalb einer Komponente
  - OMG IDL kennt nur CORBA-Referenzen

CLR

- Einheitliches Typsystem mit **Object** als Wurzel, das Wert- und Referenztypen vereint
  - Instanzen der Basistypen sind keine Objekte, können aber wie solche behandelt werden.
- Objekte als Klasseninstanzen
- einfache Implementations- und mehrfache Schnittstellenvererbung
- Persistenz von Objektidentitäten auf der Ebene von Diensten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zusammenfassung:

- Java und CLR kommen den klassischen OO-Prinzipien am nächsten
  - Ausnahme: Klassen, nicht Objekte als Kapselungseinheit
- COM und CORBA: Kapselungseinheit Objektserver, aber keine Konzepte der Interaktion von Objekten im selben Server
  - Objekte als Kapselung von Zustand und *potenziellem* Verhalten (Schnittstellendefinition)
  - Davon abgetrennt Servanten als Ort der Realisierung *tatsächlichen* Verhaltens
  - Damit näher an der Trennung Komponente – Objekt dieser Vorlesung
- Moderne Komponentenkonzepte:
  - Identifizierung des Kontexts als eigenständiger Quelle tatsächlichen Verhaltens (cross cutting concerns)
- Java, CLR: Unterscheiden zwischen internen und fernen Objektreferenzen. Letztere können nur über spezielle Infrastrukturdienste (Java RMI, CLR Remote) angesprochen und verwaltet werden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Kontraktspezifikation für Komponenten

- „Bessere Kontrakte für bessere Komponenten“ [Szyperski, 19.5]
- Anforderungen an die Kontraktspezifikation sind höher als bei klassischer Software aus folgenden Gründen:
  - technologische Aspekte sind komplexer
  - Qualitäts-, Haftungs- und Sicherheitsfragen bei der Nutzung von Komponenten „Dritter“
- Wird in heutigen Komponentenkonzepten so gut wie nicht angesprochen
- QS ist nur bei klarer Spezifikation überhaupt kommunizierbar
  - Schnittstellen-Listing mit informeller Beschreibung (etwa auf der Ebene von **javadoc**) von Funktionalität reicht dafür nicht aus.
  - Qualität wird heute meist de facto durch starke Anbieter gesichert; am besten auch gleich im Kontext von Anwendungen dieser Anbieter
    - Beispiel: OLE und Word, Excel, Internet Explorer

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Problem der Behinderung der Entstehung einer Komponentenwelt durch Unterspezifikation
  - Bsp. CORBA: vom BOA zum POA
  - Erfahrung kommt erst im praktischen Einsatz konkurrierender Implementierungen desselben Standards
  - Es geht um Konvergenz der Interpretation des Standards
    - ausgewogene Balance von Enge und Freiheit
- Schnittstellenkontrakt von Komponenten ist immer mehr als die Spezifikation des „Zusammenschaltens“
  - wird immer informelle Elemente enthalten, da es (auch) ein sozialer Kontrakt zwischen Entwicklern und Nutzern von Komponenten ist
  - klarer Link zwischen Schnittstelle (als „Kontrakt-Instanz“) und Kontraktspezifikation erforderlich
- Zu jeder Schnittstelle gehört eine solche Spezifikation
  - Verbindung von Schnittstellen-Syntax und Semantik (Bedeutung)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Konzept der Kategorien: Spezifikation nach dem Baukastenprinzip
  - Java: Marker-Schnittstellen, COM: Kategorien
  - CORBA: Repository ID‘s verbinden eindeutige ID mit OMG IDL Typen
  - CLR: Konzept der Assembly sowie Konfigurationsattribute (custom attributes) zur Fixierung von Metadaten-Informationen
- Das sind bisher aber alles rein deklarative Methoden
  - Muss weiter formalisiert und in den Komponenten-Lebenszyklus (Entwicklung, Test, Zertifizierung, Laufzeit-Monitoring, ...) integriert werden
  - Entwicklungsrichtungen:
    - ASML = Abstract State Machine Language
      u.a. Werkzeuge zur automatischen Generierung von Testorakeln und -fällen
    - TTCN = Test and Test Control Notation Language
      des ETSI (European Telecommunications Standards Institute)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentensoftware und die Grundlagen der Softwaretechnik

- Komponentenansatz enthält eine Reihe neuer Herausforderungen für einen modularen Ansatz auch in der Software-Technik
  - Schlüsselproblem: Ansatz der unabhängigen Erweiterbarkeit
  - späte Integration von Komponenten unabhängiger Hersteller
    - Konflikte mit Integrationstestkonzepten der klassischen SWT
  - Erweiterbarkeit muss selbst „designed“ werden, sonst passt nichts
  - Problem der verschiedenen methodischen Ansätze im SWE für interagierende Komponenten
  - Top-down-Design (ausgehend von der Anforderungsanalyse) trifft mit ziemlicher Sicherheit nicht die verfügbaren Komponenten
  - Bottom-up-Design ausgehend von Basisfunktionalitäten der verfügbaren Komponenten trifft mit ziemlicher Sicherheit nicht die Anforderungen
- Hier ist noch vieles unausgereift und ein umfassender Komponenteneinsatz nur aus strategischen Überlegungen heraus zu rechtfertigen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentenorientiertes Programmieren als Methodologie

- Wie OOP die Methodologie des Programmierens objektorientierter Lösungen ist, so ist COP die Methodologie des Programmierens von Komponenten

- Definition (Szyperski):

  Komponenten-orientiertes Programmieren bedeutet Unterstützung von
  - Polymorphie (Substituierbarkeit)
  - modulares Kapseln (Verstecken von Information)
  - spätes Binden und Laden (unabhängige Auslieferbarkeit)
  - Sicherheit (Typ- und Modulsicherheit)

- Bisherige Methodologien erstrecken sich nur auf die Entwicklung einzelner Komponenten
- Neuere Entwicklungen: Die Catalysis-Methode
  - http://www.catalysis.org

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten-Montage

Komponenten als Einheiten der Auslieferung durch Dritte und als Einheiten der Komposition

- Ein Weg zur Komposition ist traditionelle Programmierung
- Attraktivität von Komponenten nimmt zu, wenn einfachere Kompositions-Prinzipien verfügbar sind
  - visuelle Komposition in Grafik-Werkzeugen
  - zusammengesetzte Dokumente
  - Zusammenbinden durch Skripting
  - Zusammenbinden als Web Services
- besonders attraktiv, wenn der Endnutzer diese Montage selbst vornehmen kann (IKEA-Prinzip)

Alle diese vereinfachten Montage-Prinzipien setzen auf inhaltlicher Seite kontextuelle Kapselung und Komposition der Komponenten voraus

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Infrastruktur-Aufwand für Komponentenanbieter

- OMA: Jeder ORB-Anbieter muss seine Sprachanbindung zu allen unterstützten Sprachen herstellen
- Java: Überall lauffähig, wo eine JVM läuft
  - ein Classfile-Compiler pro unterstützter Sprache ist erforderlich
  - es gibt solche Compiler für viele gebräuchliche Sprachen
  - JVM-Standard ist allerdings für die Verwendung mit Java optimiert
- CLI: verfolgt ähnliche Strategie wie Java, zielt aber auf eine breitere Unterstützung von anderen Sprachen
  - braucht so was wie die JVM auf allen unterstützten Plattformen
    - CLR als Implementierung auf .NET (Windows, Microsoft)
    - Open-Source-Projekte Mono und Open CLI Library Project
    - FreeBSD-Version von Corel und Microsoft

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Folgerung: Komponentenkonzepte müssen in eine (technische) Infrastruktur eingebettet sein.

Eine Lehre aus CORBA:

Wenn zu viele Dimensionen von Freiheit gekoppelt werden, um eine möglichst große Variation von Lösungen zu ermöglichen, dann werden die meisten praktischen Lösungen nur für Marktnischen relevant sein.

CORBA versagt bei einem seiner zentralen Versprechen: eine breite Varietät nicht nur von möglichen, sondern von realen Lösungen zu unterstützen. Es fehlen dafür strenge low-level Integrationsstandards.

Die Maximierung der Zahl der kombinatorisch möglichen Variationen minimiert die Zahl der real verfügbaren Varianten.

Für ein Komponentenmarkt ist die Freiheit der Inhalte ebenso entscheidend wie die Beschränktheit der Design-Konzepte.

Diese Standardisierungsbemühungen stehen noch ganz am Anfang.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten und Berufsprofile für Informatiker

Komponenten-Systemarchitekt

- Komponenten funktionieren nur innerhalb eines Frameworks (konkrete Implementierung eines Architektur-Konzepts)
- Konsistentes Architekturkonzept deckt mehrere Frameworks und deren Interoperabilität ab
- Entwickelt die Architektur für die Architekten - der einzelnen Frameworks

Komponenten-Frameworkarchitekt

- Entwicklung von Konzepten und Werkzeugen, um konkrete Komponenten in eine Infrastruktur „einzustöpseln“
- Muss sich im gesamten Anwendungsbereich des Frameworks gut auskennen
- Implementierung des Frameworks ist die Basis für eine funktionierende Komponentenwelt
- Muss Anforderungen an die Komponenten-Entwickler spezifizieren

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten-Entwickler

- Komponenten-Entwickler erstellen die „Blätter“ für das Komponenten-Framework
- Die funktionalen Spezialisten in dieser arbeitsteiligen Struktur mit Spezialkenntnissen aus den konkreten Anwendungsbereichen, die von den zu entwickelnden Komponenten abgedeckt werden

Komponenten-Monteur

- Aufgabe: Anpassen, „Zuschneiden“ und Integrieren von Komponenten für den konkreten Gebrauch in speziellen Anwendersystemen
- Auflösung des klassischen Begriffs der „Anwendung“ als monolithisches und statisches System zugunsten des Konzepts einer organischen (und organisch wachsenden) IT-Infrastruktur
- End-Nutzer übernehmen in einem solchen Konzept zunehmend eine eigenständige Rolle, die vom Komponenten-Monteur abzugrenzen ist
- Aspekte der Nutzerschulung treten dann ergänzend hinzu
- Feedback zu Komponenten-Entwicklern und Framework-Architekten