Vorlesung
Software aus Komponenten

3. Komponenten-Modelle

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

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 (J2EE, CORBA, CLR, OSGi)
  - Laufzeitsystem als Infrastruktur, in der 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

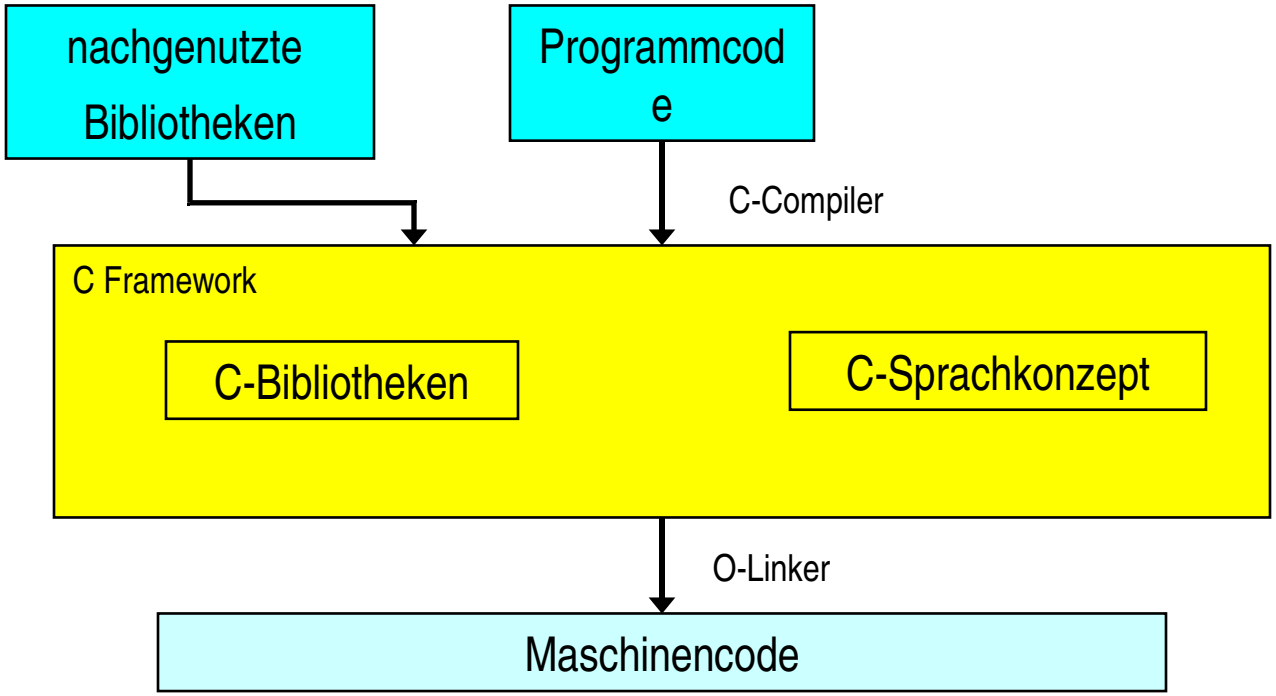
Anwendungsbereich: Desktop, Basiskomponenten

Grundlage: Gemeinsame Designprinzipien

Beispiele: C, Java, .NET

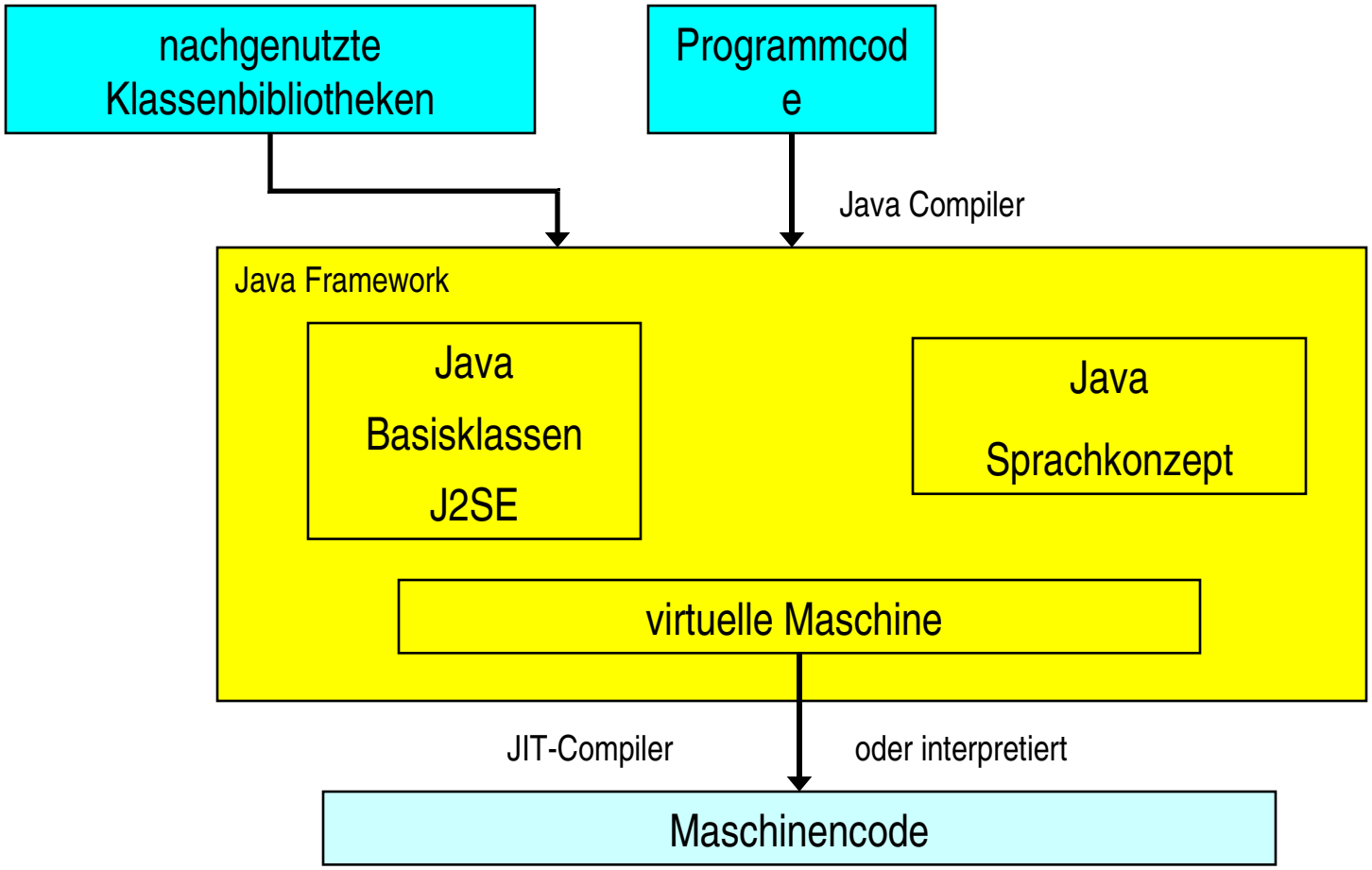
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Eine Sprache, eine Plattform: C



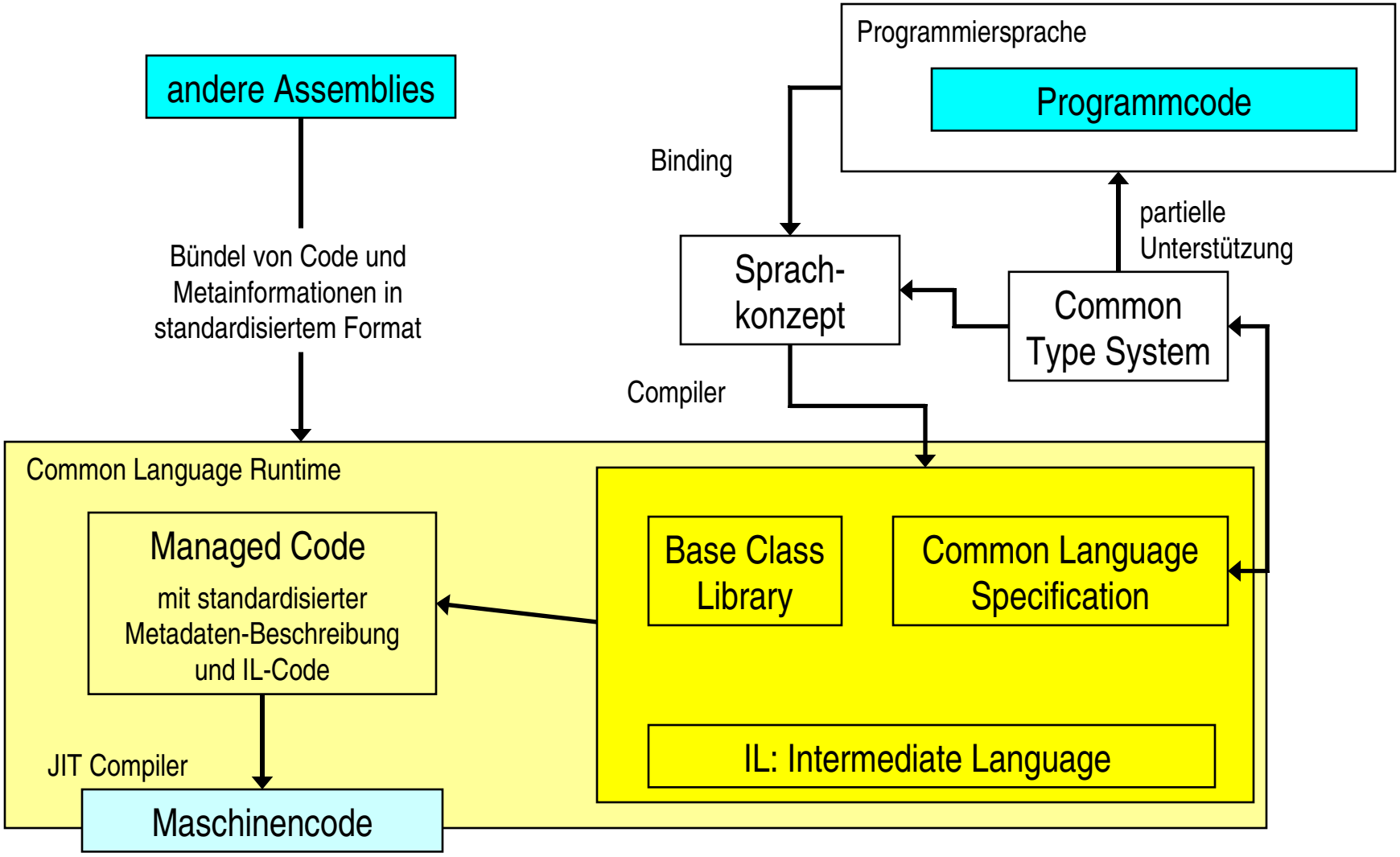
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Eine Sprache, mehrere Plattformen: Java



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Mehrere Sprachen, mehrere Plattformen: .NET



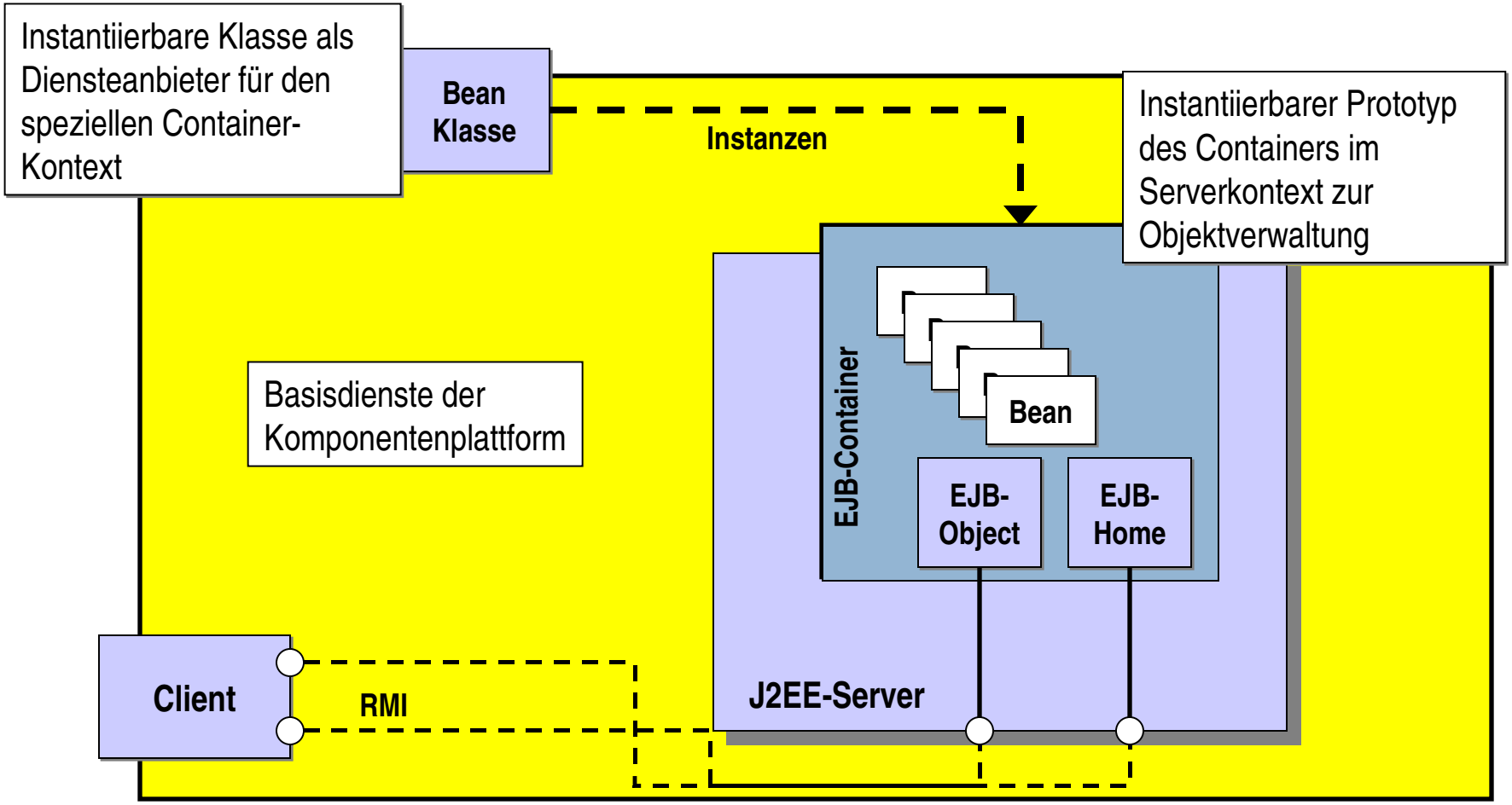
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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: EJB, Servlets, CORBA,

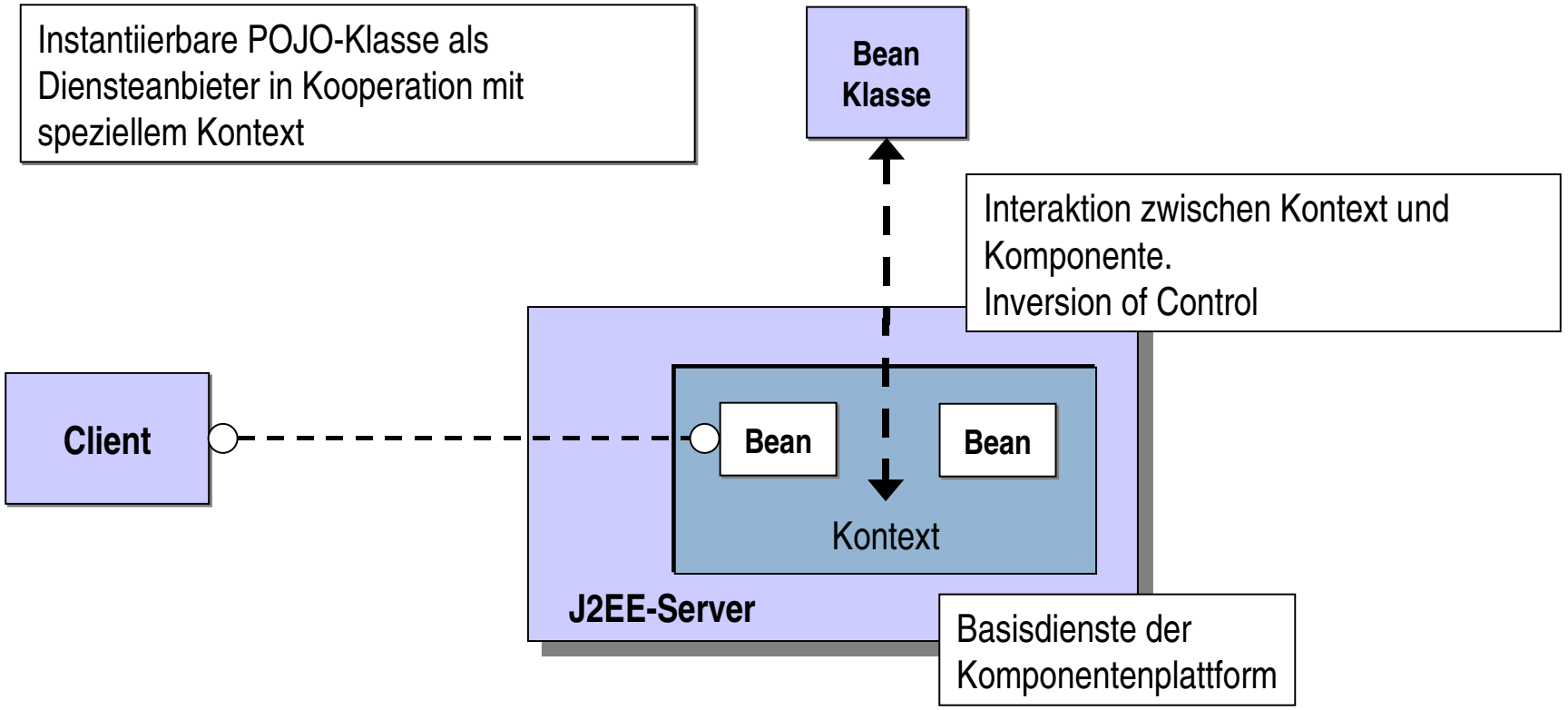
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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
  - Java: *.jar, COM: *.cab, CLI: Assemblies
- Uniformer Datentransfer
  - einheitliche Konzepte der Serialisierung von Objekten
  - Entwicklung von Persistenzmechanismen auf dieser Basis
- Ereignis- und Ereigniskanal-Konzept
- Metainformationen über Introspektion und Reflektion
  - erlaubt dynamische Erweiterung von Schnittstellen
- Einsatz von Konfigurationsinformationen
  - Montage-Beschreibungen
  - attributbasiertes Programmieren (CLI) – custom attributes
- dynamische Konfiguration
  - COM: QueryInterface, CORBA: EquivalenceInterface

Unterstützung von Austausch auf Binärstandardebene

- COM: grundlegendes Ziel, wenn auch weitgehend ohne Bedeutung außerhalb der Windows-Welt
- Java: Binärstandard an JVM über JNI gebunden
- CORBA: Nur im Rahmen von CORBA-to-* Compilern. Nicht plattformübergreifend standardisiert
- CLR: Ähnlich Java eine Ebene über Binärstandards angesiedelt, aber direkte und JIT-Compilation vorgesehen

Quellcodestandards für Kompatibilität und Portabilität

Wie kann Quellcode in verschiedenen Sprachen eingebunden werden?

- CORBA: spezielle Sprachbindungen IDL-to-*
  - Problem: Verwendung ORB-spezifischer Funktionen auf der Serverseite ist weit verbreitet. Damit nicht außerhalb einer schwergewichtigen Plattform lauffähig
- Java: So lange alles in Java geschrieben ist – kein Problem
  - direkte Übersetzung aus anderen Sprachen in Java Bytecode ist möglich (wird etwa bei J2EE-Implementierungen verwendet)
- COM: keine Standards jenseits der Microsoft de-facto Standards
- CLR und .NET: Interoperabilitätskonzept durch Sprachbindungsstandards

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

Containerverwaltetes Persistenz-Management

- Mit EJB eingeführt und mit EJB 2.0 auch auf Relationen ausgeweitet
  - sehr datenbankspezifisch, außerhalb dieses Einsatzgebiets nicht sehr performant
- OLE-Datenbanken: Konzept der Persistenz-Abbildung (pluggable persistence mapping) erlaubt Abbildung auf verschiedene externe Speichermedien

Evolution und Versionsmanagement

- Sehr wichtig, wenn man Software-Entwicklung als Prozess verstehen will. Wird aber bisher kaum unterstützt
- COM: Schnittstellen und deren Spezifikation dürfen nach Veröffentlichung nicht verändert werden (immutable)
  - Aber: Möglichkeit der dynamischen Erweiterung
- CORBA: Versionsnummern, die zur Objektinitialisierung geprüft werden
  - Aber: 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

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: Attribute werden in der Montage-Beschreibung verwaltet
  - Erstmals Faktorisierung des Montage-Schritts
- J2EE: erweitert dieses Konzept auf andere Komponentenmodelle
- .Net und EJB 3: Inversion of Control und attributgesteuerte 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