Vorlesung
Software aus Komponenten

3. Komponenten-Modelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2006/07

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Enterprise JavaBeans

- **Modell**: Beans = ununterscheidbare Objekte in einem Container
- Container-Abstraktion repräsentiert die spezielle Art, in welcher Beans an Ressourcen gebunden sind.
- kein direkter Zugriff auf Attribute und Methoden von Beans, auch nicht innerhalb desselben Containers
  - Verhältnis wie zwischen DB-Server und Datensätzen
  - Zugriff nur über zwei Schnittstellen, die **javax.ejb.EJBHome** (für Container) und **javax.ejb.EJBObject** (für Beans) erweitern
  - EJBHome: Methoden zum Management des Lebenszyklus der Beans
  - EJBObject: Methoden der Bean-Instanzen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Schnittstelle EJBObject

- interface **MyEJBObject extends javax.ejb.EJBObject**
- Aufruf-Schnittstelle, über welche ein Client auf die Dienstleistung zugreifen kann
- **Beschreibt** Dienstleistung
  - Darstellung von Attributen über get/set-Methoden
- Nichtlokale Clients rufen Schnittstelle auf Stub-Objekt, welches über RMI oder RMI-über IIOP mit dem EJB Container kommuniziert
  - deklarierte Operationen müssen Exception **java.rmi.RemoteException** auslösen können
  - Abfangen von Kommunikationsproblemen im Netz
- Lokale Clients können über lokale Version der Schnittstelle (Erweiterung von **EJBLocalObject**) zugreifen, wenn von der e-bean bereitgestellt

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Container-Schnittstelle EJBHome

- interface **MyEJBHome extends java.ejb.EJBHome**
- Verwaltungs-Schnittstelle: verwaltet Bean-Instanzen im Container
  - Erzeugen neuer Instanzen
  - Auffinden vorhandener Instanzen
  - serialisiert alle Zugriffe auf seine Beans
- Operationen **create** und **findByPrimaryKey** (entity beans)
- optional weitere Methoden, die nicht mit einzelnen Beans zu assoziieren sind (analog zu statischen Methoden einer Klasse)
- begleitet Lebenszyklus seiner Beans
  - **setEntityContext** oder **setSessionContext** erzeugt Rückverweis auf den Container-Kontext
  - **ejbCreate** / **ejbRemove**
    - Ressourcenallokation bzw. –freigabe
  - **ejbPassivate** / **ejbActivate**
    - zeitweise Trennung von den Ressourcen und Speichern in serialisierter Form auf externem Medium

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Klassen

- public class MyBeanClass implements javax.ejb.xxBean
- es gibt **SessionBeans**, **EntityBeans** und **MessageDrivenBeans** als Subklassen von **EnterpriseBeans**
- **Implementierung** der zur Verfügung gestellten Operationen
  - also eigentlich auch **... implements MyEJBObject**
  - wird aber nur informell überwacht und diese Verbindung erst bei der Installation hergestellt
  - Entwickler muss auf Übereinstimmung der Signaturen selbst achten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Klassen

Es gibt vier Sorten von Enterprise JavaBeans

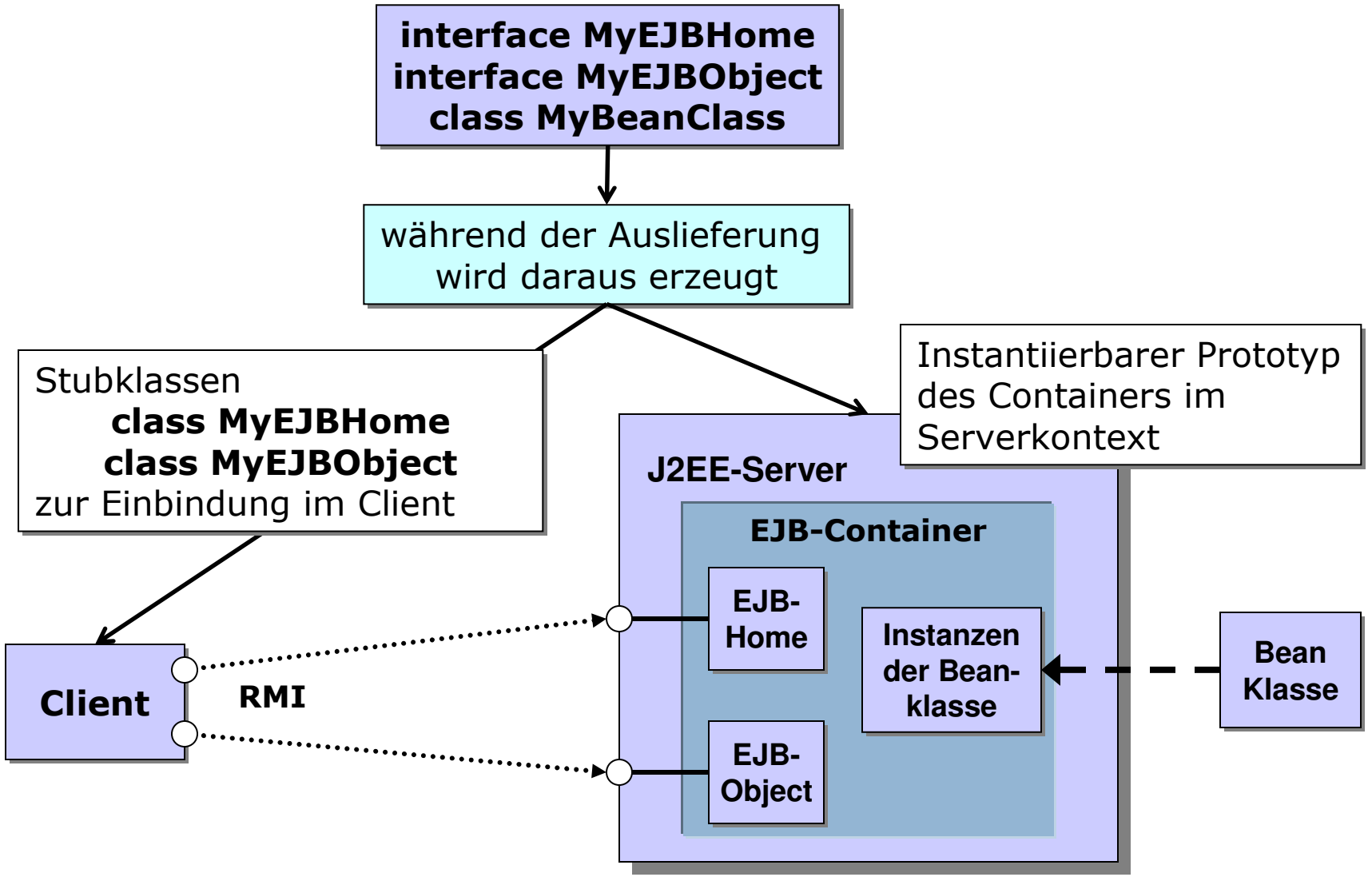
- **zustandslose** (stateless) und **zustandsbehaftete** (stateful) **Session Beans**
  - implementieren **javax.ejb.SessionBean**
  - entsprechen einer Session in der Datenbankterminologie
  - beide Arten sind transient, also nur innerhalb einer Session gültig
  - Zustand wird zwischen verschiedenen Methodenaufrufen gespeichert/nicht gespeichert

- **Entity Beans**
  - implementiert **javax.ejb.EntityBean**
  - enthalten persistente Daten, entsprechen dem Zugriff auf einen konkreten Datensatz in einer Datenbank
    - bzw. eher einem Datensatz in einem Join
    - Persistenz kann Bean-gesteuert (etwa JDBC-Implementierung) oder Container-gesteuert sein

- Zugriff wie bei Datensätzen über Primärschlüssel
  - ggf. muss eine Bean erzeugt und an den Datensatz mit diesem Schlüssel gebunden werden
  - Wert aus validem Java-Typ (auch komplexer Natur)
- Anfragesprache EJB QL ähnlich SQL

- **nachrichtengesteuerte** (message-driven) **Beans** (neu in EJB 2.0)
  - gebunden an Container, transient, aber ohne Schnittstelle
  - Bean wird einer Nachrichtenschlange zugeordnet und, wenn sie „dran" ist, ihre zentrale Methode onMessage abgearbeitet.
  - Schlange muss dem Java Messaging Standard (JMS) genügen
  - damit kann voll asynchrones Kompositionsmodell entworfen werden
    - sinnvoll etwa für workflow-orientierte automatische Systeme

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Auslieferungs-Beschreibung (Deployment-Deskriptor)

- Beschreibungsdatei im XML-Format

  Beispiel Session-Bean

```
<session>
    <ejb-name> Name der Session-Bean </ejb-name>
    <home> Name der EJBHome Schnittstelle </home>
    <remote> Name der EJBObject Schnittstelle </remote>
    <local-home> Name der EJBLocalHome Schnittstelle </local-home>
    <local> Name der EJBLocalObject Schnittstelle </local>
    <ejb-class> Name der Bean-Klasse </ejb-class>
    <session-type> stateless | stateful </session-type>
    <transaction-type> Container (default) | Bean </transaction-type>
    <ejb-ref> Importdeklaration anderer Beans </ejb-ref>
    <security-identity> eigene oder die des Aufrufers </security-identity>
</session>
```

- enthält Informationen zur Installation:
  - Schnittstellen, Attribute, Operationen
  - Rollen für Benutzer
  - Rechte dieser Rollen
  - Transaktionsverhalten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Anforderungen an den EJB–Container-Kontext

- JVM wird benötigt, ist aber allein nicht ausreichend
- EJB-Standard spezifiziert Schnittstelle und Verhalten von Container und J2EE
- Container benötigt zur Verwaltung seiner Beans
  - Namensdienst (Java Name and Directory Interface)
  - Transaktionsmonitor (Java Transaction API)
  - Datenbankzugriff (Java Data Base Connectivity)
  - eMail (Java Mail API)
  - Standard Java API
- Greifen dabei gewöhnlich auf weitere Dienste im Rahmen des J2EE Applikationsservers zu

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Schnittstelle

```
package SemOrg.Schnittstellen;
import java.rmi.*;
import javax.ejb.*;

public interface Buchung extends EJBObject {
    public void buchen(Kunde k, Seminartyp s) throws RemoteException;
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Container-Schnittstelle

```
package SemOrg.Schnittstellen;
import java.rmi.*;
import javax.ejb.*;

public interface BuchungHome extends EJBHome {
    public Buchung create() throws RemoteException, CreateException;
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Klasse

```
package SemOrg.Server;
import SemOrg.Schnittstellen.*;
import java.rmi.*;
import javax.ejb.*;

public class BuchungBean implements SessionBean{

    // Konstruktor
    public BuchungBean() {}

    //Operationen der Remote-Schnittstelle Buchung
    public void buchen(Kunde k, Seminartyp s) throws RemoteException
        { //Code zur Ausführung der Buchung }

    // Implementierung der Schnittstelle SessionBean
    public void ejbRemove() {}
    public void ejbActivate() {} // etc.
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Deployment-Deskriptor

```
<!-- Deployment descriptor -->
<enterprise-beans>
  <session>
    <ejb-name>SemOrg/Buchung</ejb-name>
    <home>SemOrg.Schnittstellen.BuchungHome</home>
    <remote>SemOrg.Schnittstellen.Buchung</remote>
    <ejb-class>SemOrg.Server.BuchungBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
  </session>
</enterprise-beans>
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Ein einfacher Client

```
package SemOrg.Client;
import java.rmi.*;
import javax.naming.*;
import SemOrg.Schnittstellen.*;

public class Client {
   public static void main(String[] args) {
      Client einClient=new Client();
      einClient.run();
   }

  public void run() {
    Kunde einKunde=getKunde();
    Seminar einSeminar=getSeminar();
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

```
// Fortsetzung Methode run
try {
  // Namenskontext des Servers finden
  Context ctx = new InitialContext();
  Object temp = ctx.lookup("java:comp/env/Buchung");

  // EJB-Container-Objekt vom Typ BuchungHome auf dem Server
  // finden und instantiieren
  BuchungHome home=
    (BuchungHome) PortableRemoteObject.narrow(
            temp, BuchungHome.class);

  // Bean anfordern und Buchung auslösen
  Buchung bean = home.create();
  bean.buchen(einKunde, einSeminartyp);
 }
 catch(Exception e) { }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Server-Provider (Server-Anbieter)

- stellt Plattform zur Verfügung
- Netzwerkanbindung
- Skalierungsfunktion
- Prozess- und Ressourcenmanagement

Container-Provider (Container-Anbieter)

- setzt auf Plattform des Server-Providers auf
- Benutzerverwaltung
- Transaktionsmanagement
- Persistenz
- Installationswerkzeuge
- Implementation der in der EJB-Spezifikation definierten Schnittstellen
- Container- und Server-Provider oft identisch
  - bessere Performance und Wartbarkeit

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Provider (Komponenten-Entwickler)

- realisiert geforderte Anwendungslogik (Geschäftslogik)
- keine grundlegenden Funktionalitäten (Persistenz, Netzwerk, etc.)
- benötigt Fachwissen über Anwendungsbereich
- Konzentration auf inhaltliche Problemstellung

Application-Assembler (Monteur)

- verbindet Komponenten zu einer Anwendung
- Clients
- Verwendung von Basiskomponenten
- Nutzung wiederverwendbarer Komponenten von Drittanbietern
- oftmals Bean-Provider und Application-Assembler in einem Haus

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Deployer (Installation)

- installiert Komponenten der Anwendung auf Zielsystem
- verwendet Werkzeuge des Container-Providers
- nutzt Deployment-Deskriptor
- konfiguriert EJBs
- generiert Stummel- und Skelettklassen
- legt Benutzerdatenbank an
- benötigt detailliertes Wissen über Server und Container

Systemadministrator

- verantwortlich für reibungslosen Ablauf
- Benutzerverwaltung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java-Basisdienste für Komponenten

Grundlegende Dienste

Java core reflection erlaubt zur Laufzeit

- Inspektion von Klassen und Schnittstellen nach Attributen und Methoden
- Konstruktion neuer Klasseninstanzen und Felder
- Zugriff und Modifikation von Attributen in Verbundobjekten oder Feldern
- Aufruf von Methoden von Objekten und Klassen

- **java.lang.reflect** als eigene Klasse hierfür
  - einige Funktionalität historisch in der (finalen) Klasse **java.lang**.

Java GuI-Klassensammlungen AWT und Swing

- delegierende Ereignisbehandlung
- Datentransfer und Zwischenablage wird unterstützt, drag and drop
- Java 2D rendering, eng damit zusammen Java printing model
- Internationalisierung
- pluggable look and feel, Palette von Standard-Komponenten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Objektserialisierung

- standardisiertes Kodierungsschema für Serialisierung
- Klasse muss dafür Interface **java.io.Serializable** implementieren
- Objektserialisierung ist sicherheitskritisch
- Mechanismen zu Serialisierung und Deserialisierung ganzer Objekt-Webs
  - nicht zu serialisierende Attribute können als transient markiert werden
    - Bsp: große Cache-Strukturen
  - private Methoden **readObject** und **writeObject** werden statt Default genommen, wenn durch Reflektion gefunden
  - Mehrfachreferenzen auf ein Objekt werden rekonstruiert
- unterstützt einfaches Versionierungsschema:
  - 64-bit-hash-Code (Serial version UID = SUID) wird über die Signatur der Klasse berechnet und kann während **readObject** ausgewertet werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Ferne Objekte und RMI

- Auf ferne Objekte kann nie direkt zugegriffen werden, sondern nur über Interface **java.rmi.Remote**. Ressourcenbindung über Namensdienst.
- Remotezugriff kann immer fehlschlagen:
  Exception **java.rmi.RemoteException**
- Parameterübergabe:
  - Referenz, wenn Parameterwert selbst vom Remote-Typ ist
  - Durch Marshalling übergebene Kopie, wenn Parameterwert lokal im aufrufenden Kontext
  - Übergabe nicht serialisierbarer Objekte führt zu Laufzeitausnahme
- Unterstützt verteiltes Garbage Collection
  - durch genaue Buchführung über Remote-Referenzen
  - basiert auf Arbeit [Birrel 1993] über Network Objects
  - eines der bedeutendsten Features von Java RMI
- Konflikt mit Begriff der Objektidentität im Java-Standard
  - Referenzen auf ein fernes Objekt sind Java Referenzen auf das lokale Proxy des fernen Objekts
  - Backcall erzeugt ein lokales Proxy im ObjectHome, neben der eigentlichen Java-Referenz

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Weitere Java-Dienste für Komponenten

JNDI: Java Naming and Directory Service

Aufgabe: Dienste über Namen bzw. Attribute finden

- Interface **Context** macht Namenskontext verfügbar
- Methode **lookup** findet Objekte über ihren Namen
- Interface **DirContext** erweitert **Context** zur Suche über Attributwerte
- Unterstützung von Kontexthierarchien, die rekursiv durchsucht werden.

JMS: Java Messaging Service

Aufgabe: Unterstützung asynchroner datengetriebener Kompositionsmodelle

- Standardisiert Java-Zugriff auf vorhandenes Nachrichtensystem, implementiert keins selbst.

JDBC: Java database connectivity

Aufgabe: Einheitlicher Zugriff auf Datenbanken über entsprechende Treiber

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

JTA: Java Transaction API

JTS: Java Transaction Service

Aufgabe: Unterstützung von Transaktionskonzepten

JCA: J2EE Connector Architecture (seit J2EE 1.3)

- Einheitliches Konzept des Ressourcen-Adapters, über welches externe Ressourcen aus einer EIS (enterprise information structure) in einen J2EE-Applikationsserver eingebunden werden können
- Definition eines entsprechenden **JCA common client interface** (CCI)
- Einsatz innerhalb von Enterprise Application Integration Frameworks

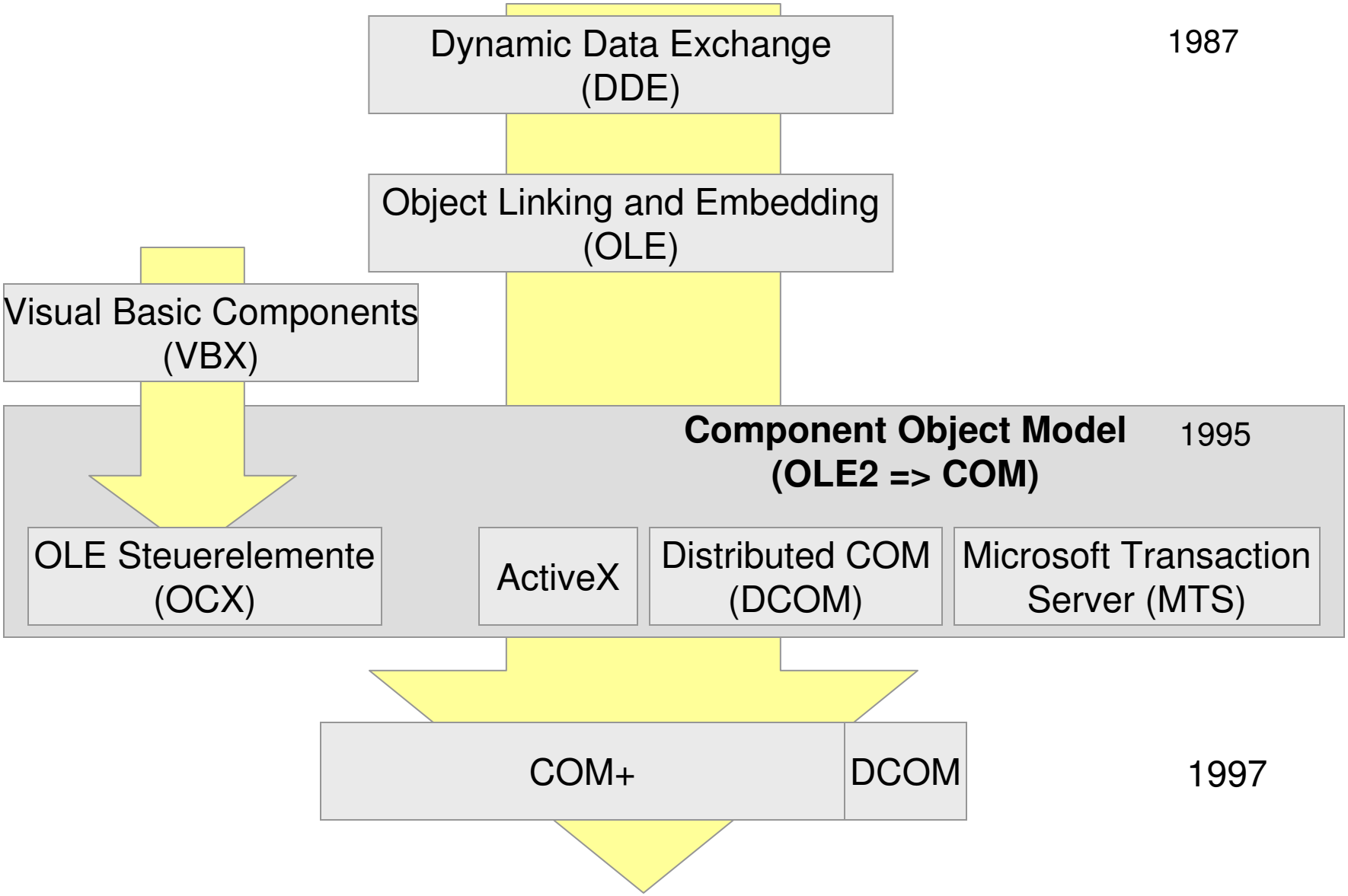
Java und XML: Java unterstützt mit entsprechenden Klassen

- XML-Dokumente (DOM)
- XML-Streaming (SAX)
- XML-Binding (JAXP)
- XML-Messaging (JAXM)
- XML-Processing (JAXP)
- XML-Registries (JAXR)

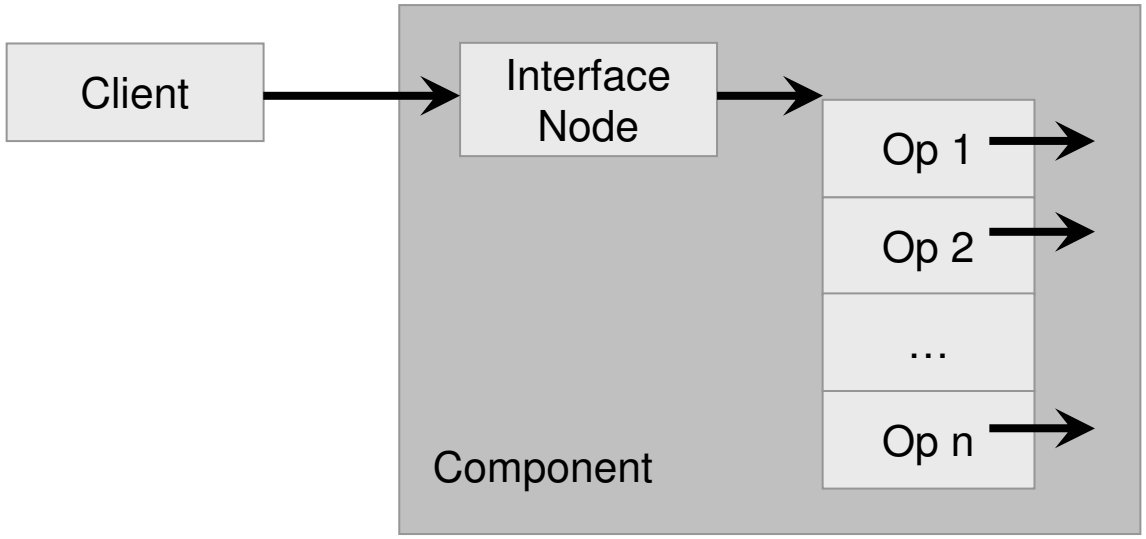
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java und CORBA:

- Java wichtigste CORBA-Referenzimplementierung
- Koexistenz in fast allen Applikationsserver-Produkten
- Zugriff auf CORBAservices über Java-spezifische Zugriffs-Schnittstellen sowie weitere Konzepte (POA, Namensdienst) seit Java 1.4
- RMI-over-IIOP als eingeschränkte RMI-Version seit 1999
  - RMI nutzt spezifisches proprietäres Protokoll
  - keine Unterstützung des verteilten Garbage Collection, so dass explizites Lebenszyklus-Management erforderlich ist
    - dafür existieren aber CORBAservices

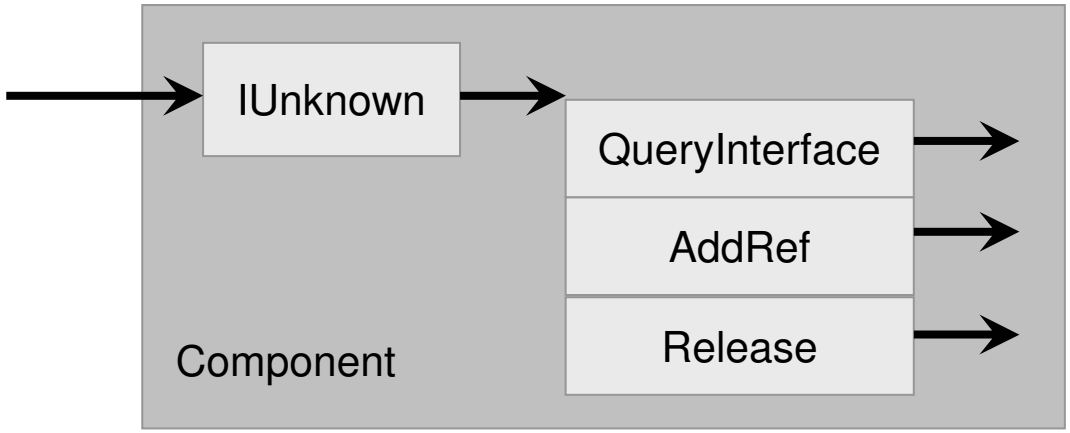

Dynamic Data Exchange (DDE)
1987
Object Linking and Embedding (OLE)
Visual Basic Components (VBX)
Component Object Model (OLE2 => COM)
1995
OLE Steuerelemente (OCX)
ActiveX
Distributed COM (DCOM)
Microsoft Transaction Server (MTS)
COM+
DCOM
1997

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



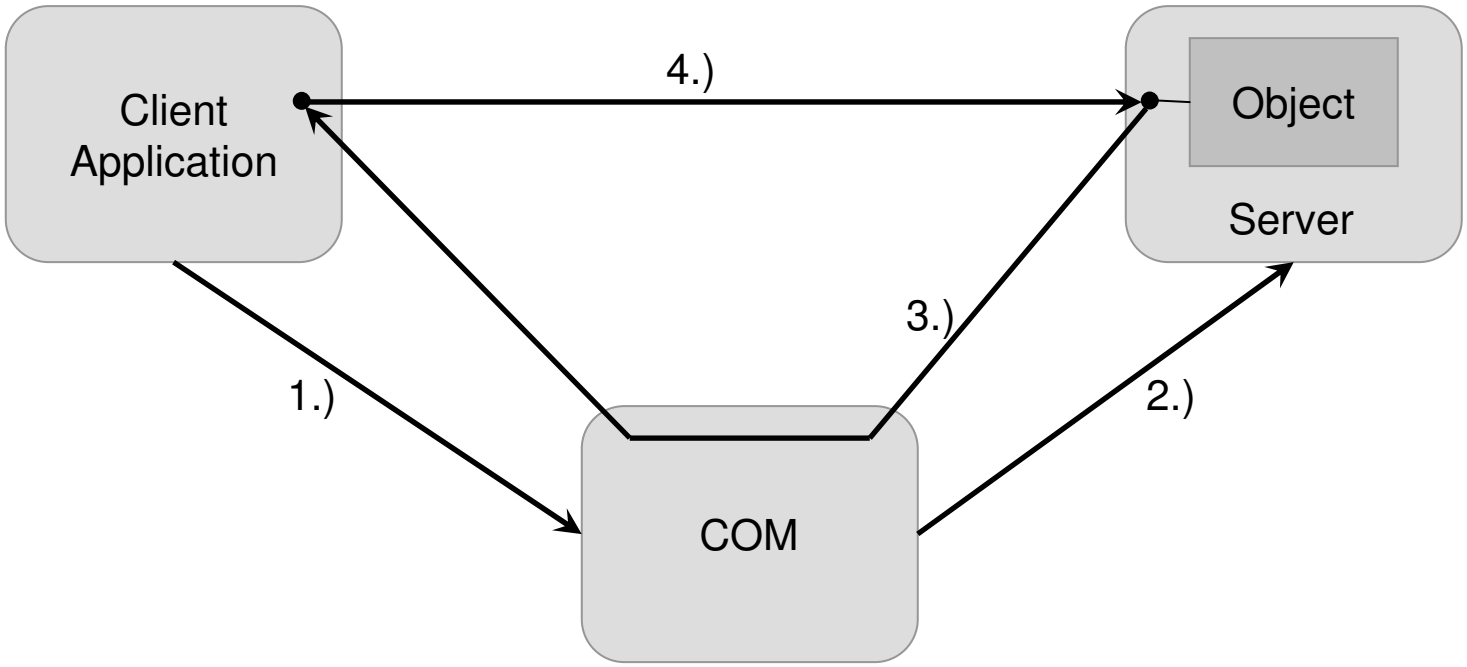
- COM ist binärer Standard
- Schnittstelle ist zentraler Punkt von COM
- Schnittstellenvererbung
- Interface Node
  - Methoden eines Objekts => **"this"**-Parameter wird an jede Methode weitergereicht
  - Komponente kann mehrere Schnittstellen implementieren

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

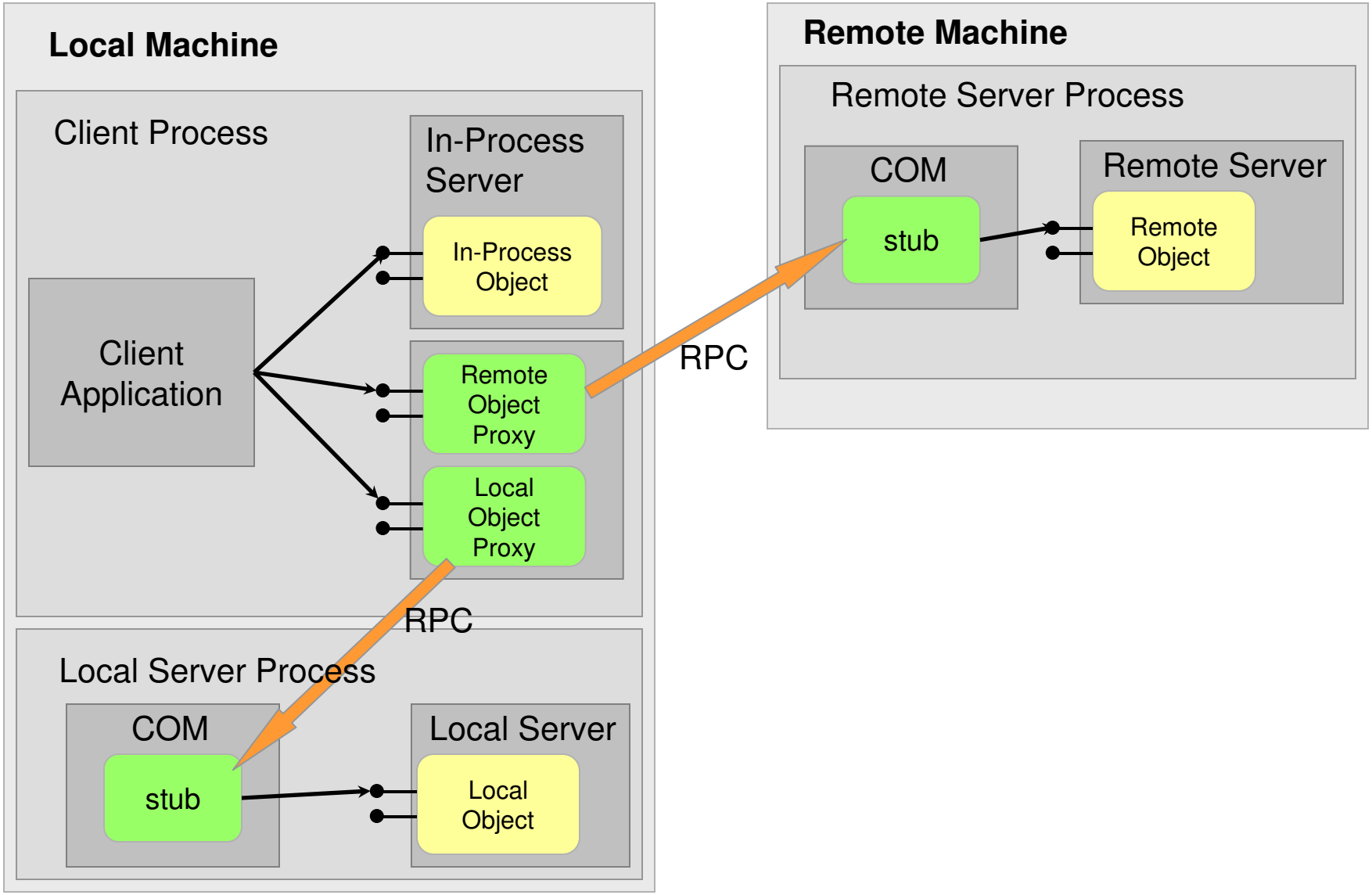


- Schnittstelle IUnknown **muss** von jeder Komponente implementiert werden
  - jede andere Schnittstelle erbt von IUnknown
- QueryInterface
  - erste Methode jedes COM-Objektes
  - gibt Zeiger auf angefordertes Interface zurück
  - benutzt Interface Identifier (IID)
  - erlaubt (dynamische) Introspektion der Schnittstelle
- AddRef, Release
  - Verwaltung des Lebenszyklus
  - Referenzzähler

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



1. Aufruf der Funktion "CreateObject"
2. COM lokalisiert/erzeugt den Server
3. Serverprozess erzeugt das Objekt und gibt einen Schnittstellenzeiger zurück
4. Client kommuniziert über den Schnittstellenzeiger mit dem Serverobjekt


Local Machine
Client Process
In-Process Server
In-Process Object
Client Application
Remote Object Proxy
Local Object Proxy
RPC
Local Server Process
COM
stub
Local Server
Local Object
RPC
Remote Machine
Remote Server Process
COM
stub
Remote Server
Remote Object

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Typinformationen (Type Information)
  - Laufzeitzugriff auf Typinformationen des COM Objektes
  - wird vom Microsoft IDL Compiler generiert und in einer Typbibliothek gespeichert
  - COM-Schnittstellen zum Interagieren mit dieser Bibliothek

- Strukturiertes Speichermodell (Structured Storage and Persistence)
  - von COM unterstützte Methode zum Speichern von Daten
  - Speicherung erfolgt analog des Dateisystems in einer Datei

- Moniker
  - Dienst, welcher ein einzelnes Objekt in einem genau definierten Zustand erzeugen und initialisieren kann
  - für Clients, die mit exakt dem gleichen Objekt weiterarbeiten müssen
  - Moniker kann an gesamtes Objekt oder an einen Teil gebunden werden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Einheitlicher Datenaustausch (Uniform Data Transfer)
  - Datentransfer zwischen COM-Objekten
  - Benachrichtigung von Datenänderungen einer Quelle (Datenobjekt) und einem Datenkonsumenten
- Verbindbare Objekte (Connectable Objects)
  - zur Ereignisverarbeitung
  - Objekt definiert ein Interface, welches für das Ereignis genutzt werden soll
  - Client implementiert dieses Interface
- COM+ Ereignisdienst (Event Service)
  - Asynchrone Kommunikation zwischen Komponenten
  - Empfänger abonniert Ereignis beim Dienst
  - Sender schickt Ereignis zum Dienst, ohne den/die Empfäner zu kennen
- COM+ Nachrichtenwarteschlange (Message Queuing)
  - Garantierte Auslieferung auch bei Nichterreichbarkeit des Empfängers
  - Für Systeme, deren Verfügbarkeit nicht immer gewährleistet ist