Vorlesung
Software aus Komponenten

3. Komponenten-Modelle

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Wichtige von Java unterstützte Grundkonzepte

- **Methoden** (Verhalten, behavior) und **Attribute** (Status, state)
- **Schnittstellen:** Es können Interfaces und abstrakte Klassen definiert werden, die später (mittels ‚*implements*') implementiert werden sollen
  - Mehrfachvererbung von Schnittstellen (ohne Status und Verhalten)
- **Klassen:** Implementierungen von Schnittstellen. Es können von bestehenden Klassen spezielle Unterklassen (mittels ‚*extends*') abgeleitet werden.
  - Einfachvererbung von Implementierungen
  - vermeidet das Diamant-Problem
  - unveränderbare Implementierungen (final class, method, attribute)
- **Pakete** und **Pakethierarchien** als Modularisierungskonzept jenseits von Klassen
  - Namensgebung und Namensräume auf dieser Basis
  - keine Unterstützung von Mehrfachversionen
  - company-name.productname Präfix als Standard
  - Namensraum-Importe

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **Sichtbarkeitsklassen** von Attributen und Methoden (default, public, protected, private)
- **Ausnahmebehandlung** (exception handling): Es besteht die Möglichkeit, über Ausnahmen (mittels ‚try-catch'-Blöcken) vom Standard-Kontrollfluss abzuweichen
- **Threads und Synchronisation:** Nebenläufige Programmabläufe lassen sich mit Threads erzeugen und synchronisieren
- **Garbage Collection:** Nicht mehr referenzierte Objekte werden automatisch auf kontrollierbare Weise (‚*finalize*') zerstört

  Anwender hat darauf aus Sicherheitserwägungen keinen Einfluss
- **Objektserialisierung:** Objekte, welche die Schnittstelle *Serializable* implementieren, können in einen Datenstrom geschrieben oder aus einem solchen aufgebaut werden (z.B. Speichern in eine Datei)
- **Ereignisse (events):** werden einige Folien später genauer besprochen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

J2EE Architektur und Javas Komponentenmodelle für Middleware-Anwendungen

- Im Zentrum steht **Familie von Komponentenmodellen**

  Client-Schicht: Anwenderkomponenten, JavaBeans, Applets

  Webserver-Schicht: Servlets, JSP

  Anwendungsserver-Schicht: EJB in vier Varianten (stateless session, statefull session, entity, message-driven session)

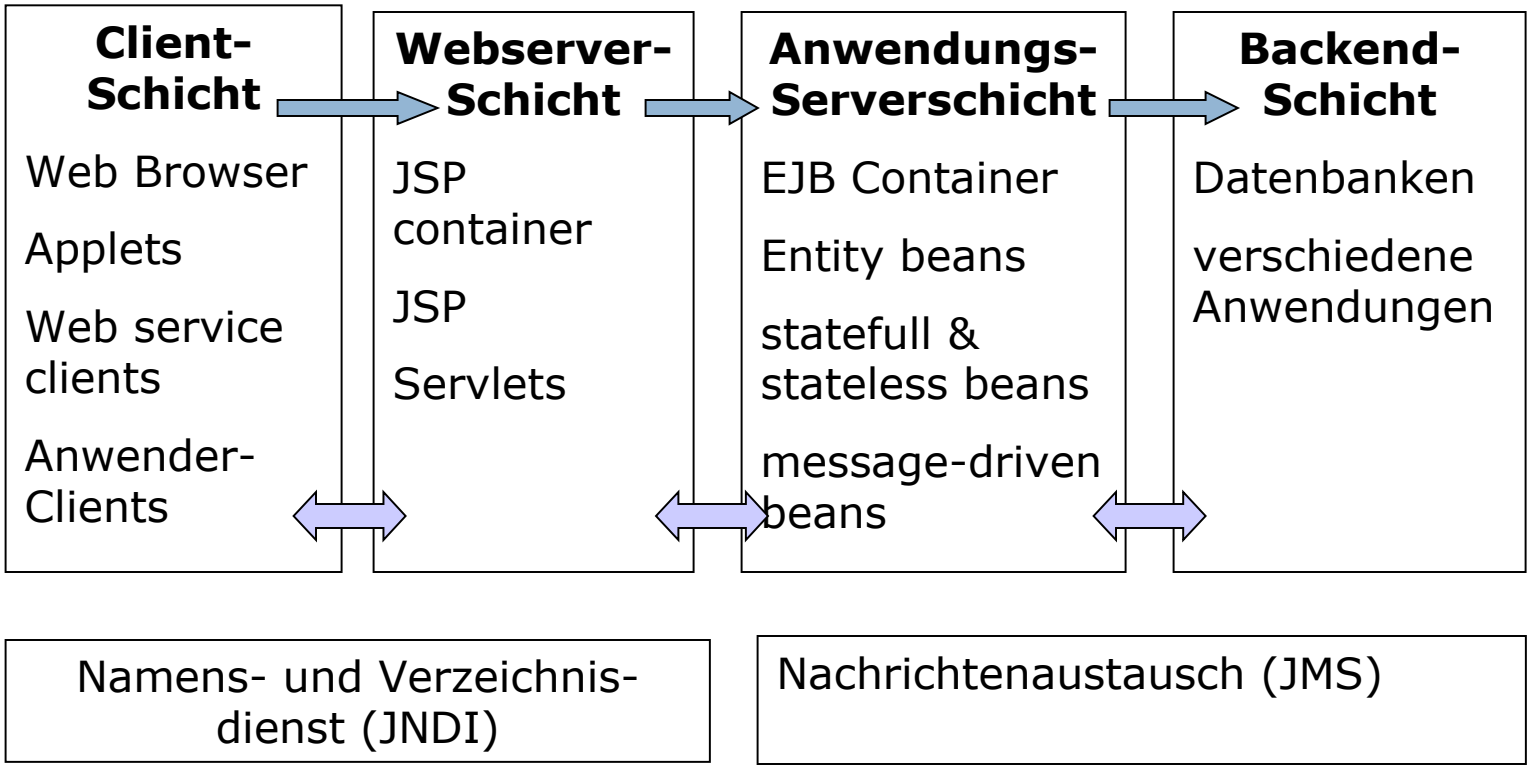
- Integrations-Ebenen (Basisdienste):

  Namens- und Verzeichnis-Infrastruktur (naming and directory interface, JNDI) sowie Nachrichten-Infrastruktur (Java messaging service, JMS) bilden die Klammern zwischen den verschiedenen Schichten

  weitere I.-E.: Transaktionskoordinierung, Sicherheitsdienste

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- J2EE Architektur
  - Kontrollfluss:
  - Datenfluss:



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java-Komponentenmodelle

- Komponentenmodelle in der Java 2 Enterprise Edition

  Servlets / Java Server Pages

  Enterprise JavaBeans und

  Application Client Components

  Alle drei Modelle sind serverseitige Komponentenmodelle.

  Wichtige Gemeinsamkeit ist die **Absetzung der Entpackungsphase** (deployment): Gesteuert durch Deployment-Deskriptor im XML-Format

  Entpackung = Prozess der Vorbereitung einer Komponente auf den Einsatz in einer speziellen **Umgebung** (context)

  Herstellen/Prüfen der (komponentenspezifischen) Voraussetzungen, unter denen die Komponente arbeitsfähig ist.

  Beispiel: Datenbank-Anbindung, Persistenzfragen

  wird unterschieden von der **Installation**, die (plattformspezifisch) eine entpackte Komponente in einer speziellen Hardware-Konfiguration verfügbar macht.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java Komponentenmodelle im Überblick

- **Applets**: herunterladbare leichtgewichtige Komponenten für den Einsatz in Webbrowser-Clients
  - Einsatzgebiet heute durch andere Technologien abgedeckt
- **JavaBeans**: unterstützt verbindungsorientiertes Programmieren
  - zentraler Ansatz: Beobachter und Ereignisse
  - wenig Gemeinsamkeit mit EJB (außer dem Namen)
- **Java Servlets** greifen den Gedanken von Applets auf, laufen jedoch auf dem Server ab und sind (meist) leichtgewichtige Komponenten
  - werden durch einen Web Server instanziiert
  - **Java Server Pages (JSP)**: verwandte Technologie, mit der dynamische Webseiten deklarativ definiert werden können
  - Vergleichbar mit den Microsoft Active Server Pages .NET (ASP.NET)
- **Application Client Components**
  - Im Gegensatz zum Prinzip des Webbrowsers konsistentere Verteilung der Funktionalität zwischen Client und Server (Rich Client)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Distribution und Packung von Java-Komponenten

- Geschäftsanwendungen (enterprise applications)
  ⇒ *.ear Dateien (enterprise archive)
- Servlets und JSP
  ⇒ *.war Dateien (web archive)
- Applets, JavaBeans, EJB
  ⇒ *.jar Dateien (Java archive)
- Entpackungs-Beschreibung (Deployment descriptor)
  - Formulierung von Anforderungen der Komponente durch Komponenten-Entwickler
  - Setzen offener Parameter (der „Blanks“) der Komponente durch Komponenten-Assemblierer
    - hart verdrahtet während der Entpackungsphase
    - Komponenten sind sonst zustandslos
  - Assemblierer ist eine andere Rolle als Komponentenentwickler, oft sogar in einer anderen Organisation
    - andere Ansätze trennen diese beiden Rollen deutlicher

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java Server Pages und Servlets

- dynamische Webseiten = Kombination dreier grundlegender Funktionen
  - ankommende Anfragen akzeptieren, auf Gültigkeit und Autorisierung prüfen und an geeignete Komponente zur Weiterverarbeitung abgeben
  - relevante Information aus den Informationsquellen extrahieren und den angefragten Inhalt (content) zusammenstellen
  - Inhalt an den Anfrager übermitteln
- Prototypisches Modell: wird von einem **Webserver** abgehandelt
  - HTTP-Anfragen empfangen
  - URL und enthaltene Parameter auswerten
  - statische oder dynamische HTML-Seite (generiert mittels Aktivierung einer Komponente, z.B. über eine einfache Schnittstelle wie CGI) zurücksenden
- Modell ist nicht auf HTML-Anfragen beschränkt
  - Szenario liegt allen typischen Web-Diensten (Web Services) zu Grunde
  - Dienste-Komponenten müssen nur eine simple Server-Schnittstelle implementieren

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **Realisierung 1:**
  Einbettung von Code direkt in das Markup einer HTML-Datei
  - Realisierung durch Active Server Pages (ASP) und Java Server Pages (JSP)
  - Aus den Script-Teilen wird HTML-Code generiert
  - Webserver ersetzt den Script-Code durch den generierten Code
- Beispiel einer einfachen JSP-Seite:

```
<HTML><BODY>
<%
    java.util.Calendar calendar = new java.util.GregorianCalendar();
    int hour = calendar.get(currTime.HOUR);
    int minute = calendar.get(currTime.MINUTE);
%>
The time is: <%= hour %>:<%= minute %> - or it was when I looked.
</BODY></HTML>
```

- **Realisierung 2: Erzeugung von Markup durch Java Code**
  - Java Servlets: Interaktion mit dem Nutzer auf der Browser-Seite über den Webserver

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Das gleiche Beispiel als implementiertes Servlet:

```
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet
{
  protected void doGet (HttpServletRequest req, HttpServletResponse resp)
  throws ServletException, java.io.IOException
  {
     java.util.Calendar calendar = new java.util.GregorianCalendar();
     int hour = calendar.get(currTime.HOUR);
     int minute = calendar.get(currTime.MINUTE);
     ServletOutputStream out = resp.getWriter();
     out.print("<HTML><BODY>\r\n");
     out.print("The time is: " +hour+ ":" +minute+
                 " - or it was when I looked.\r\n");
     out.print("</BODY></HTML>\r\n");
  }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

JSP versus Servlets:

- JSP erzeugen im Prinzip genau den Code des äquivalenten Servlets
- JSP existieren nur auf der Ebene von Java-Instanzen
  - keine Unterstützung der natürlichen Abstraktionsmechanismen von Java (Pakete, Klassen, Methoden)
- JSP können wie Servlets auf externen Java-Code zugreifen
  - Regel: Java Code in JSP auf absolut notwendigen „Klebstoff" reduzieren, funktionalen Teil in separate Klassen auslagern
- JSP: Konfusion mit clientseitigem JavaScript-Code möglich
- Ausdrucksmächtigkeit von JSP kann durch JSP Standard Tag Library (JSTL) aufgebohrt werden.
  - Besonders in Richtung XML-Verarbeitung (SOAP, Web Services)
  - Erweiterung des Sprachumfangs muss durch den Webserver unterstützt werden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vorteile des Servlet-Modells:

- Inhalts-Generierung kann über mehrere Servlets verteilt werden
- Trennung in Präsentationsgenerierung und Geschäftslogik
- weitergehende Faktorisierung von Servlets längs Aufgabengrenzen möglich
  - ⇒ Abstraktions- und Modellierungsprinzipien des klassischen Software-Entwurfs sind anwendbar
  - ⇒ Servlets als Komponentenmodell

- Servlets bieten sich auch als Einstiegspunkt in komplexere Geschäftsanwendungen an, die beispielsweise auf Enterprise JavaBeans aufsetzen
  - Probleme mit der Mischung unterschiedlicher Komponenten-Modelle:
    - mehrere Infrastrukturen müssen vorgehalten werden und interferieren
    - Kommunikation zwischen den Modellen muss entworfen werden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Einleitung

- keine Gemeinsamkeiten mit **JavaBeans**
  - Komposition durch verbindungsorientierte Programmierung
    - Beans können Ereignisquellen und –beobachter sein
    - Ereignisfluss wird festgelegt durch Anbinden von Quellen in einer Bean an Beobachter in anderen Beans

- Das **EJB-Konzept** realisiert dagegen einen klassischen OO-Zugang
  - Kommunikation über Methodenaufrufe und Objektgenerierung
  - Spezifikation, kein konkretes Produkt
  - Im Mittelpunkt steht ein **kontextuelles Kompositionskonzept**
    - automatische Komposition von Komponenteninstanzen mit zugehörigen Ressourcen und Diensten
    - Komponenten-Container-Architektur – der Container stellt die Laufzeitumgebung der Komponenten zur Verfügung und kapselt diese von der Umgebung (dem **Kontext**)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- das EJB-Konzept basiert auf **e-Beans** und **EJB Containern**
- In der Deployment-Phase erfolgt über den EJB Container eine kontextuelle Zusammenführung der Beans mit Diensten und Ressourcen
  - Container ist vergleichbar mit statischen Methoden, Beans mit Instanzmethoden einer Java-Klasse
  - Container ist der „Pate" der Beans, über welchen die gesamte Kommunikation läuft
  - Bean-Instanzen „leben" als Objekte (in unserem Sinne) in der Container-Laufzeitumgebung
  - EJB Container werden von EJB-Servern bereitgestellt
    - J2EE Standard: J2EE application server
- Beschreibung (Inhalt, Relationen, Rollen-, Sicherheits- und Transaktionsverhalten) in einem speziellen **Deployment-Deskriptor**
  - da solche Deskriptoren umfangreich sein können, ist Werkzeugunterstützung sowohl zur Erstellung als auch zur Entpackung erforderlich

- 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
- Kommunikation zwischen (clientseitigen) Stub-Klassen und (serverseitigen) Klassen im Container durch Java RMI
  - **EJBHome** und **EJBObject** Client-Klassen können wie bei CORBA aus der Interface-Beschreibung generiert werden.
  - Zu beiden gibt es **Local**-Varianten für den Einsatz in einer lokalen Umgebung
  - Container wird nach dem Factory-Prinzip innerhalb des J2EE-Servers aus den Schnittstellen-Informationen **EJBHome** und **EJBObject** und der **Bean-Klasse** erzeugt
- Komponente in unserem Sinne ist also die Schnittstellen-Information, die Bean-Implementierung und die Metainformationen über das Zusammenspiel.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Deployment-Kontrakt

- Komponente wird in einem speziellen Paketformat ausgeliefert, das eine genaue Entpackungsbeschreibung im XML-Format enthält
  - Dienst-Schnittstelle, Factory-Schnittstelle, Bean-Implementierung, Ressourcen-Zuordnung, Metainformationen

Lebenszyklus-Kontrakt

- Komponente implementiert spezielle Lebenszyklusfunktionen, die vom Container „automagisch" aufgerufen werden können
  - OnActivate() {…}
  - OnPassivate() { … }

Container-Service-Kontrakt

- Der Container stellt der Komponente ein Kontext-Objekt oder eine Schnittstelle zur Verfügung, über welche die Komponente transparent Dienste aus dem Kontext in Anspruch nehmen kann.
  - WhoIsCaller() { … }
  - AccessDataBase() { … }
  - LocateComponent() { … }

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Umgebungs-Kontrakt

- Der Container sichert eine funktionierende Umgebung für die Komponente entsprechend der Deployment-Information.

Erweiterungs-Kontrakt

- Der Container kann selbst erweiterbar sein nach dem Open-Close-Prinzip (kontextuelle Komposition)
- Verhaltensänderungen ausgerollter Komponenten
  - nutzergetriebene Unterbrechungen
  - Unterstützung der Laufzeitkonfiguration von Einheiten
  - Einbindung weiterer Dienste zur Laufzeit
  - Versionsmanagement ausgerollter Komponenten
  - Aspektorientiertes Verhalten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Client-Container-Kontrakt

- Client nutzt Komponentendienste über den Container.
- Framework bietet Dienste zum Auffinden der Komponente
  - zentralisierte (wie CORBA) oder dezentralisierte (P2P, Web Services)
- typischer Ablauf:
  - Suche Dienst: Client $\rightarrow$ Framework-Infrastruktur
  - Finde den Container: Framework $\leftrightarrow$ Container
  - Finde die Komponenten-Schnittstelle: Framework $\leftarrow$ Client
  - Objekt erzeugen: Client $\rightarrow$ Container
  - Objektzeiger zurückliefern: Client $\leftarrow$ Container
  - Dienst in Anspruch nehmen: Client $\rightarrow$ Komponente

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Enterprise JavaBeans EJB 2.1

- **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

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
  - ähnlich wie cast für **FirmalImpl** im CORBA-Beispiel
  - Entwickler muss auf Übereinstimmung der Signaturen selbst achten

Bean-Klassen

Es gibt vier Sorten von Enterprise JavaBeans

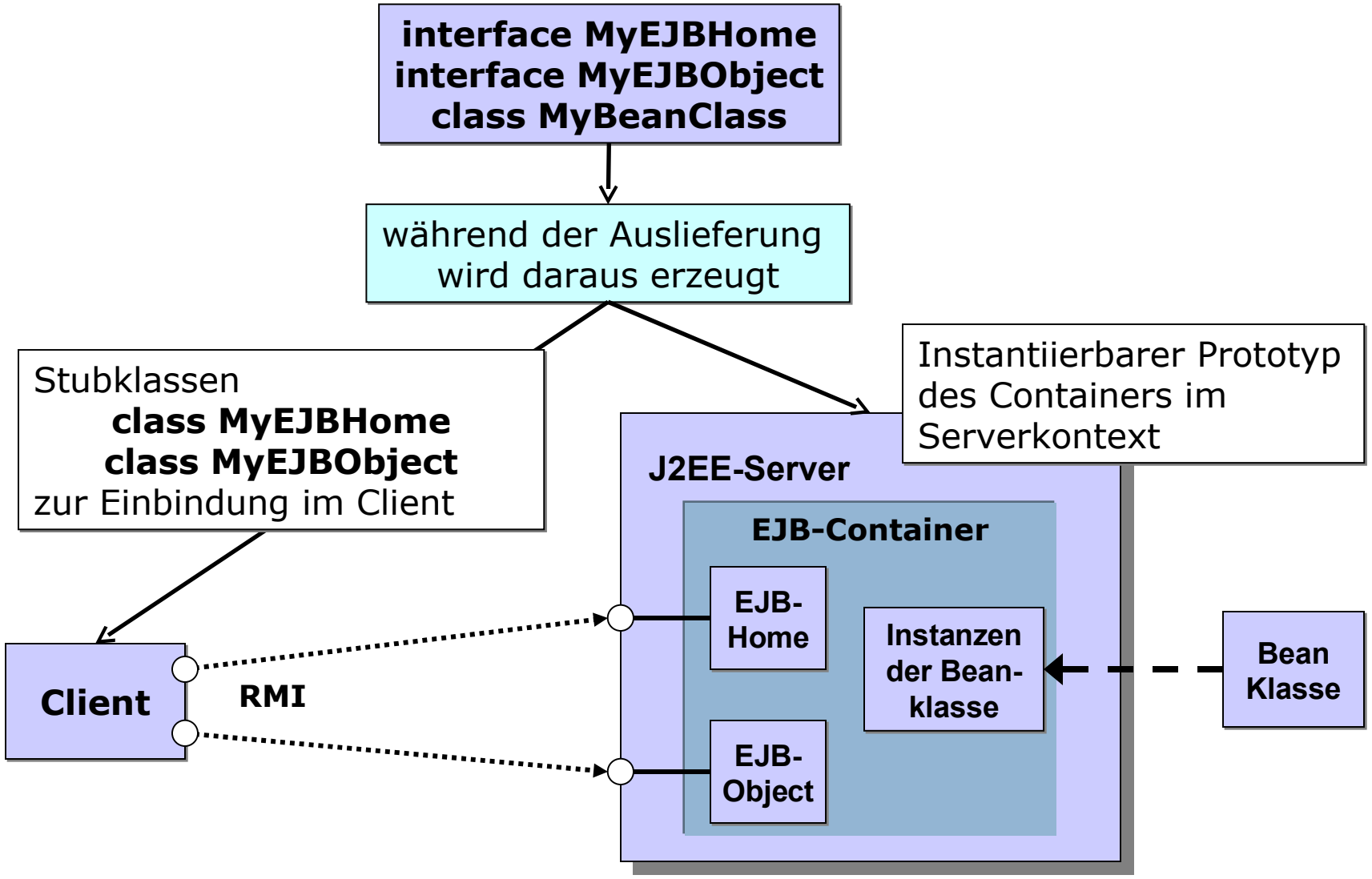
- **zustandslose** (stateless) und **zustandsbehaftete** (stateful) **Session Beans**
  - implementieren **javax.ejb.SessionBean**
  - entsprechen einer Session in der Datenbankterminologie
  - beide Arten sind transient, also nur innerhalb einer Session gültig
  - Zustand wird zwischen verschiedenen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

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

Container-Schnittstelle

```
public interface BuchungHome extends javax.ejb.EJBHome {
    public Buchung create(int owner_id)
        throws java.rmi.RemoteException, javax.ejb.CreateException;
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Klasse

```
public class BuchungBean implements javax.ejb.SessionBean {

    public BuchungBean() {}

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

    // Implementierung der Schnittstelle SessionBean
    private javax.ejb.SessionContext ctx;
    public void setSessionContext(javax.ejb.SessionContext sc) {
        this.ctx=sc; // Session-Context für Callback-Methoden
    }
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
}
```

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

```
public class Client {

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

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

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

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