Vorlesung
Software aus Komponenten

3. Komponenten-Modelle

apl. Prof. Dr. Hans-Gert Gräbe
Wintersemester 2008/09

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

- **<u>Realisierung 1</u>:**
  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>
```

- **<u>Realisierung 2</u>: 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**)

- 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
    - Eigene Schnittstelle in EJB 2, Anbindung durch Annotationen in EJB 3
  - 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
- Unterscheide (a) Methoden der Bean-Instanzen sowie (b) Methoden zum Management des Lebenszyklus der Beans
- EJB 2 bietet dafür zwei Schnittstellen, EJB 3 verwendet (a) POJO – plain old java objects – und (b) Annotationen
- Kommunikation zwischen (clientseitigen) Stub-Klassen und (serverseitigen) Klassen im Container durch Java RMI
- 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

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

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

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

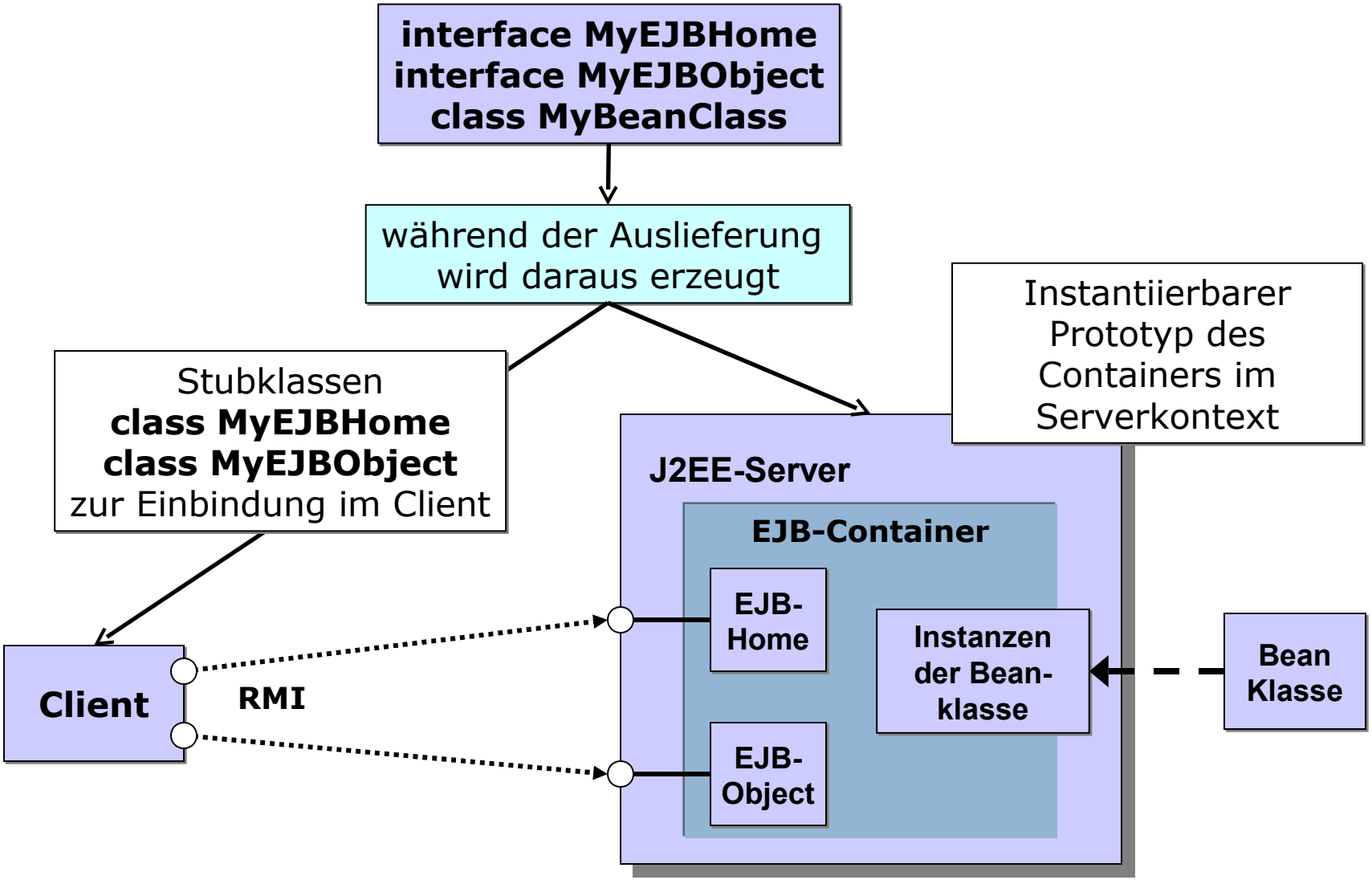
Es gibt vier Sorten von Enterprise JavaBeans

- **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


interface MyEJBHome
interface MyEJBObject
class MyBeanClass
während der Auslieferung wird daraus erzeugt
Instantiierbarer Prototyp des Containers im Serverkontext
Stubklassen
class MyEJBHome
class MyEJBObject
zur Einbindung im Client
J2EE-Server
EJB-Container
EJB-Home
Instanzen der Bean-klasse
Bean Klasse
Client
RMI
EJB-Object

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

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");
```

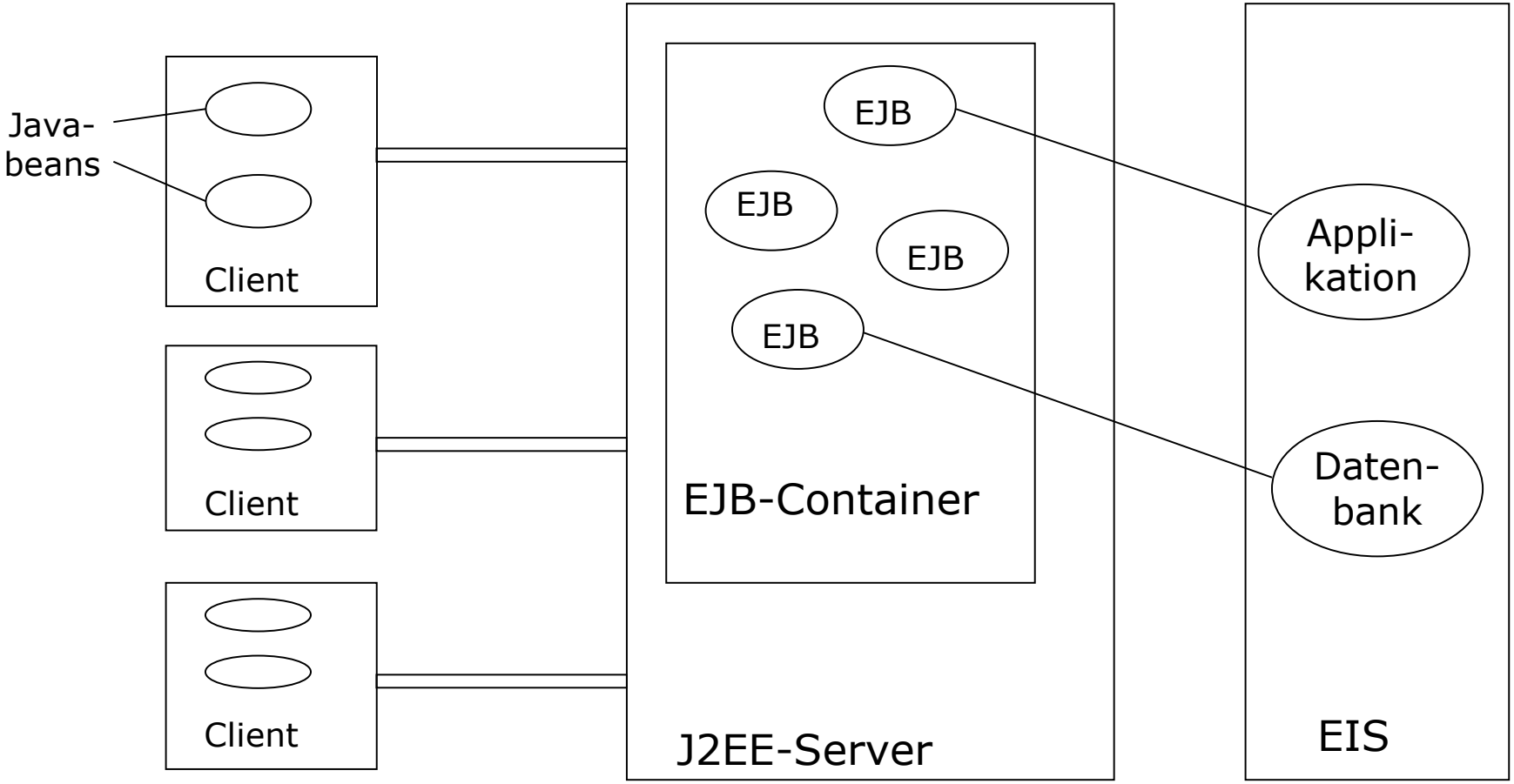
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

```
   // 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) { }
}
```

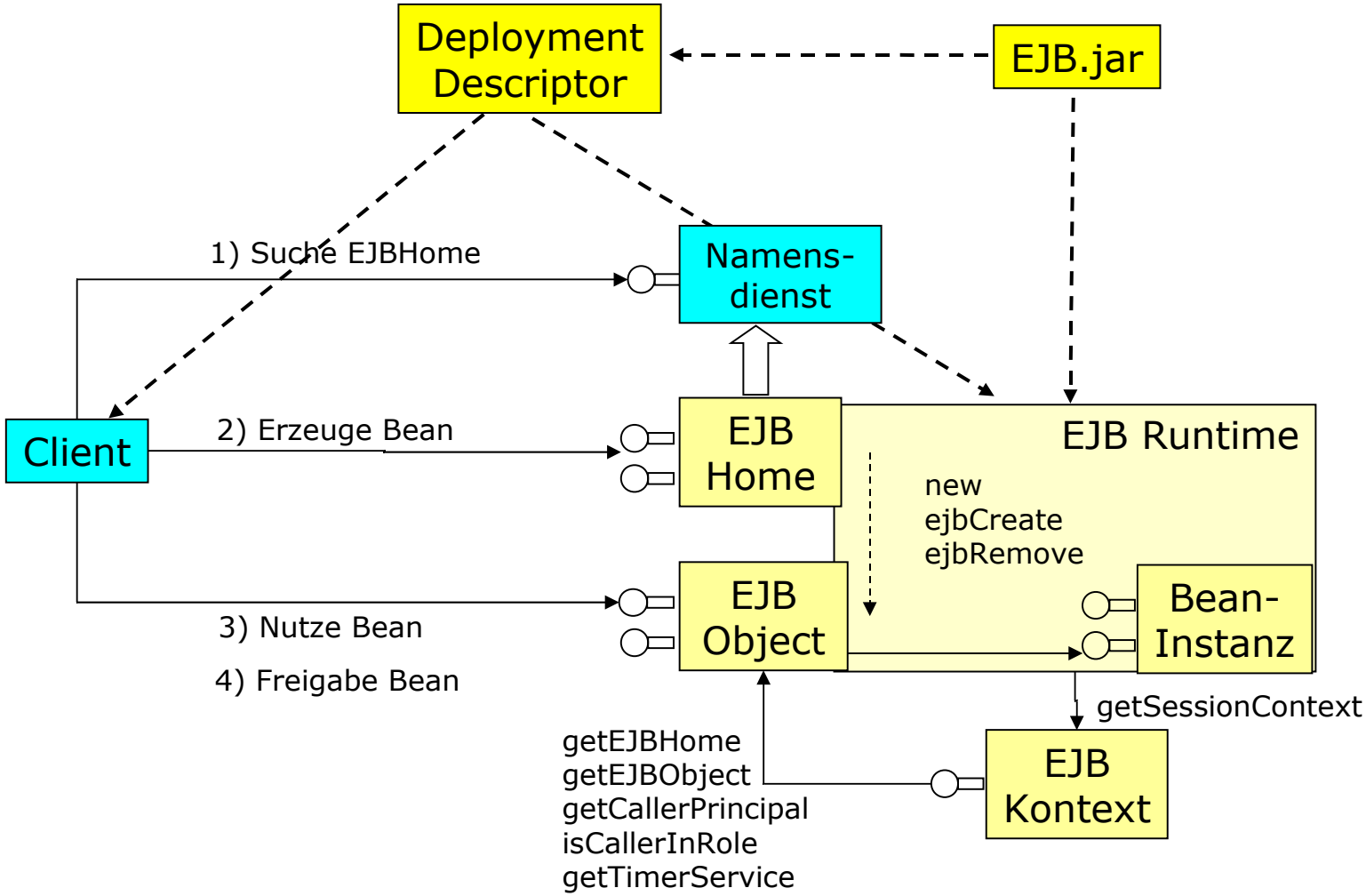
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Architektur von EJB 2



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Architektur von EJB 2



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Probleme des Standards

- Viele Schnittstellen zu implementieren (EJBHome, EJBObject, callback zum Kontext)
- Nur eine Geschäftsmethoden-Schnittstelle pro EJB Bean
- Weitere Konventionen sind zu beachten (RMI, spezielle Basis-Schnittstellen)
- Zusätzliche und andere Konventionen für EJB-Klassen im Vergleich zu normalen (plain old) Java-Klassen
- Konfiguration von Beans und Applikationen für Beans erfolgt durch riesige XML-basierte Deployment-Deskriptoren
- EJB-Laufzeit zu rudimentär spezifiziert
- Komplexität der Interaktion mit der Persistenzschicht
- Keine Schnittstellen für Logging, Tracing und andere Basisdienste, um Komponenten zu testen und in der Laufzeit zu verfolgen.
- Beans müssen vom Client manuell gesucht und angesprochen werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Ziele von EJB 3

- Genauere Spezifikation des Bean-Lebenszyklus durch mehr Erweiterungspunkte im Kontext
- Dependency Injection: Einzelne Attributwerte, Methoden- oder Klassendefinitionen werden zur Laufzeit aus dem Kontext importiert
- Verwendung von Meta-Daten
- Interzeptoren und aspektorientierte Ansätze für Infrastrukturdienste
- Vereinfachung der Persistenzbehandlung
- Reduktion der Zahl der Artefakte, die ein Bean-Entwickler bereitstellen muss
- Vereinfachung der EJB Typen durch Reduktion der Zahl der Schnittstellen
- Verbesserung der Testmöglichkeiten außerhalb des Containers

Wie wird das umgesetzt?

- Einsatz von POJO's (plain old java objects) und POJI's (plain old java interfaces)
  - Dienstschnittstelle als Java-Schnittstelle
  - kein Home-Interface mehr
  - Annotation durch Metadaten für die Konfiguration der EJB-Typen, Local/Remote, Transaktionen, Sicherheit
- Dependency Injection
  - für Attribute, Eigenschaften, Ressourcennutzung
  - werden aus der Umgebung durch Annotationen in die Laufzeit der Beaninstanz „injiziert"
- Erweiterter Lebenszyklus-Unterstützung
  - nutzerdefinierte Callback-Methoden
- Interzeptoren
  - Unterbrechung der Geschäftsmethode in AOP-Manier

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Programmiermodell: Die Bean-Klasse

- Bean-Klasse als zentrales Artefakt, das bereitgestellt wird
- keine Home-Schnittstelle mehr, Laufzeitunterstützung kommt direkt aus dem Kontext
- Beantyp wir durch Annotation festgelegt
- Beanklasse kann Ausnahmen über Annotation vom Typ @**ApplicationException** werfen
- Beanklasse wirft keine **java.rmi.RemoteException** mehr

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

@Stateful public class BuchungBean implements Buchung {
    public void buchen(Kunde k, Seminartyp s) {
        //Code zur Ausführung der Buchung
    }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Programmiermodell: Ereignisse im Lebenszyklus

- Ereignisse im Lebenszyklus werden durch den Container über annotierte Callbacks an die Bean-Klasse weitergegeben
- Callbacks können **RuntimeException** auslösen, die Rollback von Transaktionen nach sich ziehen, aber keine **ApplicationException**.
- Letzteres kann durch annotierte **CallBackListener** implementiert werden

```
@Stateful public class BuchungBean implements Buchung {
    @PreDestroy dispose() { … }
}
```

Programmiermodell: Interzeptoren

- Interzeptoren unterbrechen die Abarbeitung einer Geschäftsmethode
- Interzeptormethoden können in der Beanklasse oder in eigenen Interzeptorklassen definiert sein
  - können **RuntimeException** oder **ApplicationException** werfen, wie in der Schnittstelle der Geschäftsmethode vereinbart
  - Abarbeitung im selben Transaktions- und Sicherheitskontext wie die Geschäftsmethode
  - Können JNDI, JDBC, JMS, andere Beans und den **EntityManager** (Schnittstelle zum Persistenzkontext) nutzen
  - verwendet Dependency Injection
- Für dieselbe Geschäftsmethode können mehrere Interzeptoren definiert sein
- Signatur einer Interzeptormethode:

  public Object methodName(invocationContext) throws Exception

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Programmiermodell: Interzeptoren (Fortsetzung)

- **InvocationContext** definiert als

```
public interface InvocationContext {
  public Object getBean();
  public Method getMethod();
  public Object[] getParameters();
  public void setParameters(Object[]);
  public EJBContext get EJBContext();
  public java.util.Map getContextData();
  public Object proceed() throws Exception;
}
```

- Kontext-Daten werden von allen Interzeptoren derselben Routine gemeinsam genutzt.
- Aufrufende von **proceed**() innerhalb des Interzeptors - Rückkehr in die unterbrochene Routine

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Wichtige Annotationen und deren Defaultwerte

- **@Local** und **@Remote**: Annotation von Klassen und Schnittstellen
  - Default: @Local

```
@Remote public interface Buchung {
    public void buchen(Kunde k, Seminartyp s) ;
}
```

- **@TransactionManagement**: Annotation der Beanklasse
  - Default TransactionManagementType.CONTAINER
- **@TransactionAttribute**: Annotation der Klasse oder einzelner Methoden
  - Beschreibt das Transaktionsverhalten
  - Default: REQUIRED
- **@Timeout**: Definition einer Timeout-Methode der Bean
- **@ApplicationException**
- Weitere Annotationen für Sicherheitsaspekte:
  - **@RolesReferences**, **@RolesAllowed**, **@RunAs**, **@SecurityRoles**

Zustandslose Beans

- Implementieren plain old Java Interfaces
- keine Home-Schnittstelle, Laufzeitkontrolle durch Kontext
- Annotation **@Stateless**
- Definierte Callbacks **@PostConstruct** und **@PreDestroy**
- Interzeptor **@AroundInvoke** um eine einzelne Session
- Unspezifizierte Transaktions- und Sicherheitskontexte
- Dependency Injection muss vor dem ersten Aufruf einer Geschäftsmethode erfolgen

Bean und Interzeptor. Beispiel

```
@Stateless @Interceptors ({MyInterceptor.class})
public class BuchungBean implements Buchung {
   public void buchen(Kunde k, Seminartyp s) { … }
}


public class MyInterceptor {
   @AroundInvoke
   public Object zeitVerbrauch(InvocationContext ic) throws Exception {
          long time = System.currentTimeMillis();
          try { return ic.proceed(); }
          finally {
               long total = System.currentTimeMillis() - time;
               System.out.println("Aufruf von " + ic.getMethod().getName()
               + " dauerte "+total+" Millisekunden.");
          }
}
```

Zustandsbehaftete Beans

- Implementieren plain old Java Interfaces
- keine Home-Schnittstelle, Laufzeitkontrolle durch Kontext
- Annotation **@Stateful**
- kann Schnittstelle **SessionSynchronization** implementieren
- Definierte Callbacks **@PostConstruct**, **@PreDestroy**, **@PostActivate**, **@PrePassivate**
- Interzeptoren
  - **@afterBegin** – Ausführung zu Beginn jeder Session
  - **@beforeCompletion** – Ausführung von Ende jeder Session
  - **@AroundInvoke** – Einbettung einer Session
- Dependency Injection muss vor dem ersten Aufruf einer Geschäftsmethode erfolgen
- Client bekommt Referenz auf Session direkt über JNDI oder über Dependency Injection
- Annotation **@Remove** für Methode zum Auflösen der Beaninstanz

Zustandbehaftete Bean. Beispiel

```
@Stateful public class BuchungBean implements Buchung {
    Connection rc;
    @Resource SessionContext sc;

    @PostConstruct @PostActivate public void init() {
            rc = Connection.Open();
    }

    @PreDestroy @PrePassivate public void close() {
            try { rc.close(); }
            catch (CloseException) { /* no op */ }
    }
// … weiter
```

```
    @Remove public void dispose() { /* was auch immer */ }

    @AroundInvoke public Object monitor(InvocationContext ic) {
            try {
                Object res = ic.proceed();
                if ( (OpResult) res == OpResult.SUCCEED ) {
                    System.out.println("Aufruf war okay");
                }
                return res;
            } catch (Exception ex) { throw ex; }
    }

    // Geschäftsmethode
    public void buchen(Kunde k, Seminartyp s) { … }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zustandbehaftete Bean. Beispiel

Zugriff durch den Client:

```
static public void testBean(Kunde k, Seminartyp s) throws Exception {
    javax.naming.Context ctx = new javax.naming.InitialContext();

    Buchung test = ctx.lookup(Buchung.class.getName());

    // oder Injection, wenn innerhalb einer anderen Bean:
    /* @EJB Buchung test; */

    test.buchen(k,s);          // Test der Geschäftsmethode
    test.dispose();                    // Test der Bean dispose-Methode
}
```

Nachrichtengesteuerte Beans

- Implementieren Nachrichten-Listener-Schnittstelle des jeweiligen Nachrichtentyps (bspw. **javax.jms.MessageListener**)
- Annotation **@MessageDriven**
- implementiert nicht notwendig **javax.ejb.MessageDrivenBean**
- kann Schnittstelle **SessionSynchronization** implementieren
- Definierte Lebenszyklus-Event Callbacks **@PostConstruct**, **@PreDestroy**
- Interzeptoren für Listener-Aufrufe
- Dependency Injection muss vor dem ersten Aufruf einer Geschäftsmethode erfolgen

```
@MessageDriven(activateConfig = { … })
public class ExampleMDB implements iavax.jms.MessageListener {
    public void onMessage(Message msg) { … }
                                                }
```

Bean-Kontext und Umgebung

- Bean spezifiziert ihre Abhängigkeiten durch **Abhängigkeits-annotationen** für Typ, Name und Charakteristika eines benötigten Objekts oder Ressource

```
@Resource (name = "DB", type = "javax.sql.CataSource.class")
```

- Bean-Entwickler annotiert erforderliche Instanzvariablen, die vom Container automatisch nach Setzen von **EJBContext** und vor dem Aufruf der ersten Geschäftsmethode aus dem Kontext instanziiert werden

```
@Stateless public class MySessionBean implements MySession {
    @Resource(name="DB") public DataSource myDB;
    public void getData() { // ohne try-catch !
            Connection c = myDB.getConnection();
                                                    }
                                                  }
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Kontext und Umgebung (Fortsetzung)

- Setter Injection als Alternative: Der Container ruft die Setter-Methode auf, statt die Attributvariable direkt zu instanziieren

```
@Resource(name="DB")
public void setDataSource(DataSource myDB) { this.myDB = myDB;}

@Resource public void setSessionContext(Context ctx) { this.ctx=ctx; }
```

Persistenz

- Entity-Klassen sind durch **@Entity** annotiert und müssen einen **public-Konstruktor ohne Argumente** haben.
  - können abstract oder konkret sowie in Vererbungshierarchien eingebunden sein
  - müssen (in der Regel) Schnittstelle **Serializable** implementieren
- Instanzen einer Entity-Klasse sind leichtgewichtige persistente Objekte: Persistenz auf der Ebene einzelner Attribute
  - Kennzeichnung durch Annotation
  - Persistenz-Anbieter greift auf entsprechende Attribute zu; diese müssen nicht nach außen für andere sichtbar sein
  - Zugriff direkt auf das Attribut oder über Getter-Methode (Default)
- Default: Abbildung auf Datenbank-Tabelle mit demselben Namen
  - Primärschlüssel müssen aus einer Klasse sein, die **equals** und **hashCode** korrekt unterstützt.

Persistenz. Beispiel

```
@Entity @Table(name ="KundenListe")
public class KundenListe implements java.io.Serializable {
    private int id; // Primärschlüssel für die Kundenliste
    private Collection<Kunde> kundenListe;

    @Id(generate = GenerationType.AUTO)    public int getId() { return id; }
            // spezifiziert, dass dies die Getter-Methode für den Primärschlüssel ist
    public void setId(int id) { this.id = id; }

    @OneToMany public Collection<Kunde> getKunden() { return kundenListe; }
            // Auslesen der Kundenliste
...
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Persistenz (Fortsetzung)

- Persistente Aggregate (Ganzes aus mehreren Teilen) werden über die Annotation **@Embedded** innerhalb einer **@Entity** realisiert
- Eine Entity-Bean kann auf mehrere Datenbank-Tabellen abgebildet werden: **@SecondaryTable**
- Relationen zwischen Entitäten über Annotationen **@OneToOne**, **@OneToMany**, **@ManyToOne**, **@ManyToMany**
  - können unidirektional oder bidirektional sein
- Persistenz kann in Vererbungshierarchien mit vererbt werden: **@Inheritance**

EntityManager

- **EntityManager** ist für die Verwaltung des Lebenszyklus von Entity-Beans verantwortlich
- Assoziiert mit dem Persistenzkontext, also einer Menge von Entity-Beans

```
@Stateless public class AuftragsListe {
   @PersistenceContext EntityManager em;
   public void neuerAuftrag(int kundenId, Auftrag neuerAuftrag) {
            Kunde k = (Kunde) em.find("Kunde", kundenId);
            k.getAuftraege().add(neuerAuftrag);
            neuerAuftrag.setKunde(k);
   }
}
```

- EntityManager können vom J2EE-Server zur Verfügung gestellt werden, aber auch von der Applikation → **EntityManagerFactory**

Weitere Konzepte

- Transaktionskontrolle: entweder via JTA oder über EntityTransaction
- Persistenz-Kontexte: Innerhalb eines solchen Kontexts wird die Eindeutigkeit der Zuordnung Entity – Persistenzobjekt garantiert
- Query API mit eigener, an SQL angelehnter EJB Query Language
- Entity Packaging: Im Deployment-Deskriptor können genauere Informationen über EntityManager gespeichert werden.

Zusammenfassung

- Aufgaben des EJB-Container werden an den Kontext delegiert
- Spezielle Eigenschaften des Ausführungskontexts werden durch Annotationen direkt im Quelltext festgelegt statt als Deployment-Informationen
- Persistenz folgt dem Konzept der Java Data Objects JDO
- Standard fixiert in JSR 220

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Server-Provider (Server-Anbieter)

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

EJB-Kontext-Provider (Container-Anbieter)

- setzt auf Plattform des Server-Providers auf
- Benutzerverwaltung, Transaktionsmanagement, Persistenz
- Herstelllung der Installationswerkzeuge
- Implementierung der EJB-Kontextfunktionalität
- 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