# Vorlesung
# Component Ware und Web-Services

## - Komponentenkonzepte -

9. Java Servlets/JSP & Enterprise JavaBeans (EJB)

Dr. Hans-Gert Gräbe, F. Schumacher
Wintersemester 2003/2004

## Inhalt

- Einleitung
- Java Servlets / Java Server Pages (JSP)
- Enterprise Java Beans
  - Einleitung
  - Aufbau
    - Bean-Schnittstelle
    - Container-Schnittstelle
    - Bean-Klassen
    - Deployment-Deskriptor
  - Architektur
    - Container
    - Arbeitsweise
    - Kommunikation
    - Persistenz
    - Identifikation
- Beispiel Seminarorganisation
- Rollenkonzept

## Einleitung

### Komponentenmodelle im Java-Umfeld

- neben den Applets und JavaBeans gibt es drei weitere Komponentenmodelle im Java-Umfeld (alle sind Teil der Java 2 Enterprise Edition)
  - **Servlets / Java Server Pages**
  - **Enterprise JavaBeans** und
  - **Application Client Components**
- Alle drei Modelle sind serverseitige Komponentenmodelle.
- Wichtige Gemeinsamkeit ist das Konzept der **deployment-Deskriptor** im XML-Format
  - **Deployment** = 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
  - zu unterscheiden von der **Installation**, die (plattformspezifisch) eine entfaltete Komponente in einer speziellen Hardware-Konfiguration verfügbar macht.

## Einleitung

- **Applets**: herunterladbare leichtgewichtige Komponenten für den Einsatz in Webbrowser-Clients
  - Einsatzgebiet rasch durch andere Technologien (GIF animated images, Shockwave, Flash, JavaScript) 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**
  - unspezifizierte Java-Applikationen auf einem Client, die auf definierte Weise auf einen Server und dessen Ressourcen zugreifen

## Einleitung

Distribution und Deployment

- Geschäftsanwendungen (enterprise applications)
  - ⇒ *.ear Dateien (enterprise archive)
- Servlets und JSP
  - ⇒ *.war Dateien (web archive)
- Applets, JavaBeans, EJB
  - ⇒ *.jar Dateien (Java archive)
- Deployment-Beschreibung
  - Formulierung von Anforderungen der Komponente
  - Setzen offener Parameter (der „Blanks" der Komponente)
  - Deployer ist eine andere Rolle als Komponentenentwickler, oft sogar in einer anderen Organisation
    - andere Ansätze trennen diese beiden Rollen deutlicher

## Java Servlets / JSP

### 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

## Java Servlets / JSP

- Einfachste Schnittstelle: Einbettung von Code direkt in das Markup einer HTML-Datei
  - Realisierung durch Active Server Pages (ASP) und Java Server Pages (JSP)
  - Aus den Script-Teile wird HTML-Code generiert
  - Webserver ersetzt den Script-Code durch den generierten Code
- JSP: JSP-Seiten werden in Servlets kompiliert
  - JSP-Server aktiviert bei Bedarf (Anfragen) solche Servlets
- Servlets können auch direkt über das API erstellt werden
  - Umkehrung des Programmiermodells:
    Einbettung von HTML Markup in Java-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>
```

## Java Servlets / JSP

- 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");
  }
}
```

## Java Servlets / JSP

- Java Server Pages
  - spezielle Tags <% ... %> sowie <%= ... %>
  - JSP-Seite entspricht Instanz einer Servlet-Klasse
  - Regel: Java-Code in einer JSP-Seite minimieren
    - inhaltlich ausrichten auf erforderlichen „Klebstoff"
    - Java Abstraktionsmechanismen (Klassen, Pakete) verwenden
- Spezielle Tags zur Einbindung von JavaBeans-Instanzen in JSP-Seiten durch Erweiterung der Klasse **javax.servlet.jsp.tagext.TagSupport**

```
<% taglib uri="/hour" prefix="calendar" %>
<% taglib uri="/minute" prefix="calendar" %>
<HTML><BODY>
The time is: <calendar:hour>:<calendar:minute> - or it was when I looked.
</BODY></HTML>
```

## Java Servlets / JSP

- JSTL (JSP standard tag library) stellt auch Tags zur Steuerung des Kontrollflusses und zur XML-Verarbeitung zur Verfügung
  - Damit kann JSP zur Implementierung einfacher Web-Dienst-Protokolle aufgebohrt werden (Ansatz von ASP.NET)
  - JSP-Server muss um die entsprechenden Elemente erweitert werden

- Inhalts-Generierung kann über mehrere Servlets verteilt werden
  - Trennung in Präsentationsgenerierung und Geschäftslogik
  - weiter gehende Faktorisierung von Servlets längs Aufgabengrenzen

    $\Rightarrow$ Servlets als Komponentenmodell

- Servlets bieten sich auch als Einstiegspunkt in komplexere Geschäftsanwendungen an, die beispielsweise auf Enterprise JavaBeans aufsetzen
  - Probleme mit den unterschiedlichen Komponenten-Modellen

## Enterprise JavaBeans (EJB)

### Einleitung

- keine Gemeinsamkeiten mit JavaBeans
  - 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
  - Könnte mit entsprechenden Konzepten (Beans-Container, Ereignisentkopplung durch InfoBus) auch andere Kompositionskonzepte (datenorientierte bzw. kontextorientierte Komposition) realisieren
  - spielt aber heute praktisch kaum eine Rolle, obwohl die Infrastruktur dafür vorhanden ist
- Das EJB-Konzept realisiert einen klassischen OO-Zugang
  - Kommunikation über Methodenaufrufe und Objektgenerierung
  - Komposition von e-beans nur über eigenes Design
  - generische verbindungsorientierte Kompositionskonzepte werden nicht unterstützt

## Enterprise JavaBeans (EJB)

### Einleitung

- Spezifikation, kein konkretes Produkt
- Im Mittelpunkt steht ein **kontextuelles Kompositionskonzept**
  - automatische Komposition von Komponenteninstanzen mit zugehörigen Ressourcen und Diensten
- 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
  - EJB Container werden von EJB-Servern bereitgestellt
    - J2EE Standard: J2EE application server
- Beschreibung in einem speziellen **Deployment-Deskriptor**

## Enterprise JavaBeans (EJB)

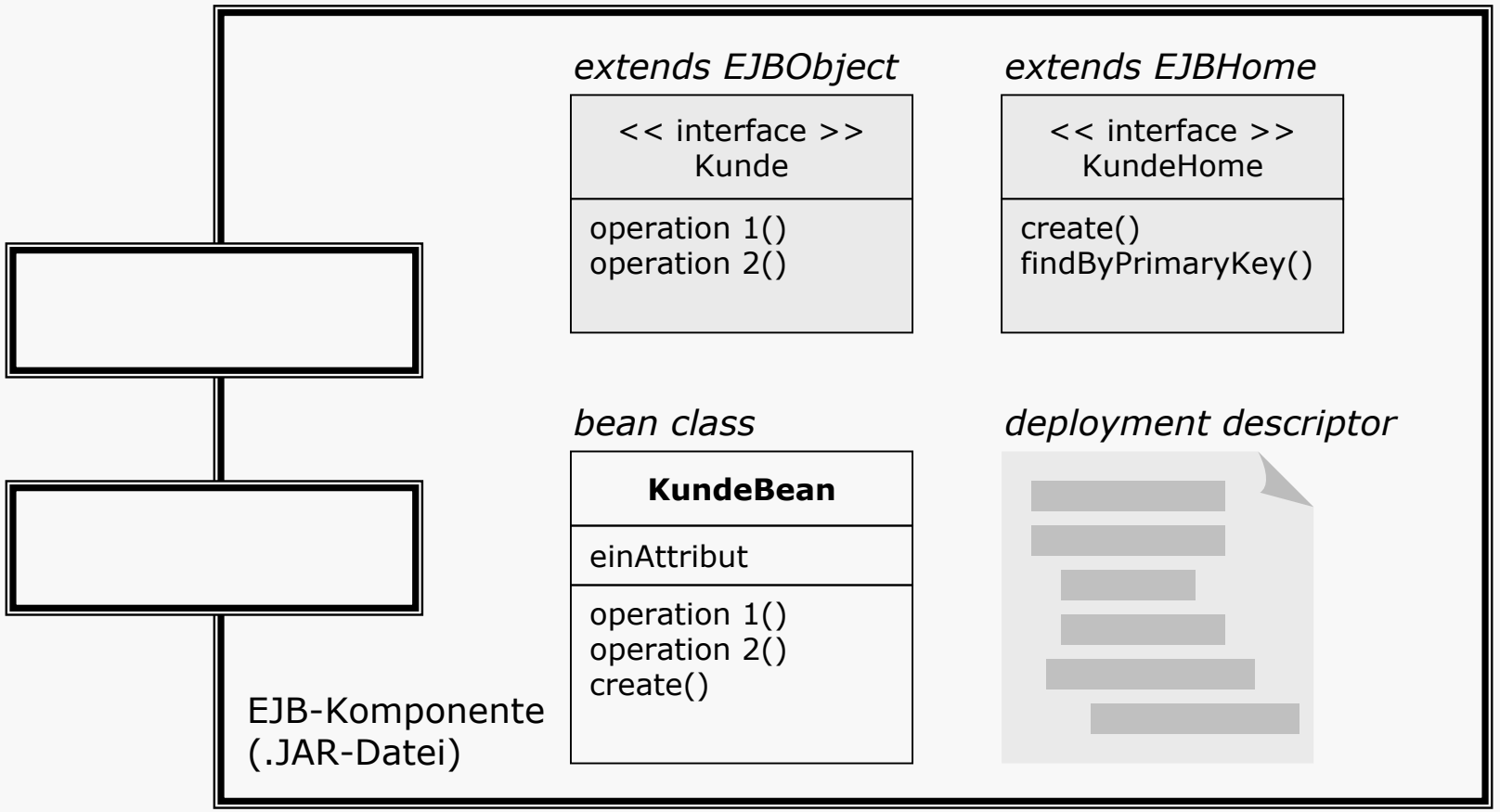
### Einleitung

- kein direkter Zugriff auf Attribute und Methoden von Beans, auch nicht innerhalb desselben Containers
  - 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
- Zu beiden gibt es **Local**-Varianten für den Einsatz in einer lokalen Umgebung

- Implementation von:
  - Borland (Borland AppServer)
  - HP (HP Bluestone)
  - IBM (IBM WebSphere Application Server)
  - Oracle (Oracle 9iApplicationServer)
  - ❖ weitere: http://java.sun.com/j2ee/compatibility.html

## Aufbau

### Schema



## Aufbau

### Bean-Schnittstelle EJBObject

- Java-Interface, welches von **javax.ejb.EJBObject** erbt
- beschreibt Dienstleistung für Clients
  - Client muss die Schnittstelle implementieren, um die Dienste der Bean in Anspruch zu nehmen
- Nichtlokale Clients implementieren Schnittstelle auf Stub-Objekten, die über RMI oder RMI-über IIOP mit dem EJB Container kommunizieren
  - deklarierte Operationen müssen Exception **java.rmi.RemoteException** auslösen können
  - Abfangen von Kommunikationsproblemen im Netz
  - Lokale Clients können lokale Version der Schnittstelle (Erweiterung von **EJBLocalObject**) implementieren, wenn von der e-bean bereitgestellt
- Attribute durch get-/set-Operationen
- es können mehrere Bean-Schnittstellen spezifiziert werden
- Können in der Deployment-Phase aus der jar-Datei der Bean generiert werden

## Aufbau

### Container-Schnittstelle EJBHome

- erbt von **java.ejb.EJBHome**
- verwaltet Bean-Instanzen im Container
  - Erzeugen neuer Instanzen
  - Auffinden vorhandener Instanzen
- 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 Link auf den Container-Kontext
  - **ejbCreate** / **ejbRemove**
    - Besonderheit bei entity beans
  - **ejbPassivate** / **ejbActivate**

## Aufbau

### Bean-Klassen

- Implementierung der zur Verfügung gestellten Operationen
- Implementierungen der Bean-Schnittstellen werden **nicht** mitgeliefert, sondern in der Deployment-Phase erzeugt
- muss dennoch Operationen der Schnittstellen zur Verfügung stellen
  - Ausnahme: **findByPrimaryKey** (wird automatisch generiert)
- Entwickler muss auf Übereinstimmung der Signaturen selbst achten
- Es gibt vier Sorten von Enterprise JavaBeans
  - **zustandslose** (statless) und **zustandsbehaftete** (stateful) **Session Beans**
    - implementieren **javax.ejb.SessionBean**
    - entsprechen einer Session in der Datenbankterminologie
    - beide Arten sind transient
  - **Entity Beans**
    - implementiert **javax.ejb.EntityBean**
    - enthalten persistente Daten
    - entsprechen einem Datensatz in der Datenbankterminologie
  - **nachrichtengesteuerte** (message-driven) **Beans** (neu in EJB 2.0)

## Aufbau

### Deployment-Deskriptor

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

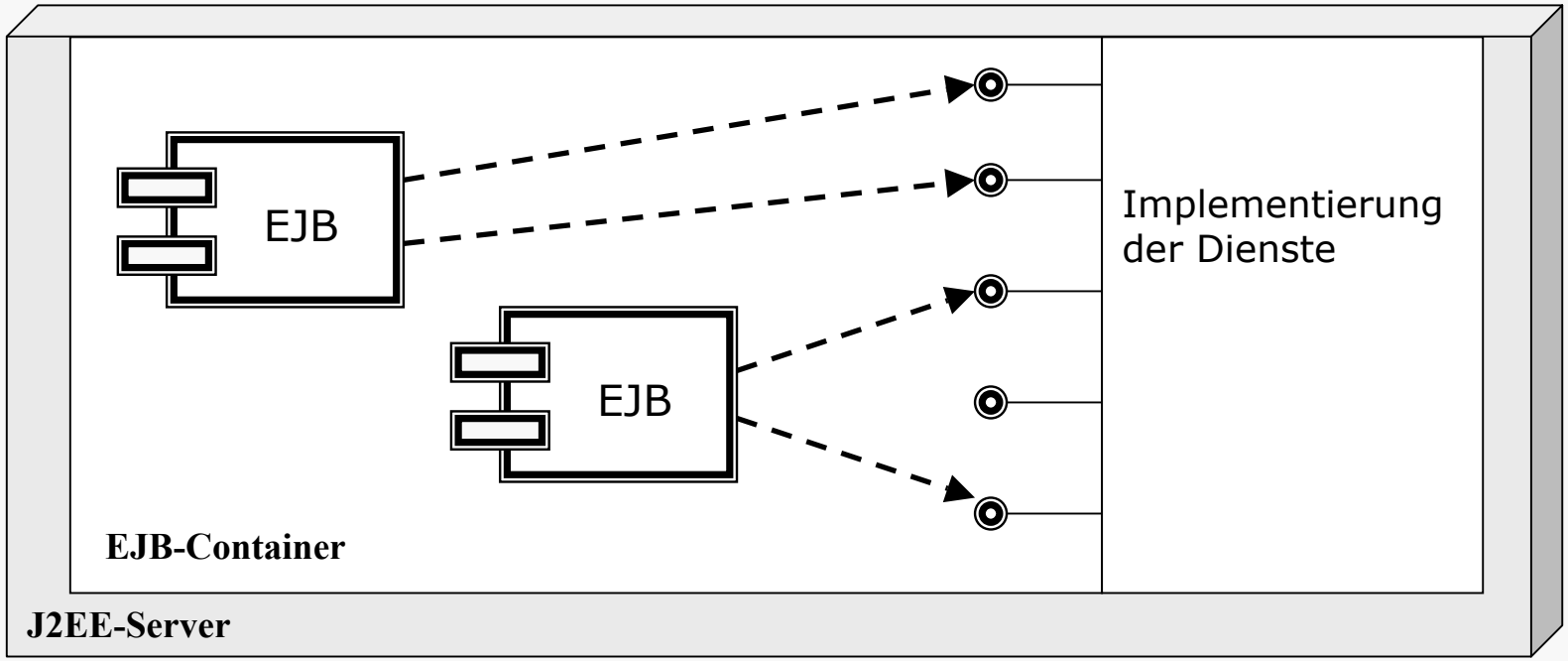
## Architektur

### EJB–Container

- JVM wird benötigt, ist aber allein nicht ausreichend
- auf einem J2EE–Server können mehrere EJB–Container gleichzeitig laufen
- in einem Container können wiederum mehrere (Arten von) Beans gekapselt sein
- Schnittstelle und Verhalten von Container und J2EE sind spezifiziert
- Container stellt Dienste zur Verfügung
  - 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
- optional: eine Bean kann eine Schnittstelle implementieren, über die sie vom Container über Ereignisse informiert wird

# Architektur

## Schema



## Architektur

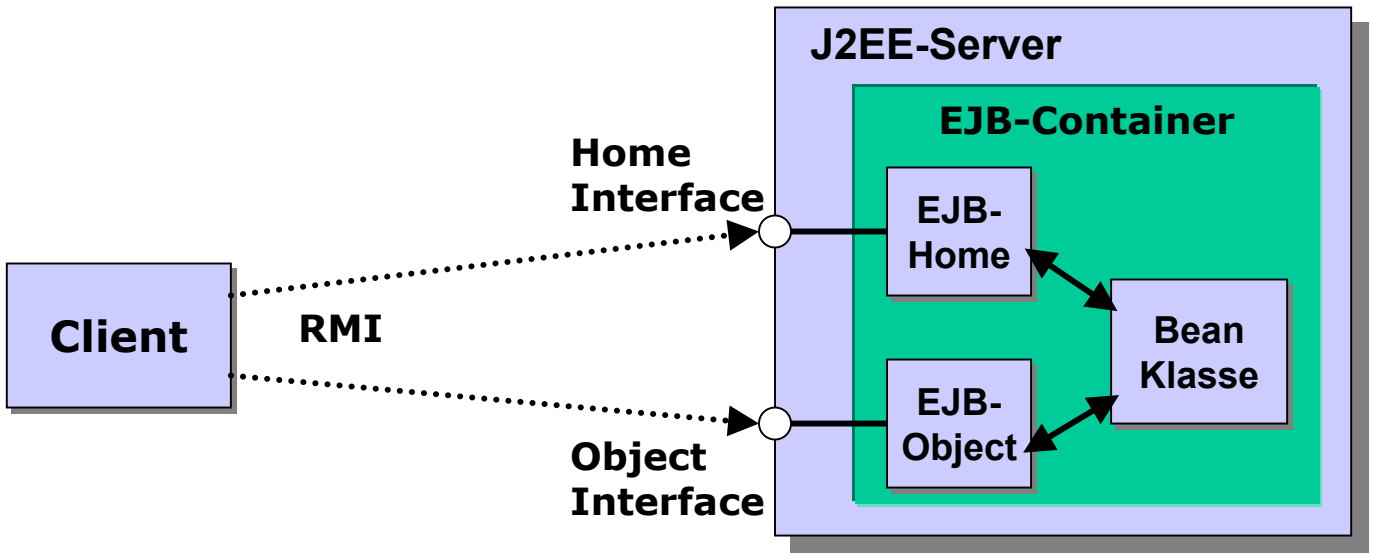
### Arbeitsweise

- Deployment-Phase:
  - EJBHome- und EJBObject-Klassen werden generiert
  - Einbindung der Implementierungen von Local- und Remote-Interface
  - Anlegen der Skelettklassen
- Laufzeit:
  - Erstellung von Skelettobjekten (2 pro e-Bean) für Remote-Anbindung
  - Client ruft Operationen der Skelettobjekte
  - Prüfung, Verwaltung, dann Weiterleitung an Bean-Klasse
- Container für Skelettklassen verantwortlich

## Architektur

### Kommunikation

- immer über Skelettobjekte
  - hohe Ortstransparenz und Skalierbarkeit
- standardmäßig über RMI
- alternativ RMI über IIOP (Internet Inter ORB Protocol) von CORBA



## Architektur

### Persistenz

- relevant für entity beans (Datenkomponenten)
- 2 Modelle: container managed persistence, bean managed persistence
- Art wird im Deployment-Deskriptor angegeben
- container managed persistence:
  - Container ist verantwortlich
  - bean wird auf Tabelle einer relationalen Datenbank abgebildet
  - Attribute eines Beans-Objekts als Datensatz gespeichert
  - Auslesen und Setzen der Attribute
  - automatische Persistenz
- bean managed persistence
  - Bean ist verantwortlich

## Architektur

### Identifikation

- Primärschlüssel, Home–Objekt
- Primärschlüssel setzt sich aus Teilmenge der Attribute der bean zusammen
- **findByPrimaryKey** sucht Bean-Instanz im Container
- falls nötig, wird neues Objekt erstellt

### Ereignisse

- kein Ereignismodell
- Versenden und Empfangen von Ereignissen über Mail-API

## Beispiel Seminarorganisation

### 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
}
```

## Beispiel Seminarorganisation

### Container-Schnittstelle

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

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

## Beispiel Seminarorganisation

### Bean-Klasse

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

public class BuchungBean implements SessionBean
{
    //Operationen der Home-Schnittstelle Buchung
    public void ejbCreate() throws RemoteException, CreateException
        { //Initialisierung des Objekts }

    //Operationen der Schnittstelle SessionBean des Containers
    public void ejbRemove()
        { //Aufräumarbeiten }

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

## Beispiel Seminarorganisation



### 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>
```

## Beispiel Seminarorganisation

### Ein einfacher Client

```
package SemOrg.Schnittstellen;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.naming.InitialContext;
import SemOrg.Schnittstellen.*;

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

## Beispiel Seminarorganisation

### Ein einfacher Client

```
public void start()
   {
      try
      {
         ...
         BuchungHome home=(BuchungHome)
                                    javax.rmi.PortableRemoteObject.
                                    narrow(temp, BuchungHome.class)
         Buchung bean = home.create();
         bean.buchen(einKunde, einSeminartyp);
         bean.remove()
      }
      catch(Exception e)
      { }
   }
}
```

# Rollenkonzept

## Allgemein

- Ziel:
  - flexible, leicht wart- und skalierbare Anwendungen
  - unternehmensübergreifende Zusammenarbeit
- Problem:
  - hoher Komplexitätsgrad von Unternehmenslösungen
  - Aufwand bei Installation und Verwaltung
- Lösung:
  - Aufteilung der Verantwortlichkeiten für unterschiedliche Stadien des Entwicklungsprozesses
  - Spezialisierung auf Rollen

# Rollenkonzept

## 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

# Rollenkonzept

## 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

# Rollenkonzept

## 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

## Quellen

- "Lehrbuch der Softwaretechnik" von Helmut Balzert
  - LE 29 | Kap. 3.9.4. | S. 916 – 924
- Sun Microsystems Homepage
  - http://java.sun.com/products/ejb