

Vorlesung Software aus Komponenten

3. Komponentenmodelle

apl. Prof. Dr. Hans-Gert Gräbe
Wintersemester 2009/10

3.5. Enterprise Java Beans

EJB 2.1

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

3.5. Enterprise Java Beans

EJB 2.1

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

3.5. Enterprise Java Beans

EJB 2.1

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

3.5. Enterprise Java Beans

EJB 2.1

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

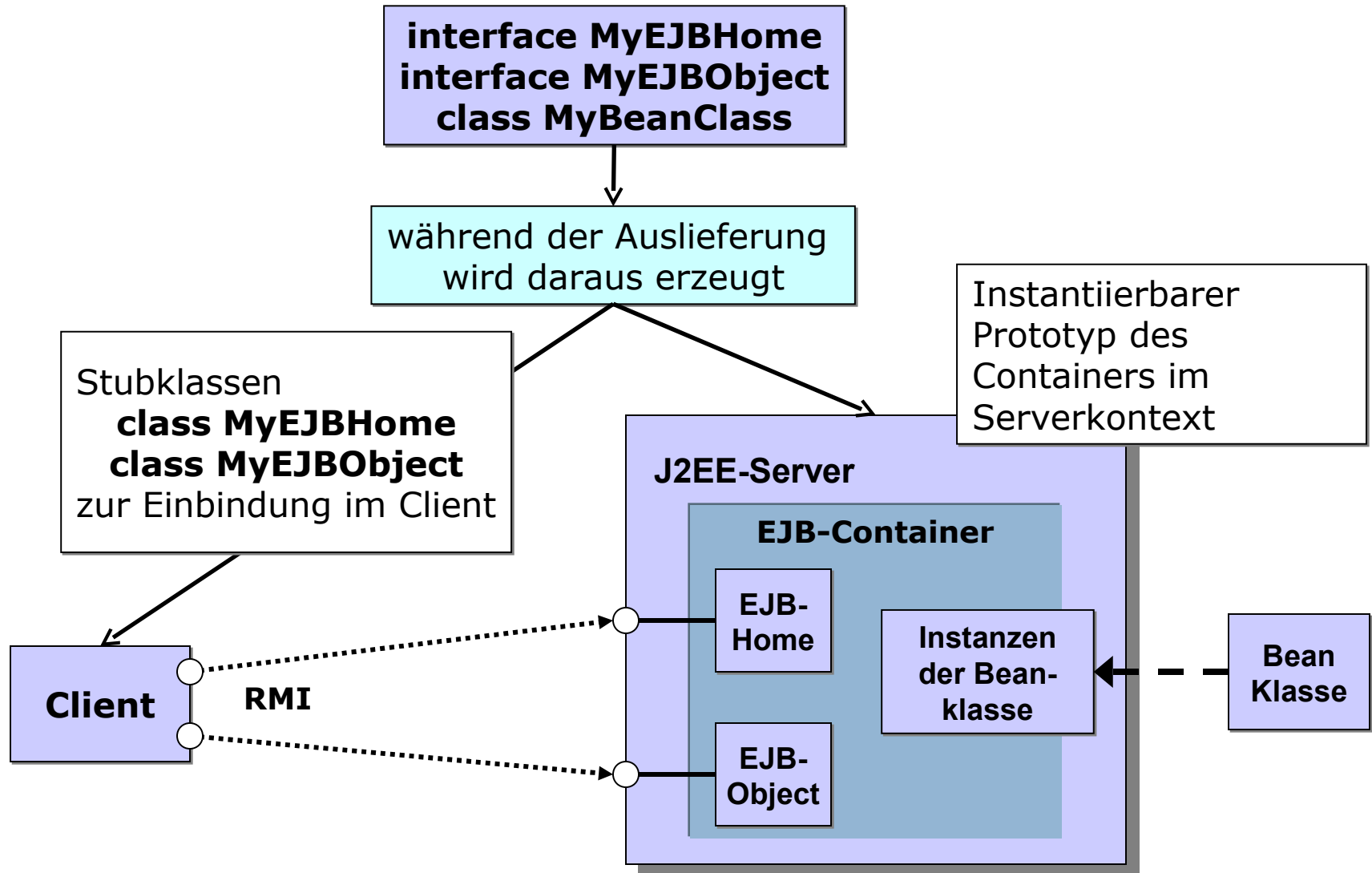
3.5. Enterprise Java Beans

EJB 2.1

- 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

3.5. Enterprise Java Beans

EJB 2.1



3.5. Enterprise Java Beans

EJB 2.1

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

3.5. Enterprise Java Beans

EJB 2.1

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

Bean-Schnittstelle

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

Container-Schnittstelle

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

Bean-Klasse

```
package SemOrg.Server;
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() {}
}
```

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>
```

Ein einfacher Client

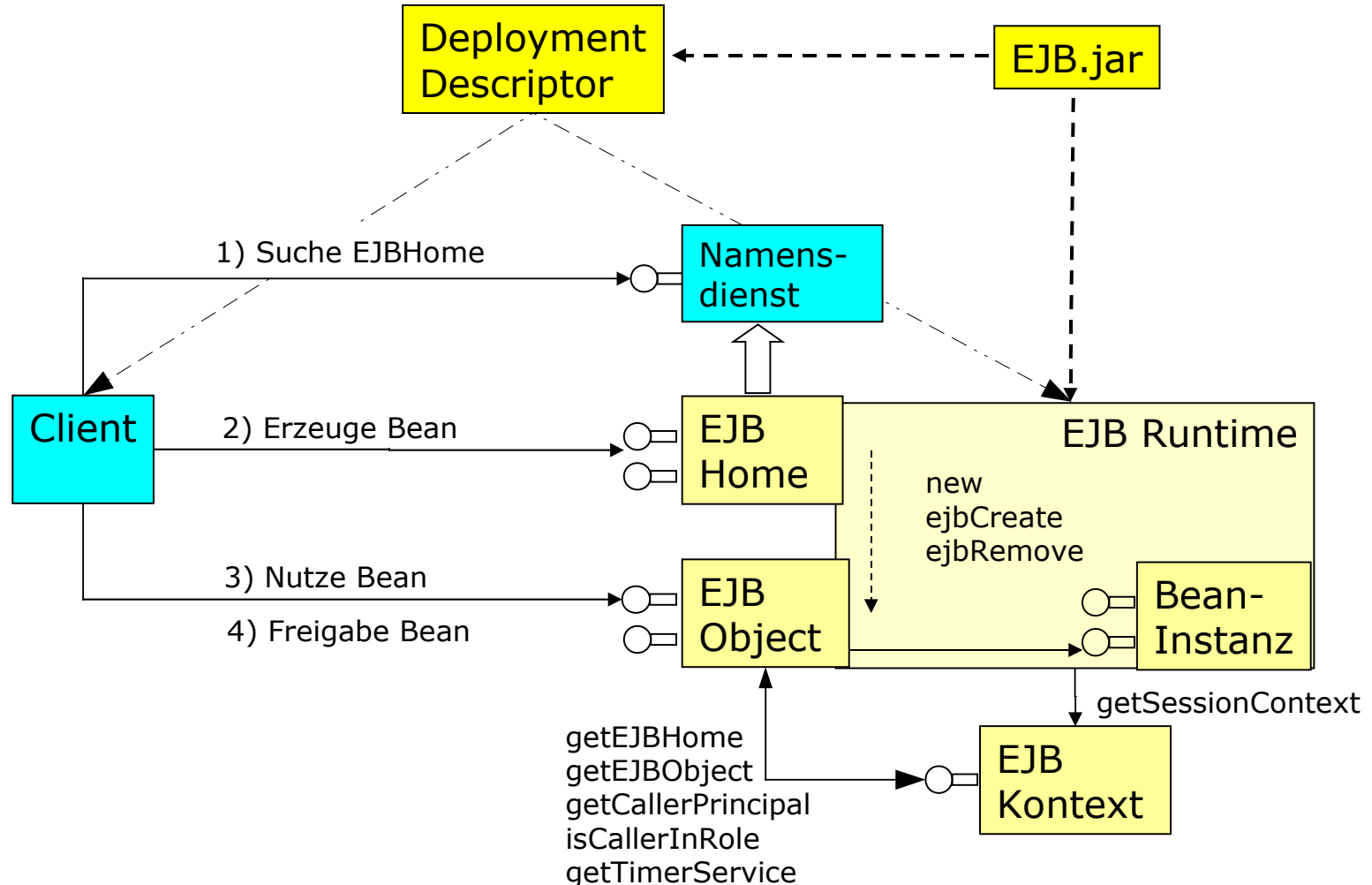
```
public class Client {  
  
    public static void main(String[] args) {  
        Client einClient=new Client();  
        einClient.run();  
    }  
  
    public void run() {  
        Kunde einKunde=getKunde();  
        Seminar einSeminar=getSeminar();  
  
        try {  
            // Namenskontext des Servers finden  
            javax.naming.Context ctx = new javax.naming.InitialContext();  
            Object temp = ctx.lookup("java:comp/env/Buchung");  
        }  
    }  
}
```

3.5. Enterprise Java Beans

Beispiel Seminarorganisation

```
// EJB-Container-Objekt vom Typ BuchungHome  
// auf dem Server finden und instanziiieren  
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) { }  
}
```

Architektur von EJB 2



3.5. Enterprise Java Beans

EJB 2 – Nachteile

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.

3.5. Enterprise Java Beans

EJB 3

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

3.5. Enterprise Java Beans

EJB 3

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

Programmiermodell: Die Bean-Klasse

- Bean-Klasse als zentrales Artefakt, das bereitgestellt wird
- keine Home-Schnittstelle mehr, Laufzeitunterstützung kommt direkt aus dem Kontext
- Beantyp wird 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  
    }  
}
```

3.5. Enterprise Java Beans

EJB 3

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() { ... }  
}
```