Enterprise Java Beans

Fachhochschule Hagenberg
WS 2001 / 2002

Silbergrau Consulting & Software GmbH
Dr. Andreas Erlach

Inhaltsübersicht

- Application Server
- J2EE Architektur bzw. EJB Spezifikation
- Session, Entity und Message-Driven Beans
- EJB Deployment
- EJB Clients (Thin / Rich)
- Best Practices und Probleme
- EJB Application Server Auswahl
- Technologie-Vergleich (CORBA, COM+)

Session Bean I

- Session Bean
  - nicht persistent
    - muss nach einem Absturz des Servers neu erzeugt werden
  - existiert nur für einen Client
  - entspricht einem Use Case
  - kann Stateless bzw. Stateful (hält Conversational State) sein
  - kann Transaktionen Container bzw. Bean managed (CMT/BMT) implementieren

Session Bean II

- Session Bean besteht aus
  - Home Interface zum Erzeugen einer Session Bean
  - Remote Interface definiert die Methoden der Session Bean
  - EnterpriseBean implementiert die Methoden des Home/Remote Interface

Anm.: Methoden, Parameter und Returnwerte müssen den RMI Regeln entsprechen

Session Bean III

- Anforderungen an ein SessionBean Home Interface
  - Muss von javax.ejb.EJBHome abgeleitet sein
  - Methoden müssen den RMI/IIOP Regeln entsprechen
    - Argumente und Return values müssen gültige RMI/IIOP Typen sein und die throws-Clause muss java.rmi.RemoteException enthalten.
  - Home Interface muss eine oder mehrere create(...) Methoden definieren
    - Zu jeder create-Methode muss es eine entsprechende ejbCreate-Methode geben.
    - Der Return type einer create-Methode muss das Remote Interface der Session Bean sein.
    - Die throws-Clause muss zumindest javax.ejb.CreateException enthalten.

Session Bean IV

- **public interface EJBHome extends Remote**
  - An enterprise Bean's home interface defines the methods that allow a client to create, find, and remove EJB objects.
- **void remove(Handle handle) throws RemoteException, RemoveException;**
  - Remove an EJB object identified by its handle
- **EJBMetaData getEJBMetaData() throws RemoteException;**
  - Obtain the EJBMetaData interface for the EJB
- **HomeHandle getHomeHandle() throws RemoteException;**
  - Obtain a handle for the home object
- Home Interface einer Session Bean muss zumindest create Methode implementieren
- **RemoteInterface create() throws CreateException, RemoteException;**

Session Bean V

- Anforderungen an ein SessionBean Remote Interface
  - Muss von javax.ejb.EJBObject abgeleitet sein
  - Methoden müssen den RMI/IIOP Regeln entsprechen
  - Für jede Business Methode des Remote Interface muss es eine entsprechende Methode (mit gleicher Signatur) in der Session Bean Klasse geben

Session Bean VI

- **public interface EJBObject extends Remote**
  - The EJBObject interface is extended by all enterprise Bean's remote interface. An enterprise Bean's remote interface provides the client's view of an EJB object.
- **EJBHome getEJBHome() throws RemoteException;**
  - Obtain the enterprise Bean's home interface.
- **void remove() throws RemoteException, RemoveException;**
  - Remove the EJB object.
- **Handle getHandle() throws RemoteException;**
  - Obtain a handle for the EJB object.
- **boolean isIdentical(EJBObject obj) throws RemoteException;**
  - Test if a given EJB object is identical to the invoked EJB object.

Session Bean VII

- Anforderungen an eine Session Bean Klasse
  - Muss das javax.ejb.SessionBean implementieren
  - Muss public sein und darf weder final noch abstract sein
  - Muss einen public Default-Konstruktur definieren
  - Darf die finalize-Methode nicht implementieren
  - Kann (muss nicht) das Remote Interface implementieren
  - Muss entsprechende Business- und ejbCreate-Methoden implementieren
  - Argumente und Return-Types der Business- und ejbCreate-Methoden müssen den RMI/IIOP Regeln entsprechen
  - Kann das javax.ejb.SessionSynchronization Interface implementieren (CMT)
  - Kann beliebige andere (Hilfs-)Methoden implementieren

Session Bean VIII

- **public interface SessionBean extends EnterpriseBean**
  - The SessionBean interface is implemented by every session enterprise Bean class.
- **void setSessionContext(SessionContext ctx) throws EJBException, RemoteException;**
  - Set the associated session context. The container calls this method after the instance creation.
- **void ejbRemove() throws EJBException, RemoteException;**
  - A container invokes this method before it ends the life of the session object.
- **void ejbActivate() throws EJBException, RemoteException;**
  - The activate method is called when the instance is activated from its "passive" state.
- **void ejbPassivate() throws EJBException, RemoteException;**
  - The passivate method is called before the instance enters the "passive" state. The instance should release any resources that it can re-acquire later in the ejbActivate() method.
- **void ejbCreate() throws EJBException, RemoteException;**

Session Bean IX

- **public interface SessionContext extends EJBContext**
- **EJBObject getEJBObject() throws IllegalStateException;**
  - Returns the session bean’s remote interface.
- **EJBHome getEJBHome();**
  - Returns the session bean’s home interface.
- **Identity getCallerIdentity();**
  - Returns the Principal of the invoker of the Session Bean.
- **boolean isCallerInRole(Identity);**
  - Tests if the session bean caller has a particular role.
- **boolean getRollbackOnly();**
  - Tests if the current transaction is marked for rollback (Only for container-managed transaction demarcation).
- **void setRollbackOnly();**
  - Marks the current transaction for rollback.
- **UserTransaction getUserTransaction() throws IllegalStateException;**
  - UserTransaction can be used to demarcate transactions and to obtain transaction status.

Session Bean X

- Session Beans können Stateful sein
  - einen Zustand über mehrere Aufrufe hinweg halten (Conversational State)
  - Zustand wird in non-transient Fields der SessionBean gehalten
- Werte müssen folgenden Regeln entsprechen
  - null oder ein serialisierbares Objekt
  - ein EJB Remote bzw. Home Interface
  - ein SessionContext oder eine UserTransaction
  - JNDI Environment naming context (java:comp/env)
  - alles andere muss bei ejbPassivate „bereinigt“ werden
- Zustand wird bei Bedarf auf Hintergrundspeicher gespeichert

Session Bean XI

- Stateless Session Bean sollte verwendet werden
  - wenn ein Client keine bestimmte Session Bean Instanz benötigt
  - die Client-spezifischen Informationen nicht sehr umfangreich sind
- Session Beans sind allerdings oft stateful
- Der Vorteil einer stateless Session Bean ist die bessere Skalierbarkeit

Transaction Handling I

- EJB Server sind transaktionsorientiert
  - erzeugen und koordinieren Transaktionen mit den Resourcemanagern
  - Transaktionssystem basiert auf dem XA Standard (nur flat transactions)
- Transaktionen werden vom Container gesteuert werden (container managed transactions)
  - Transaktionseigenschaften werden dann beim Deployment der Bean festgelegt
- Transaktionen können bei Session Beans vom Bean selbst gesteuert werden (bean managed transactions)

Transaction Handling II

- Session Bean hat über den SessionContext Zugriff auf die Transaktion
  - eine Transaktion kann nur gestartet werden, wenn die vorherige beendet wurde (flat transactions)
  - ein stateless SessionBean muss die Transaktion beenden bevor der Aufruf beendet wird
  - ein statefull SessionBean kann eine Transaktion über mehrere Aufrufe halten
- Client hat über JNDI ebenfalls Zugriff auf die Transaktion

Transaction Handling III

- Container- managed transaction
  - Der Container kontrolliert die Transaktionen für das EJB
  - Wie der Container die Transaktionen für EJB behandeln soll, wird pro Methode eingestellt (transaction- attribute)
- Die Attribute werden im Deployment Descriptor eingestellt
- Nicht jede Methode eines EJB wird mit einem gültigen Transaktionskontext aufgerufen
- Kein Zugriff auf getUserTransaction()

Transaction Handling IV

- Transaktionsattribute
  - Not supported
    - die Methode wird ohne gültigen Transaktionskontext aufgerufen
  - Required
    - die Methode wird mit einem Transaktion Kontext aufgerufen. Ist der Client mit keinem Transsktion Kontext assoziiert, erzeugt der Server eine neue Transaktion, und beendet diese, nachdem die Methode beendet wurde.
  - Supports
    - ist der Client mit einem Kontext assoziiert, wird dieser für den Aufruf verwendet. Andernfalls wird die Methode ohne gültigen Kontext aufgerufen.

Transaktion Handling V

- Transaktionsattribute (Fortsetzung)
  - RequiresNew
    - die Methode wird immer in einer neuen Transaktion aufgerufen, die nach Verlassen der Methode beendet wird.
  - Mandatory
    - der Client muss einen gültigen Transaktionskontext zur Verfügung stellen, ansonsten signalisiert der Server einen Fehler.
  - Never
    - die Methode wird ohne gültigen Transaktionskontext aufgerufen. Stellt der Client einen solchen Kontext zur Verfügung, signalisiert der Server einen Fehler.

Transaktion Handling VI

- EJBs können Transaktion mit setRollbackOnly() als "rollback only" markieren
  - Wann die Transaktion tatsächlich zurückgesetzt wird, hängt davon ab, wer die Transaktion erzeugt hat
  - Nicht aufrufbar für Methoden mit transaction- attribute Never, Not Supported oder Supports
- SessionBeans können sich für transaktions-relevante Callbacks registrieren
  - Nur für transaction- attribute Required, RequiresNew und Manatory möglich
  - Implementierung des javax.ejb.SessionSynchronization Interfaces (afterBegin, afterCompletion, beforeCompletion)

Transaction Handling VII

- Session Bean mit container-managed transaction demarcation kann das SessionSynchronization Interface implementieren
  - Session Bean kann dadurch in den Lifecycle der Transaktion „eingreifen“
  - **`void afterBegin() throws RemoteException;`**
    - `Signals that a new transaction has begun.`
  - **`void beforeCompletion() throws RemoteException;`**
    - `Is called prior to committing the resource managers.`
  - **`void afterCompletion(boolean status) RemoteException;`**
    - `Signals that the current transaction has completed. False indicates that a rollback has occurred.`

Transaction Handling VIII

- Bean- managed transaction
  - Nur möglich für SessionBeans
  - Zugriff auf Transaktionen (javax.trasaction.UserTransaction) über Context Objekt
- Eine Transaktion kann nur gestartet werden, wenn die vorherige beendet wurde
- Ein stateless SessionBean muss die Transaktion beenden bevor der Aufruf beendet wird
- Ein statefull SessionBean kann eine Transaktion über mehrere Aufrufe halten

Transaction Handling IX

- **public interface UserTransaction**
- **void begin() throws NotSupportedException, SystemException;**
  - **Create a transaction and associate it with the current thread.**
- **void commit() throws RollbackException, HeuristicMixedException, HeuristicRollbackException, SecurityException, IllegalStateException, SystemException;**
  - **Complete the transaction associated with the current thread.**
- **void rollback() throws IllegalStateException, SecurityException, SystemException;**
  - **Rollback the transaction associated with the current thread.**
- **void setRollbackOnly() throws IllegalStateException, SystemException;**
  - **Mark the transaction for rollback.**
- **int getStatus() throws SystemException;**
  - **Get the status of the transaction. See javax.transaction.Status.**
- **void setTransactionTimeout(int secs) throws SystemException;**
  - **Modify the timeout value that is associated with the transactions started by the current thread using the begin method.**

Transaction Handling X

- Beispiel zu Bean- managed transaction

```
import javax.ejb.*;
public class CreateNewCustomer implements SessionBean {
    SessionContext context;
    ...
    public void createCustomer() throws EJBException{
        try {
            context.getUserTransaction().begin();        // Start a new transation
            // JDBC Calls go here
        } catch (Exception e) {
            throw new EJBException(e);
        }
    }
    ...
```

Transaction Handling XI

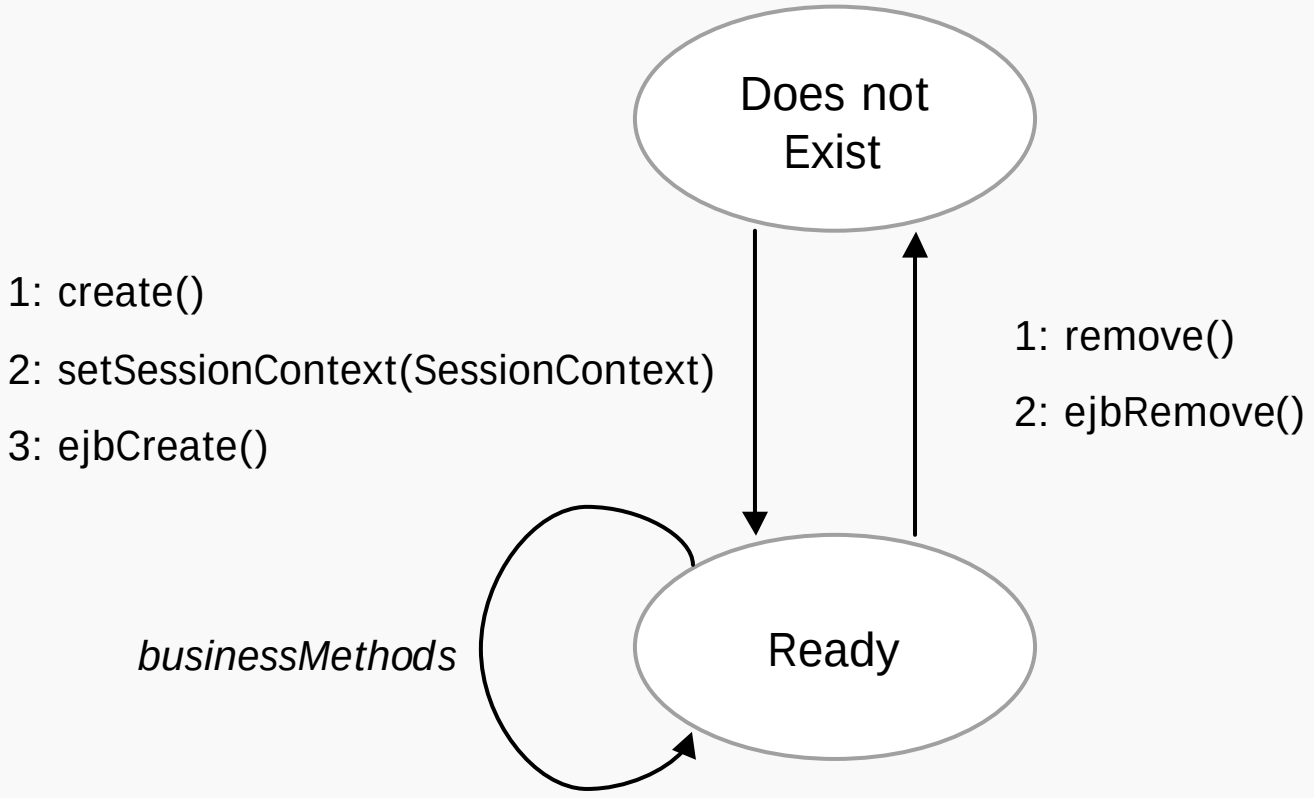
- Beispiel zu Bean- managed transaction (Forts.)

```
    ...
    public void saveCustomer() throws EJBException{
        try {
          // JDBC Calls go here
          context.getUserTransaction().commit();      // Commit the transation
        } catch (Exception e) {
          throw new EJBException(e);
        }
    }
    ...
}
```

Transaktion Handling XII

- Transaction Isolation Levels
  - Read Uncommited, Read Commited, Repeatable Read, Serializable
- EJB bietet kein API zur Einstellung des Isolation-Levels
  - Grund: Isolation-Levels sind Ressource-Manager spezifisch
- Einstellung möglich z.B. über
  - java.sql.Connection.setTransactionIsolation(...)

Session Bean XII

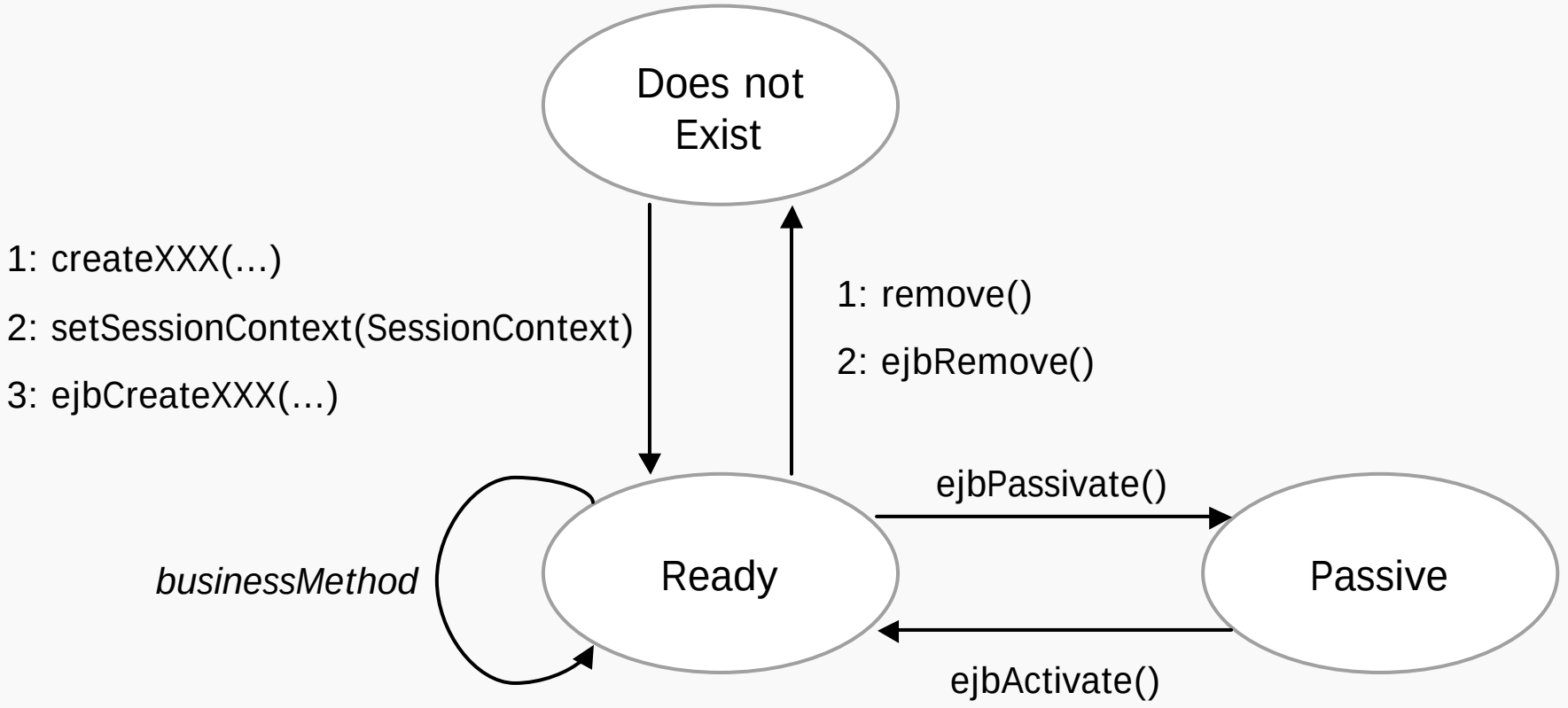


Life Cycle einer Stateless Session Bean

Session Bean XIII

Bean Methode	Erlaubte Operationen in einer Bean Methode	
	Container-managed transaction Demarcation	**Bean-managed transaction Demarcation**
Constructor	-	-
setSessionContext	SessionContext methods: *getEJBHome* JNDI access to java:comp/env	SessionContext methods: *getEJBHome* JNDI access to java:comp/env
ejbCreate ejbRemove	SessionContext methods: *getEJBHome*, *getEJBObject* JNDI access to java:comp/env	SessionContext methods: *getEJBHome*, *getEJBObject, getUserTransaction* UserTransaction methods JNDI access to java:comp/env
Business Method	SessionContext methods: *getEJBHome*, *getCallerIdentity*, *getRollbackOnly, isCallerInRole*, *setRollbackOnly*, *getEJBObject* JNDI access to java:comp/env Resource manager access Enterprise bean access	SessionContext methods: *getEJBHome*, *getCallerIdentity*, *isCallerInRole*, *getEJBObject, getUserTransaction* UserTransaction methods JNDI access to java:comp/env Resource manager access Enterprise bean access

Session Bean XIV



Life Cycle einer Stateful Session Bean

Session Bean XV

Bean method	Erlaubte Operationen in einer Bean Methode	
	Container-managed transaction demarcation	**Bean-managed transaction demarcation**
Constructor	-	-
setSessionContext	SessionContext methods: *getEJBHome* JNDI access to java:comp/env	SessionContext methods: *getEJBHome* JNDI access to java:comp/env
ejbCreate ejbRemove ejbActivate ejbPassivate	SessionContext methods: *getEJBHome*, *getEJBObject* JNDI access to java:comp/env Resource manager access Enterprise bean access	SessionContext methods: *getEJBHome*, *getEJBObject, getUserTransaction* UserTransaction methods JNDI access to java:comp/env Resource manager access Enterprise bean access
Business method	SessionContext methods: *getEJBHome*, *getCallerIdentity*, *getRollbackOnly, isCallerInRole*, *setRollbackOnly*, *getEJBObject* JNDI access to java:comp/env Resource manager access Enterprise bean access	SessionContext methods: *getEJBHome*, *getCallerIdentity*, *isCallerInRole*, *getEJBObject, getUserTransaction* UserTransaction methods JNDI access to java:comp/env Resource manager access Enterprise bean access

Session Bean XVI

Bean method	Erlaubte Operationen in einer Bean Methode	
	Container-managed transaction Demarcation	**Bean-managed transaction Demarcation**
afterBegin beforeCompletion	SessionContext methods: *getEJBHome*, *getCallerIdentity*, *getRollbackOnly, isCallerInRole*, *setRollbackOnly*, *getEJBObject* JNDI access to java:comp/env Resource manager access Enterprise bean access	Session Bean with bean-managed transaction demarcation must not implement the SessionSynchronization interface
afterCompletion	SessionContext methods: *getEJBHome*, *getCallerIdentity, isCallerInRole*, *getEJBObject* JNDI access to java:comp/env	

Exception Handling I

- System Exceptions
  - Client werden damit Systemfehler mitgeteilt
  - Führen zum Rollback der Transaktion
  - Müssen von der Bean nicht behandelt werden
  - Wenn in einer Bean Methode eine Exception auftritt (die z.B. nicht behandelt werden kann) soll eine EJBException signalisiert werden

Exception Handling II

- Application Exceptions
  - Client werden damit Anwendungsfehler mitgeteilt
  - Führen nicht unbedingt zum Rollback der Transaktion
  - Bean Provider ist für Konsistenz des Zustands verantwortlich
  - Dürfen weder Subklasse von RuntimeException noch java.rmi.RemoteException sein

Exception Handling III

Methode	Exception	Grund
ejbCreate	CreateException	Ein Input Parameter ist ungültig
ejbRemove	RemoveException	Die Bean konnte nicht gelöscht werden
Any Method	EJBException	Ein Systemfehler ist aufgetreten
Business Method	Application Exception	Ein Anwendungsfehler ist aufgetreten

Naming Conventions I

Item	Syntax	Example
Enterprise bean name (DD)	<name>EJB	CurrencyConverterEJB
EJB JAR display name (DD)	<name>JAR	CurrencyConverterJAR
Enterprise bean class	<name>Bean	CurrencyConverterBean
Home interface	<name>Home	CurrencyConverterHome
Remote interface	<name>	CurrencyConverter

Beispiel: Currency Converter I

- Soll einen Geldbetrag von einer Währung in eine andere Währung konvertieren können.
  - Wenn eine Währung nicht bekannt ist, soll eine Exception erzeugt werden.
- Soll eine Liste aller bekannten Währungen liefern.
- Zusätzlich zu einem „einfachen“ Client soll ein http-Zugriff über Servlet ermöglicht werden.

Beispiel: Currency Converter II

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface CurrencyConverterHome extends EJBHome {
   CurrencyConverter create()
       throws CreateException, RemoteException;
}
```

Beispiel: Currency Converter III

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import java.util.Collection;
public interface CurrencyConverter extends EJBObject {
  double convert(double monetaryAmount,
    String fromCurrency,String toCurrency)
      throws RemoteException, UnknownCurrencyException;
  Collection getCurrencies() throws RemoteException;
}
```

Beispiel: Currency Converter IV

```
public class CurrencyConverterBean implements SessionBean {
public void ejbCreate() {
   trace("CurrencyConverterBean.ejbCreate()");
}
public void setSessionContext(SessionContext ctx) {
   trace("CurrencyConverterBean.
   setSessionContext(SessionContext)");
}
public void ejbRemove() {
   trace("CurrencyConverterBean.ejbRemove()");
}
public void ejbActivate() {
   trace("CurrencyConverterBean.ejbActivate()");
}
public void ejbPassivate() {
   trace("CurrencyConverterBean.ejbPassivate()");
}
```

Beispiel: Currency Converter V

```
public double convert(double amount, String fromCurrency,
   String toCurrency) throws UnknownCurrencyException {
   trace(„CurrencyConverterBean.convert(double, String,
   String)");
   return convertFromEuros(convertToEuros(amount,
   fromCurrency), toCurrency);
}
protected double convertFromEuros(double amount, String
   currency) throws UnknownCurrencyException {
   return amount * getRate(currency);
}
protected double convertToEuros(double amount, String
   currency) throws UnknownCurrencyException {
   return amount / getRate(currency);
}
```

Beispiel: Currency Converter VI

```
protected double getRate(String currency) throws
   UnknownCurrencyException {
   Double rate = (Double) getRates().get(currency);
   if (rate == null) {
        throw new UnknownCurrencyException(currency);
   }
   return rate.doubleValue();
}
protected HashMap getRates() { ... }
public Collection getCurrencies() {
   trace(„CurrencyConverterBean.getCurrencies()");
   return new ArrayList(getRates().keySet());
}
protected static void trace(String message) {
   System.out.println(message);
}
}
```

Beispiel: Currency Converter VII

```
// add import statements
public class CurrencyConverterClient {
  public static void main(String[] args) {
    double amount = Double.parseDouble(arg[0]);
    String fromCurrency = arg[1];
    String toCurrency = arg[2];
    try {
      // see next slide
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

Beispiel: Currency Converter VIII

```
Properties env = new Properties();
env.put(Context.PROVIDER_URL, "... ");
env.put(Context.INITIAL_CONTEXT_FACTORY, "... ");
IntialContext ic = new InitialContext(properties);
Object ref = ic.lookup("ConverterHome");
CurrencyConverterHome home =
  (CurrencyConverter)
    PortableRemoteObject.narrow(
      home, CurrencyConverterHome.class);
CurrencyConverter bean = home.create();
System.out.print(fromCurrency + " " + amount + " = ");
System.out.print(
  bean.convert(amount, fromCurrency, toCurrency));
System.out.println(" " + toCurrency);
bean.remove();
```

Beispiel: Currency Converter IX

```
import java.io.*;
import java.rmi.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;
import javax.rmi.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ConverterServlet extends HttpServlet {
  protected CurrencyConverter converter;
  ...
}
```

Beispiel: Currency Converter X

```
public void init(ServletConfig config) throws
   ServletException {
   try {
       InitialContext ctx = new InitialContext();
       Object obj = ctx.lookup("CurrencyHome");
       CurrencyConverterHome home =
         (CurrencyConverterHome)
           PortableRemoteObject.narrow(obj,
             CurrencyConverterHome.class);
       converter = home.create();
   } catch (Exception ex) {
       ex.printStackTrace();
       throw new ServletException(ex.getMessage());
   }
}
```

Beispiel: Currency Converter XI

```
public void destroy() {
   if (converter != null) {
       try {
             converter.remove();
             converter = null;
       } catch (Exception ex) {
             ex.printStackTrace();
       }
   }
}
```

Beispiel: Currency Converter XII

```
public void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.print("<HTML><HEAD><TITLE>...</TITLE></HEAD><BODY>");
   double fromAmount =
     Double.parseDouble(request.getParameter("FROMAMOUNT"));
   String fromCurrency = request.getParameter("FROMCURRENCY");
   String toCurrency = request.getParameter("TOCURRENCY");
   double toAmount = converter.convert(fromAmount, fromCurrency,
   toCurrency);
   out.print("<H1>Conversion</H1></P>");
   out.print(fromCurrency + " " + fromAmount + " = ");
   out.print(toCurrency + " " + toAmount);
   out.println("</BODY></HTML>");
   out.close();
}
```

Session Bean XVII

- Auffinden des Home Interface

```
Context ic = new InitialContext();
Object obj = ic.lookup(“CurrentDate”);
CurrentDateHome home = (CurrentDateHome)
  PortableRemoteObject.narrow(obj,
  CurrentDateHome.class);
```

A1: Der Cast-Operator anstelle von PortableObject.narrow kann bei einem Container mit RMI-IIOP einen Fehler verursachen!

Java Naming & Directory Service I

- zur Zuordnung von Namen und Attributen zu Objekten
- die Namen der Objekte werden folgender-maßen festgelegt
  - für EnterpriseBeans während des Deployments
  - für Ressourcen (z.B. JDBC) vom Administrator
  - für spezielle Server-Objekte (z.B. UserTransaction) durch den EJB Server
- der Lookup auf des Home Interface erfolgt über einen InitialContext mittels lookup(„XXX“)

Java Naming & Directory Service II

- ein InitialContext wird folgendermaßen erzeugt
  - im EJB Server durch den Default Constructor

```
InitialContext ic = new InitialContext();
```

  - im Client können die Properties des JNDI Service Providers erforderlich (abh. vom Deployment)
    - Properties File – jndi.properties
    - Bzw. beim Erzeugen des InitialContext als Argumente

```
Properties env = new Properties();
env.put(Context.PROVIDER_URL, „...“);
env.put(Context.INITIAL_CONTEXT_FACTORY, „...“);
IntialContext ic = new InitialContext(properties);
```

- bez. Context und InitialContext siehe javax.naming

Session Bean XVIII

- Erzeugen einer Session Bean

```
CurrentDate session = home.create();
```

A1: Mögliche Exceptions sind RemoteException und CreateException

A2: Die Session Bean muss eine ejbCreate() Methode implementieren!

A3: Es können auch create-Methoden mit Argumenten definiert werden. Die Session Bean muss dann entsprechende ejbCreate-Methoden implementieren!

Session Bean XIX

- Verwenden einer Session Bean

```
Date currentDate = session.getCurrentDate();
```

A1: Mögliche Exceptions sind RemoteException und EJBException

A2: Der Return-Type java.util.Date muss serialisierbar sein!

Session Bean XX

- „Löschen“ einer Session Bean

```
session.remove();
```

oder

```
home.remove(session.getHandle());
```

A1: Mögliche Exceptions sind RemoteException und RemoveException
A2: Die Session Bean muss eine ejbRemove() Methode implementieren!
A3: In der ejbRemove-Methode sollten alle von der Session Bean verwendeten Ressourcen freigegeben werden!

Deployment I

- Während des Deployments werden
  - Bean Klasse, Home und Remote Interface festgelegt
  - Bean Typ (Session, Entity, Message-Driven) festgelegt
  - Transaktions- und Security-Eigenschaften festgelegt
  - Referenzen auf andere EJBs aufgelöst
  - Umgebungsinformationen für die Beans spezifiziert
  - Referenzen von ResourceManager aufgelöst
  - Implementierungen der Interfaces generiert

Deployment II

- Die Deployment-Information wird in einer XML Datei gespeichert
  - <!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
  - <!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>
- Das Packaging/Deployment kann ein mehrstufiger Vorgang sein
  - kann z.B. teilweise vom Bean Entwickler und vom Application Assembler durchgeführt werden

Deployment III

- Der Bean Developer legt beim Deployment fest
  - Allgemein
    - Bean Name (entspricht nicht JNDI Namen)
    - Bean Klasse, Home und Remote Interface
    - Bean Typ (Session, Entity, Message-Driven)
    - EJB und Resource Manager Referenzen
    - Security role Referenzen
    - Environment entries
  - Session Bean
    - State Management Type (Stateful/Stateless)
    - Transaction demarcation Typ
  - Entity Bean
    - Persistence management (CMP/BMP)
    - Container-managed fields und Primary key Klasse

Deployment IV

- Der Application Assembler ändert u.U. beim Deployment
  - Enterprise Bean Name
  - Environment entries
  - Description fields
- Der Application Assembler legt beim Deployment fest
  - Binding von Enterprise Bean Referenzen
  - Security roles
  - Method permissions
  - Linking von security role Referenzen
  - Transaction attributes

Deployment V

- Die Environment Informationen ermöglichen es, das EJB ohne Codeänderungen an die Laufzeitumgebung anzupassen
- Environment Informationen entsprechen im wesentlichen den Java Property Dateien
- Eine Bean hat Zugriff auf die Informationen durch JNDI über den Namen java:comp/env

```
Context ic = new InitialContext();
Context env = ic.lookup("java:comp/env");
String value = env.lookup(„property");
```

Deployment VI

- Der Zugriff von einem EJB auf andere EJBs erfolgt über logische Namen (statt direkten Referenzen)
- Der Application Assembler kann während des Deployments die Implementierung des referenzierten EJBs bestimmen
- Ein EJB findet das referenzierte EJB im Subcontext java:comp/ejb

```
Context ic = new InitialContext();
Object home = ic.lookup("java:comp/ejb/OtherBean");
```

Deployment VII

- EJBs müssen Resourcen mittels Resource Manager erzeugen
  - Resource Manager werden vom Server zur Verfügung gestellt
    - z.B. javax.sql.DataSource statt java.sql.DriverManager
  - Die Resource Manager, die ein Bean verwendet werden während des Deployments logischen Namen zugeordnet
- EJBs finden den Resource Manger im Subcontext java:comp/<type>

```
InitialContext ic = new InitialContext();
DataSource ds = (DataSource) ic.lookup("java:comp/jdbc/TheDB");
con =  ds.getConnection();
```

Deployment VIII

- Deployment Descriptor am Beispiel CurrencyConverter

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ...>

<ejb-jar>
  <display-name>CurrencyConverterEJB</display-name>
  <enterprise-beans> ... </enterprise-beans>
  <assembly-descriptor>
    <method-permission> ... </method-permission>
    <container-transaction> ... </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

Deployment IX

```
<enterprise-beans>
   <session>
     <display-name>CurrencyConverterBean</display-name>
     <ejb-name>CurrencyConverterBean</ejb-name>
     <home>examples.ejb.CurrencyConverterHome</home>
     <remote>examples.ejb.CurrencyConverter</remote>
     <ejb-class>examples.ejb.CurrencyConverterBean</ejb-class>
     <session-type>Stateless</session-type>
     <transaction-type>Container</transaction-type>
     <security-identity>
       <description></description>
       <use-caller-identity></use-caller-identity>
     </security-identity>
     <resource-ref></resource-ref>
   </session>
 </enterprise-beans>
```

Deployment X

```
<assembly-descriptor>
  <method-permission>
    <unchecked />
    ...
    <method> ... </method>
    ...
  </method-permission>
  <container-transaction>
    <method> ... </method>
    <trans-attribute>Supports</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

Deployment X

```
<method-permission>
  <unchecked />
  ...
  <method>
    <ejb-name>CurrencyConverterBean</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>convert</method-name>
    <method-params>
      <method-param>double</method-param>
      <method-param>java.lang.String</method-param>
      <method-param>java.lang.String</method-param>
    </method-params>
  </method>
  ...
</method-permission>
```

Deployment XI

```
<container-transaction>
  <method>
    <ejb-name>CurrencyConverterBean</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>convert</method-name>
    <method-params>
      <method-param>double</method-param>
      <method-param>java.lang.String</method-param>
      <method-param>java.lang.String</method-param>
    </method-params>
  </method>
  <trans-attribute>Supports</trans-attribute>
</container-transaction>
```

silbergrau
Consulting & Software GmbH

Danke für die Aufmerksamkeit!

Silbergrau Consulting & Software GmbH
Dr. Andreas Erlach