EJBs Kurz und Gut

Eine kurze Einführung in die Java 2 Enterprise Beans

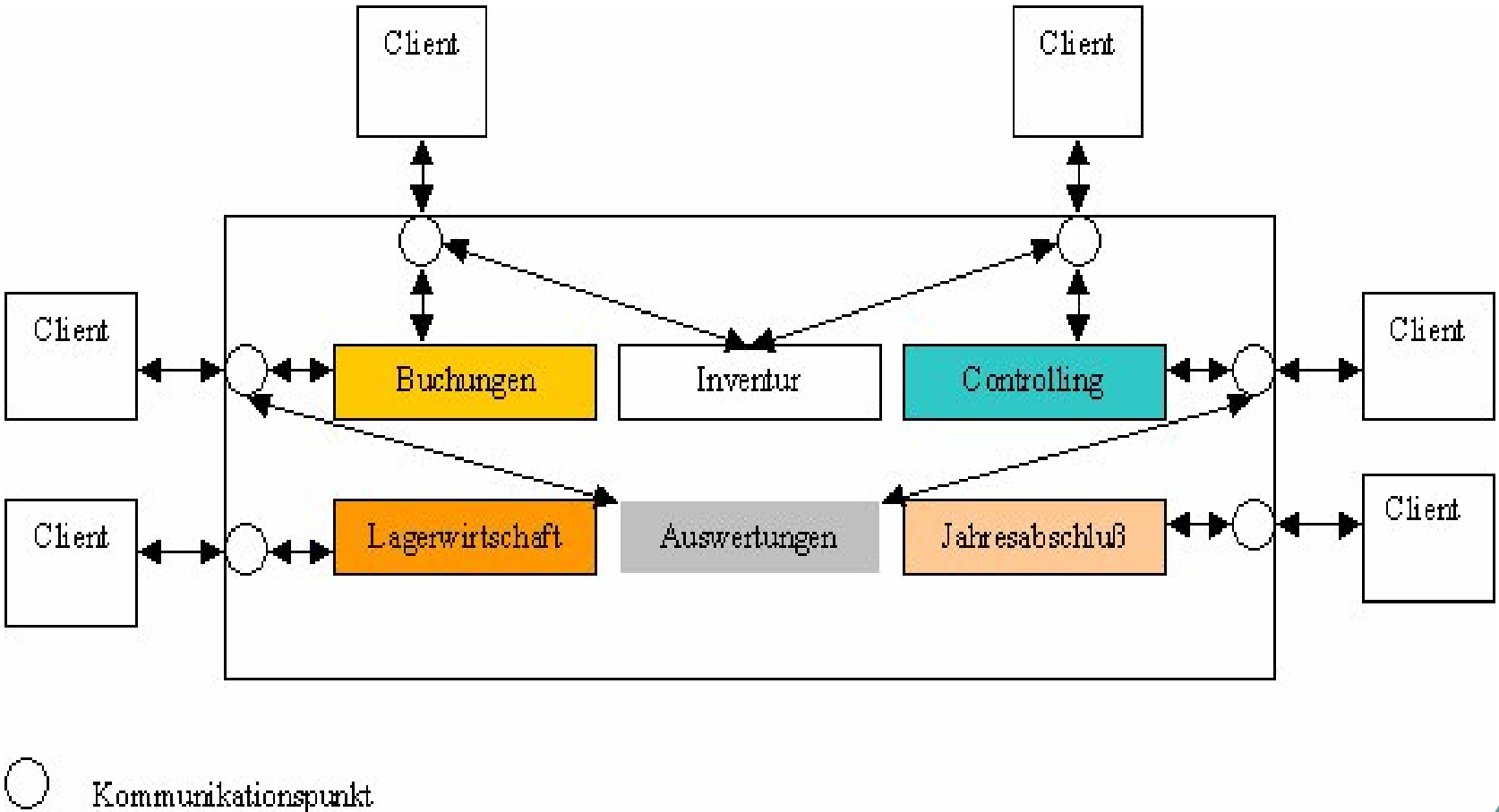
EJBs Einblick in die Thematik

- Durch den Druck der Wirtschaftsglobalisierung werden Unternehmen dazu gezwungen, ihre Geschäfte überregional und international zu expandieren. Dies geschieht unter anderem dadurch, daß ihre Standorte über mehrere Regionen und Länder verteilt werden müssen. Ein weiteres Problem, das hierdurch und durch den heftigen Konkurrenzkampf entsteht, ist das Verlangen nach EDV-Systemen , die den Business-Prozess erleichtern und beschleunigen sollen, mit höchster Flexibilität und Stabilität. Der bisherige Ansatz, an jedem Standort ein eigenes System im Einsatz ist, kann dieses Problem mit Sicherheit nicht lösen. Veränderungen in der EDV-Welt in Unternehmen müssen durchgeführt werden.
- Ein neues System für alle Standorte soll eingeführt werden, das folgende Anforderungen erfüllen muß:
- Mehrbenutzerfähigkeit
- Verfügbarkeit
- Skalierbarkeit
- Kommunikation mit der Außenwelt
- Integration mit anderen Anwendungen
- kurze Entwicklungszyklen
- Konfigurierbarkeit
- Datenablage
- schrittweise Migration
- leichte Erweiterbarkeit
- ***Einer von vielen Ansätzen zur Entwicklung dieses neuen Systems ist das Konzept von Enterprise JavaBeans***.

Was ist ein EJB überhaupt

- Enterprise JavaBeans ist eine Architektur für komponentenorientierte, verteilte Anwendungen [Sun Microsystems, 1999]. Dies bedeutet, daß Unternehmen selbst Komponenten entwickeln oder von Herstellern kaufen können. Diese serverseitigen Komponenten, Enterprise Beans, sind verteilte Objekte, die sich in Enterprise JavaBeans Containern befinden und Clients, ihre Services über ein Netzwerk bereitstellen.
- Buchhaltung, Inventur, Lagerwirtschaft, Controlling, Auswertung und Jahresabschluß sind die einzelnen Komponenten, die sich in einem Container auf einem Server befinden, stellen den Clients, die weltweit verstreut sein können, ihre Dienste zur Verfügung.

Eine Unternehmensstruktur



Komponenten basierte Struktur

- Die Einsatzgebiete und Ausprägungen einer Komponentenarchitektur sind sehr unterschiedlich. EJB stellt eine Komponentenarchitektur für verteilte, serverseitige und transaktionsorientierte Komponenten dar. Enterprise Beans sind Komponenten die, vielen Clients auf einem Server ihre Dienste anbieten. Diese Komponenten sind in eine Art von Laufzeitumgebung eingebettet, die ihnen notwendige Dienste zur Verfügung stellt. Ohne diesen Rahmen würde jede Komponente ein eigener Server sein. Dies bedeutet, daß die Entwicklung einer derartigen Komponente erschweren würde. Und der Einsatz mehrerer solchen Komponenten auf einem Rechner verursacht eine hohe Belastung von Ressourcen. Folgenden Anforderungen sind an eine Komponentenarchitektur gestellt:
- Unabhängigkeit der Umgebung: Unabhängig von Programmiersprachen, Betriebssystemen, Netztechnologien usw. sollten Softwarekomponenten eingesetzt werden bzw.. zusammenarbeiten können.
- Ortstransparenz: Für den Benutzer ist es unbedeutend, ob die Komponenten ihre Dienste im lokalen, im Adreßraum des anderen oder entfernten Servers anbieten. Die Mechanismen für die Nutzung von Komponenten müssen von der Architektur zur Verfügung gestellt werden.
- Trennung von der Schnittstelle und Implementierung: Vollständig unabhängig von der Implementierung sollte die Spezifikation einer Komponente erfolgen. Diese Anforderung wird durch das Konzept Interface in objektorientierten Sprachen unterstützt.
- Selbstbeschreibende Schnittstellen: Um dynamische (zur Laufzeit) Kopplung von Komponenten zu erreichen, sollte jede Komponente über ihre Fähigkeit Auskunft geben.
- Plug&Play: Jede Komponente sollte ohne jegliche Änderung auf jeder Plattform nutzbar sein.
- Integrations- und Kompositionsfähigkeit: Ein Komponente sollte in der Lager sein, sich mit anderen Komponenten zur Schaffung einer neuen gültigen Komponente zu kombinieren.

J2EE

- Die Java 2 Plattform, Enterprise Edition ist eine Mengen von Middleware-Dienste, die die Entwicklung von serverseitigen Applikationen erleichtern. Die J2EE Technologien beinhalten u.a.:
- Enterprise JavaBeans (EJB)

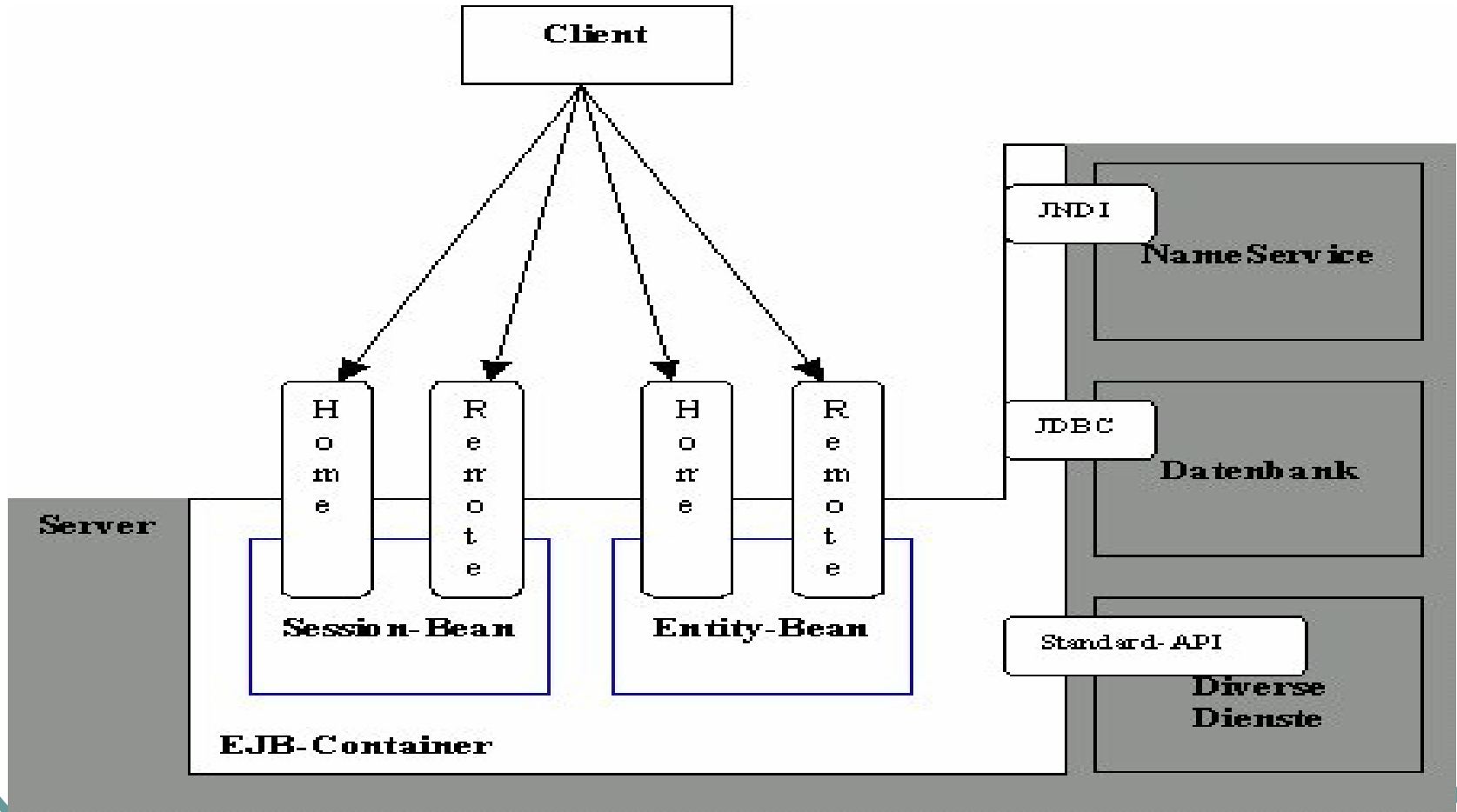
- Java Remote Method Invocation (RMI) und Remote Method Invocation via Internet Inter-ORB Protocol (RMI-IIOP): Für die verteilte Kommunikation zwischen Objekten wird RMI benutzt. Für die Kompatibilität mit CORBA kann auch RMI-IIOP eingesetzt werden.

- Java Naming and Directory Interface (JNDI): Hier handelt es sich um Namens- und Verzeichnisdienst. Ein Namendienst bietet zum einen die Möglichkeit, Referenz auf entfernte Objekte unter bestimmten Namen an einem definierten Platz zu hinterlegen (Binding). Zum anderen bietet er die Möglichkeit, gebundene Objekte über deren Namen wiederzufinden (Lookup).
  Ein Verzeichnisdienst unterstütz nicht nur das Binden von Objektreferenzen an einen Namen, sondern er kann auch andere Ressourcen wie z.B. Dateien, Drucker verwalten. Er bietet auch die Möglichkeit, diese Ressourcen zu verwalten.

- Java Database Connectivity (JDBC): Es erlaubt den Entwickler, über JDBC auf die Datenbank zuzugreifen.

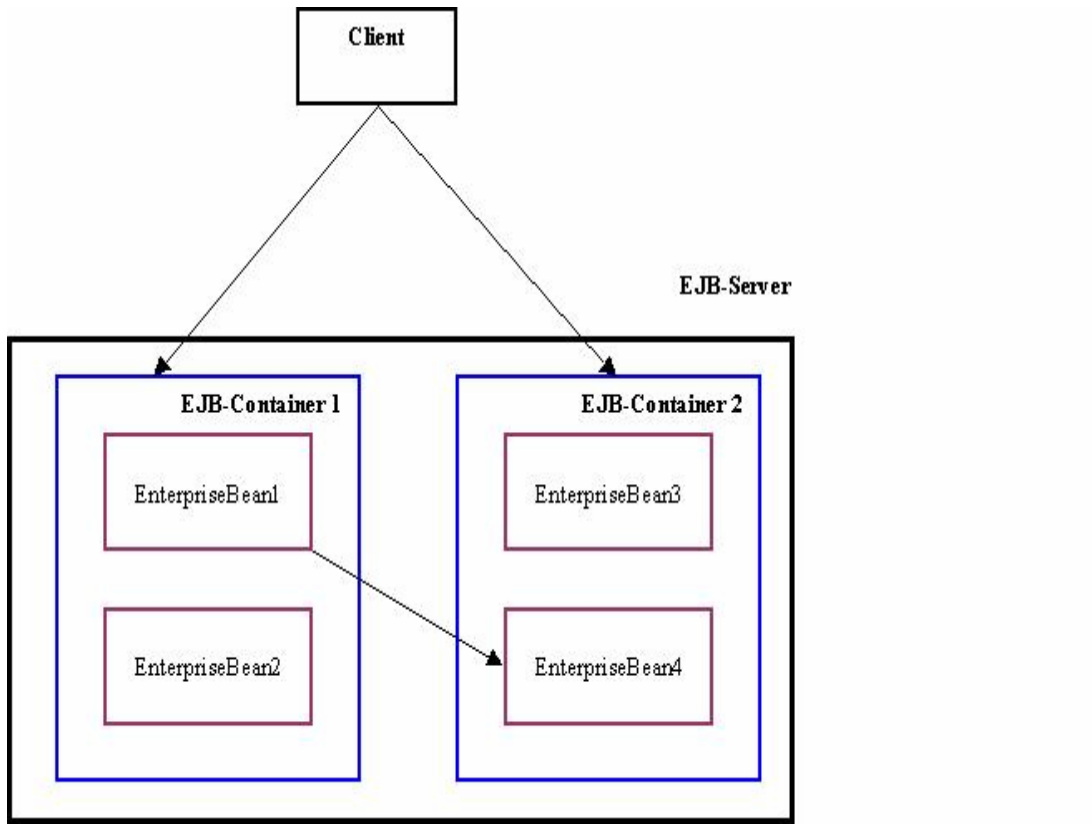
- Java Servelts und Java Server Pages (JSP): Servlets und JSPs sind z.B. geeignet für die Kommunikation mit Clients über HTTP.

EJB Architektur



Der EJB Server (z.B. OC4J)

Der Server ist die Basiskomponente der EJB-Architektur. Er ist eine Laufzeitumgebung für mehrere Container. Jeder Container stellt wiederum eine Laufzeitumgebung für bestimmte Enterprise Beans zur Verfügung. Es wird von Herstellern immer mehr dazu tendiert, Java-Applikationsserver zu entwickeln, die J2EE-Plattform zu unterstützen. Kaum Hersteller entwickeln reinen EJB-Server.



Der EJB Container

- Wie es bereits kurz dargestellt wurde, ist der EJB-Container eine Laufzeitumgebung für Enterprise Beans. Er stellt den Komponenten zur Laufzeit bestimmte Dienste über Standard-APIs zur Verfügung. Dies führt dazu, daß der Container dazu verpflichtet ist, Enterprise Beans mindestens folgenden APIs zugänglich zu machen:
- Das API des JDK 1.1.x oder das der Java-2-Plattform
- Das API der Spezifikation der Enterprise JavaBeans 1.1
- Das API des JNDI 1.2 (Java Naming and Directory Interface)
- Das UserTransaction-API aus JTA 1.0.1 (Java Transaction API)
- Das API der JDBC-2.0-Erweiterung (Java DataBase Connectivity)
- Das API von Java Mail 1.1 (für das Verschicken von E-Mails)
- Hersteller von Java-Applikationsserver können zusätzlich Dienste über Standard-APIs anbieten.

Laufzeitumgebung

- **Kontrolle des Lebenszyklus einer Enterprise Bean:** Der EJB-Container ist für die Kontrolle des Lebenszyklus einer Bean zuständig. Wenn Anfragen von Clients kommen, muß der EJB-Container die Bean-Instanzen angemessen erzeugen, zerstören bzw. wiederverwenden. Fordert beispielsweise ein Client nach einer Bean-Instanz, die noch nicht existiert, muß der Container sie neu erzeugen. Falls sie schon existiert, wird sie einfach an den Client zurückgeliefert. Der Container muß auch dafür sorgen, daß nicht mehr benötigte Bean-Instanzen gelöscht werden. Das Ganze wird als Instanzen-Pooling genannt.

- **Instanzen-Pooling und Aktivierung bzw. Passivierung:** Ein Anwendungssystem muß mit großer Belastung zurechtkommen. Es muß in der Lage sein, viele Clients bedienen zu können, ohne daß die lange Antwortzeiten entstehen. Je größer die Anzahl der Anfragen von Clients ist, desto höher ist die Anzahl der erzeugten Bean-Instanzen. Und damit zum einen die Anzahl von Bean-Instanzen nicht unkontrolliert wächst und zum anderen sie ständig wieder gelöscht werden müssen, hält der Container bestimmte Anzahl von Instanzen in einem Pool vorrätig. Solange sie sich im Pool befinden, sind sie im Zustand *Pooled* und deaktiviert. Bei Bedarf wird eine passende Bean aus dem Pool entnommen. Sie wird reaktiviert und befindet sich dann im Zustand *Ready*

- **Verteilung:** Der EJB-Container sorgt dafür, daß Clients Beans benutzen können. Es ist für den Client unerheblich , wo sich die Beans auf dem Server befinden. Der Ort, an dem die Beans existieren, ist für den transparent. Die Benutzung von Beans, die sich auf einem anderen Rechner, unterscheidet sich für den Client nicht wesentlich von der Benutzung von Objekten im gleich Adressraum. Für die Kommunikation zwischen Server und Clients wird Java RMI eingesetzt. Für die Kompatibilität mit CORBA kann auch RMI-IIOP benutzt werden. Der EJB-Container sorgt dafür, daß die Beans innerhalb des Containers von außen angesprochen werden können.

- **Transaktion (sowohl Laufzeitumgebung als auch Dienst):** Bei verteilten Anwendungen entstehen oft Fehlersituationen durch den gleichzeitigen Zugriff mehrerer Benutzer auf gemeinsame Daten. Um diese Fehlersituationen zu behandeln, werden Entwickler von Transaktionen unterstütz. Die auszuführenden Aktionen werden vom Entwickler in Transaktionen unterteilt. Der EJB-Container muß die Sorge dafür tragen, daß die einzelnen Aktionen einer Transaktion erfolgreich durchgeführt werden müssen. Falls ein Fehler auftritt, müssen alle bisher einzelnen erfolgreichdurchgeführten Aktionen wieder rückgängig gemacht werden. Die Unterstützung von Transaktionen ist ein wesentlicher Bestandteil der EJB-Spezifikation.

- **Sicherheit:** Jeder EJB-Container wird von der Spezifikation verpflichtet, eine Infrastruktur für das Sicherheitsmanagement als Bestandteil der Laufzeitumgebung zur Verfügung zu stellen. Zu dem Aufgabenbereich des jenigen, der die Beans installiert, gehört die Festlegung der Sicherheitspolitik. Für die Umsetzung dieser Sicherheitspolitik muß der EJB-Container die Sorge tragen. Die Sicherheitsmechanismen müssen für die Beans transparent sein, damit sie in möglichst vielen Systemen eingesetzt werden können. Deshalb soll im Code einer Enterprise-Bean möglichst keine Implementierung der Sicherheitsmechanismen vorhanden sein. Viel mehr sinnvoller ist die Verlagerung der Sicherheitsmechanismen in die Laufzeitumgebung der Komponenten. Zu den Sicherheitsmechanismen gehören die Rolle- und Rechteerteilung, die Authentifizierung des Benutzers durch eine Benutzerkennung und ein Paßwort und sichere Kommunikation z.B. durch den Einsatz von Secure-Socket-Layer.

Dienste

- **Namens- und Verzeichnisdienst:** Ein Client ist auf den Namensdienst angewiesen, damit er eine Bean finden kann. Ein Namendienst bietet zum einen die Möglichkeit, Referenzen auf entfernte Objekte unter bestimmten Namen an einem definierten Platz zu hinterlegen (Binding). Diese Namen können auch freigegeben werden. Zum anderen bietet er die Möglichkeit, die gebundene Objekte über deren Namen wiederzufinden (Lookup).

  Ein Verzeichnisdienst unterstütz nicht nur das Binden von Referenzen auf verteilte Objekte, sondern das Verwalten und Administrieren anderer Ressourcen wie z.B.: Drucker, Dateien, Anwendungsserver usw.

- **Persistenz:** Die Beans haben die vom Container gestellte Möglichkeit, über den Namens- und Verzeichnisdienst auf Datenbankverbindungen zuzugreifen. Damit können sie die Verantwortung dafür tragen, daß ihr Zustand persistent gemacht wird. Es gibt aber auch einen anderen Mechanismus, mit dessen Hilfe der Zustand von Beans automatisch persistent gemacht werden kann. Bei diesem Ansatz werden Daten von Beans durch den Container in der Regel in einer Datenbank. Als Ablageort muß es nicht unbedingt eine Datenbank sein. Es ist auch denkbar, daß es andere Container gibt, die die Daten in anderen Speichersystemen ablegen. Es besteht auch die Möglichkeit, Container zu entwickeln, die die Schnittstellen anderer Anwendungen benutzt, um dort Daten abzulegen oder auszulesen.

  Entscheidend ist es die Tatsache, daß es im Falle der automatischen Persistenz einer Enterprise-Bean für die Bean uninteressant ist, wo die Daten abgespeichert sind. Der Container muß dafür sorgen, daß die Daten abgespeichert werden und damit sie zu jedem Zeitpunkt einen konsistenten Zustand haben.

Enterprise Bean

- Eine Enterprise-Bean ist eine serverseitige Komponente, die die Anwendungslogik implementiert, auf die Clients zurückgreifen. Der EJB-Server und -Container sorgen nur dafür, daß die Beans von den Clients benutzt werden können. Enterprise-Beans werden in einem EJB-Container installiert, der ihnen bestimmte Dienste und Laufzeitumgebungen zur Verfügung stellt.

Typen von EJBs

- **Session-Beans:** Sie stellen Geschäftsprozesse dar. Sie implementieren Geschäftslogik, -regel und Workflow. Beispielsweise handelt es sich um das Anlegen eines neuen Kunden im Warenwirtschaftssystem, die Durchführung einer Buchung im Buchungssystem, Banktransaktionen oder komplexe Berechnungen. Session-Beans repräsentieren Tätigkeiten, die auf einem Server für Clients durchgeführt werden. Diese Betrachtungsweise durch die Tatsache deutlicher dargestellt, daß eine Session-Bean eine private Ressource eines bestimmten Clients ist. Session-Beans werden als Session-Beans bezeichnet, weil sie solange existieren wie die Sitzung von Clients, die diese Session-Beans benutzen. Beispielsweise wenn ein Client eine bestimmte Bean anfordert, ist der Server dafür zuständig, eine neue Instanz von dieser Bean zu erzeugen. Wird diese Instanz später von dem Client nicht benötigt, wird sie weggeworfen. Es wird unterschieden von zustandslosen und zustandsbehafteten Session-Beans.

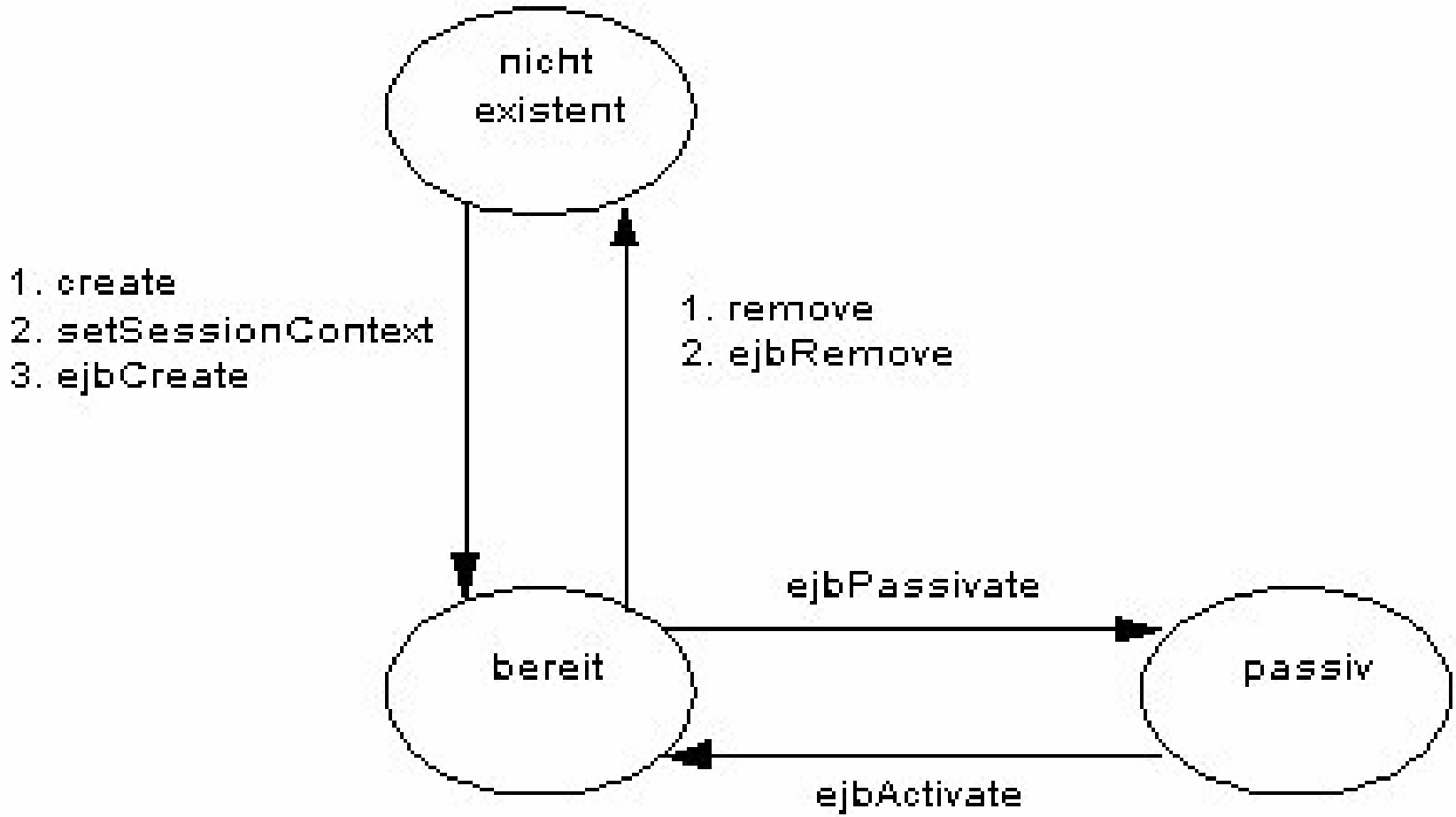
  Zustandslose Session-Beans speichern von einem Methodenaufruf zum nächsten keine Daten. Die Methoden einer zustandslosen Session-Bean arbeiten nur mit Daten, die als Parameter übergeben werden. Zustandslose Session-Beans des gleichen Typs besitzen alle die gleiche Identität. Da sie keinen Zustand besitzen, besteht es keine Notwendigkeit und Möglichkeit, sie zu unterscheiden.

  Zustandsbehaftete Session-Beans speichern dagegen Daten über mehrere Methodenaufrufe hinweg. Aufrufe von Methoden an den zustandsbehafteten Session-Beans können den Zustand einer Bean verändern. Der Zustand geht verloren, wenn die Sitzung vom Client beendet oder der Server heruntergefahren wird. Die zustandsbehaftete Session-Beans des gleichen Typs haben zur Laufzeit unterschiedliche Identitäten. Der EJB-Container muß sie unterscheiden können, weil sie für ihre Clients jeweils die unterschiedliche Zustände verwalten müssen.

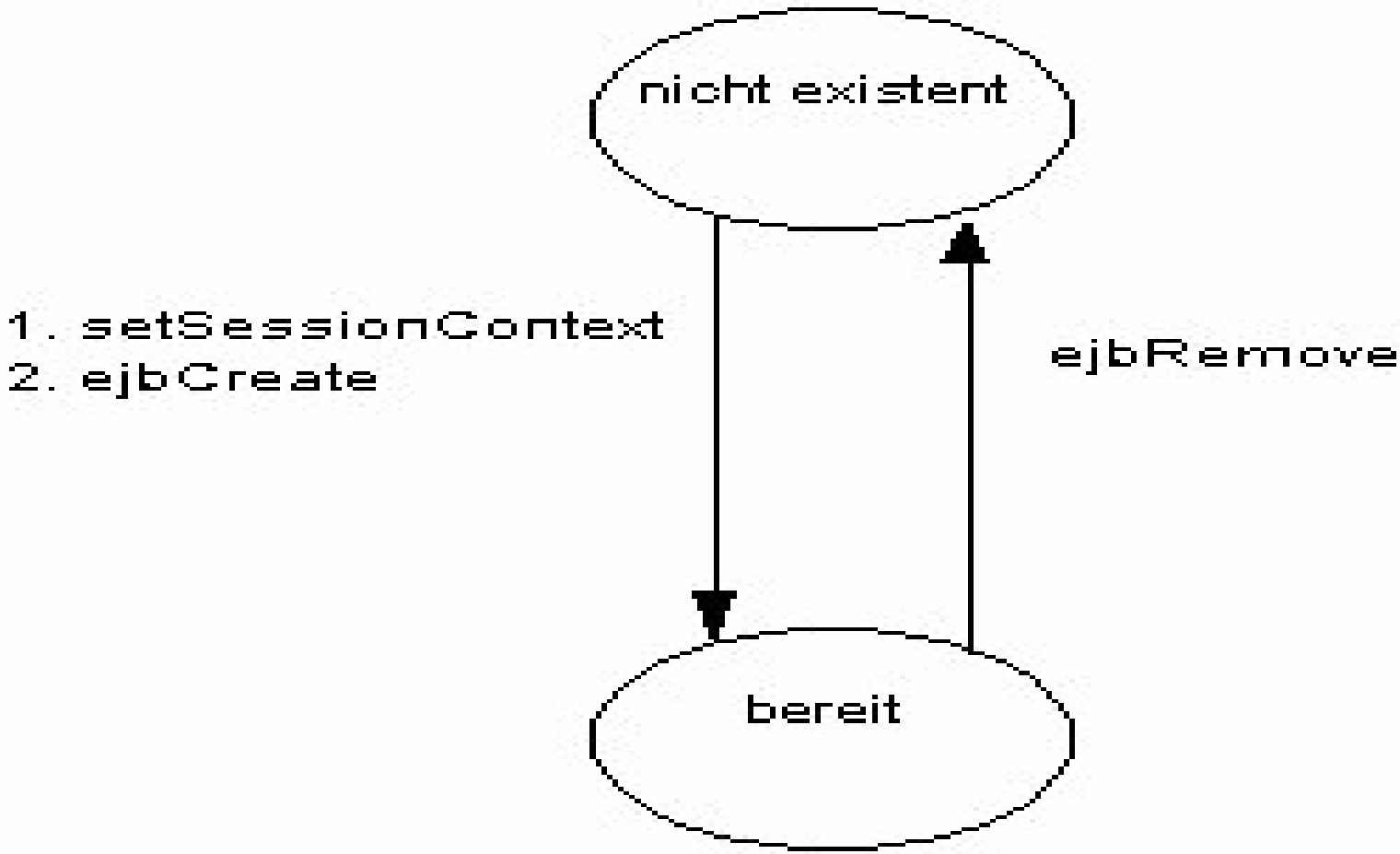
- **Entity-Beans:** Entity-Beans repräsentieren Dinge des realen Lebens, die mit bestimmten persistenten Daten assoziiert werden wie z.B. ein Kunde, ein Buchungskonto oder ein Produkt. Eine Instanz von einer Entity-Bean kann von mehreren Clients benutzt werden. Entity-Beans implementieren keine Geschäftsprozesse. Sie modellieren nur Daten und können von Session-Beans zum Repräsentieren von Daten benutzt werden, die sie benötigen.

  Entity-Beans lassen sich dadurch unterscheiden, ob sie selbst verantwortlich sind, daß ihre Daten persistent gemacht werden, oder ob der der EJB-Container diese Aufgabe übernehmen soll. In erstem Fall spricht man von bean-managed Persistence und in zweitem container-managed Persistence. Eine Entity-Bean des gleichen Typs wird zur Laufzeit durch ihren Primärschlüssel identifiziert, den sie vom Container zugeteilt bekommt. Dadurch wird sie an bestimmte Daten gebunden, die sie repräsentiert. Die Identität einer Entity-Bean wird nach außen sichtbar.

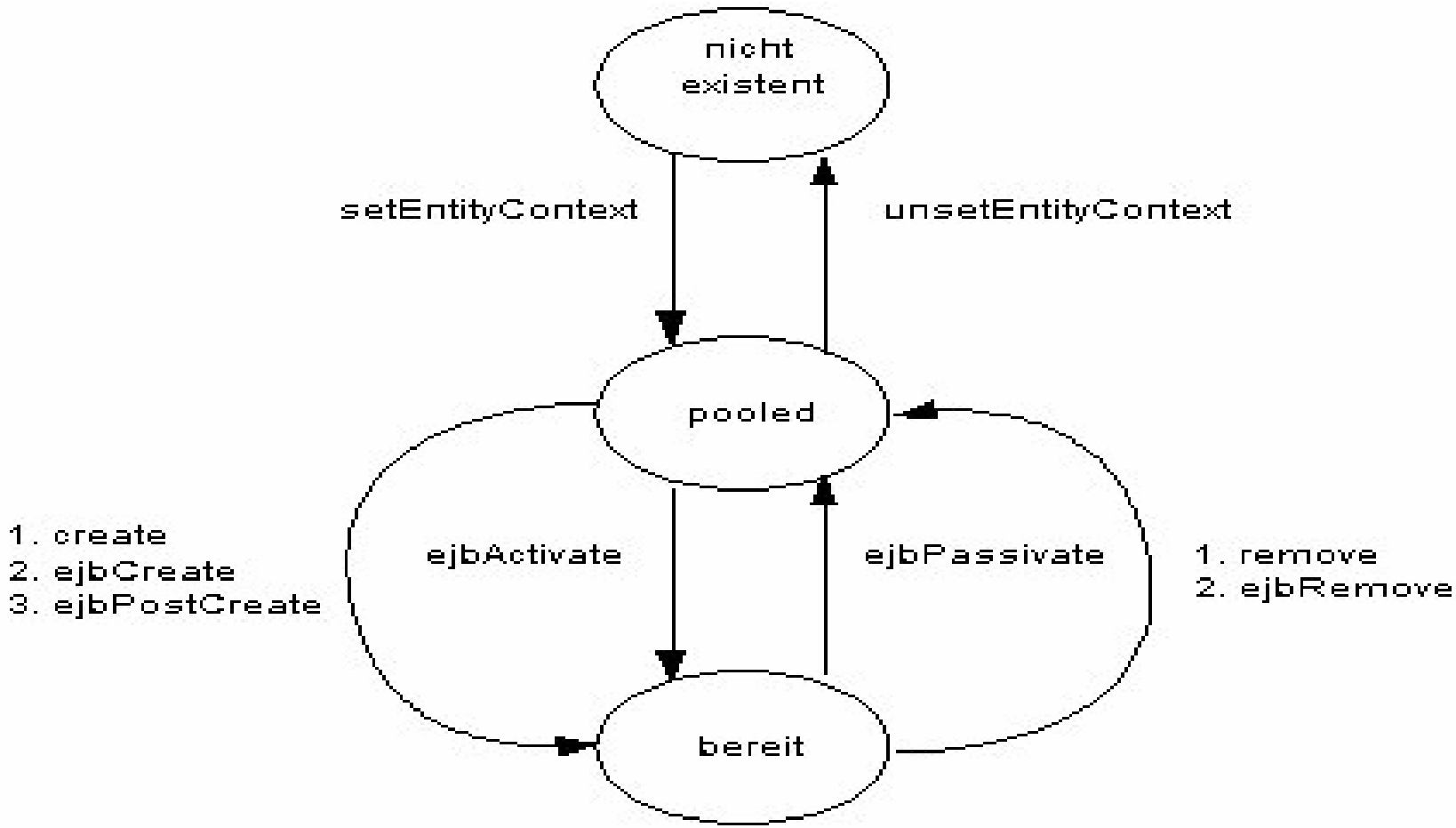
Lifecycle einer SessionBean



Lifecycle einer stateless SessionBean



LifeCycle einer EntityBean



Bestandteile einer Bean

- **Remote-Interface**
  Das Remote-Interface definiert diejenigen Methoden, die von einer Enterprise-Bean nach außen hin angeboten werden. Diese Methoden können von Clients aufgerufen werden. Das Remote-Interface muß von javax.ejb.EJBObject abgeleitet werden. EJBObject ist seinerseits wiederum von java.rmi.Remote abgeleitet.
-
- **Home-Interface**
  Im Home-Interface werden alle Methoden deklariert, die den Lebenszyklus einer Enterprise-Bean betreffen. Auch dieses Interface wird nach außen veröffentlicht.Clients benutzten es um Bean-Instanzen zu erzeugen, aktivieren, passivieren, zu finden oder zu löschen. Das Home-Interface muß von javax.ejb.EJBHome abgeleitet sein, das seinerseits wiederum von java.rmi.Remote abgeleitet ist.
- **Bean-Klasse**
  Die Bean-Klasse implementiert die Methoden, die im Home- und im Remote-Interface definiert wurden, ohne diese beiden Interfaces im Sinne des Schlüsselwortes **implements** tatsächlich einzubinden. Die Signaturen der im Home- und im Remote-Interface deklarierten Methoden müssen mit den entsprechenden Methoden in der Bean-Klasse übereinstimmen. Die Bean Klasse muß in Abhängigkeit von ihrem Typ ein Interface implementieren, und zwar **javax.ejb.EntityBean** oder **javax.ejb.SessionBean.**

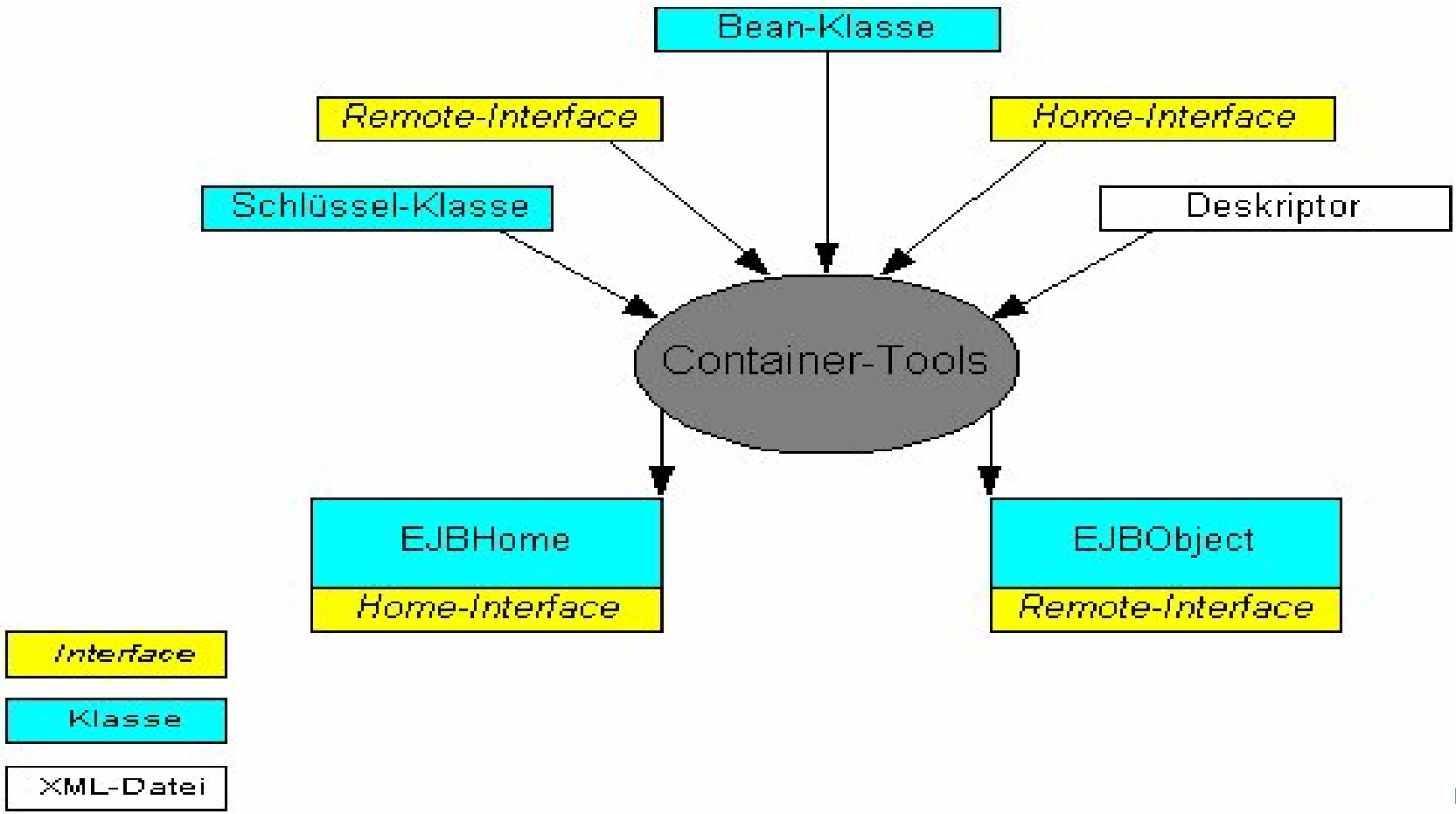
- **Der Primärschlüssel (Primärschlüsselklasse)**
  Er ist nur bei Entity-Beans relevant. Er dient dazu, eine Entität eines bestimmten Typs eindeutig zu identifizieren, die dann vom Container mit einer Entity-Bean-Instanz eines passenden Typs assoziiert wird. Über diesen Schlüssel wird die Identität einer Entity-Bean nach außen sichtbar. Diese Bean-Klasse muß das Interface **java.io.Serializable** implementieren.

- **Der Deployment-Deskriptor**
  Der Deployment-Deskriptor ist eine Datei im XML-Format und beschriebt eine oder mehrere Beans. In diesem Diskriptor befinden sich deklarative Informationen, die im Code einer Bean nicht zu finden sind. Diese Informationen sind für diejenigen Personen wichtig, die die Beans zu Anwendungen zusammenfassen bzw. sie in einem Container installieren. Über diesen Deployment-Deskriptor wird der Container mitgeteilt, wie er die Enterprise-Beans zur Laufzeit zu behandeln hat.
- Informationen, die sich im Deployment-Deskriptor befinden, sagen zum einen über die Struktur einer Bean und ihre externe Abhängigkeit aus wie z.B. zu anderen Beans oder zu Verbindungen zu einer Datenbank aus. Zum anderen sagen sie aus, wie sich eine Komponente zur Laufzeit verhalten soll bzw. wie sie mit anderen Beans zu komplexeren Bausteinen kombiniert werden kann.

Orchester der Beans

- Sind alle Bestandteile einer Bean vorhanden, so werden sie in eine Datei im jar-Format (Java-Archiv) verpackt. Damit sind die Bestandteile einer Enterprise-Bean komplett als Komponente in einer jar-Datei verpackt. Dann kann die Installation in einem EJB-Container mit Hilfe von entsprechenden Tools erfolgen. Wie diese Tools aussehen und wie sie zu bedienen sind, ist dem Container-Hersteller überlassen. Entscheidend ist jedoch, daß die Implementierung des Remote- und Home-Interfaces vom Container aus den Bestandteilen der Bean generiert wird.

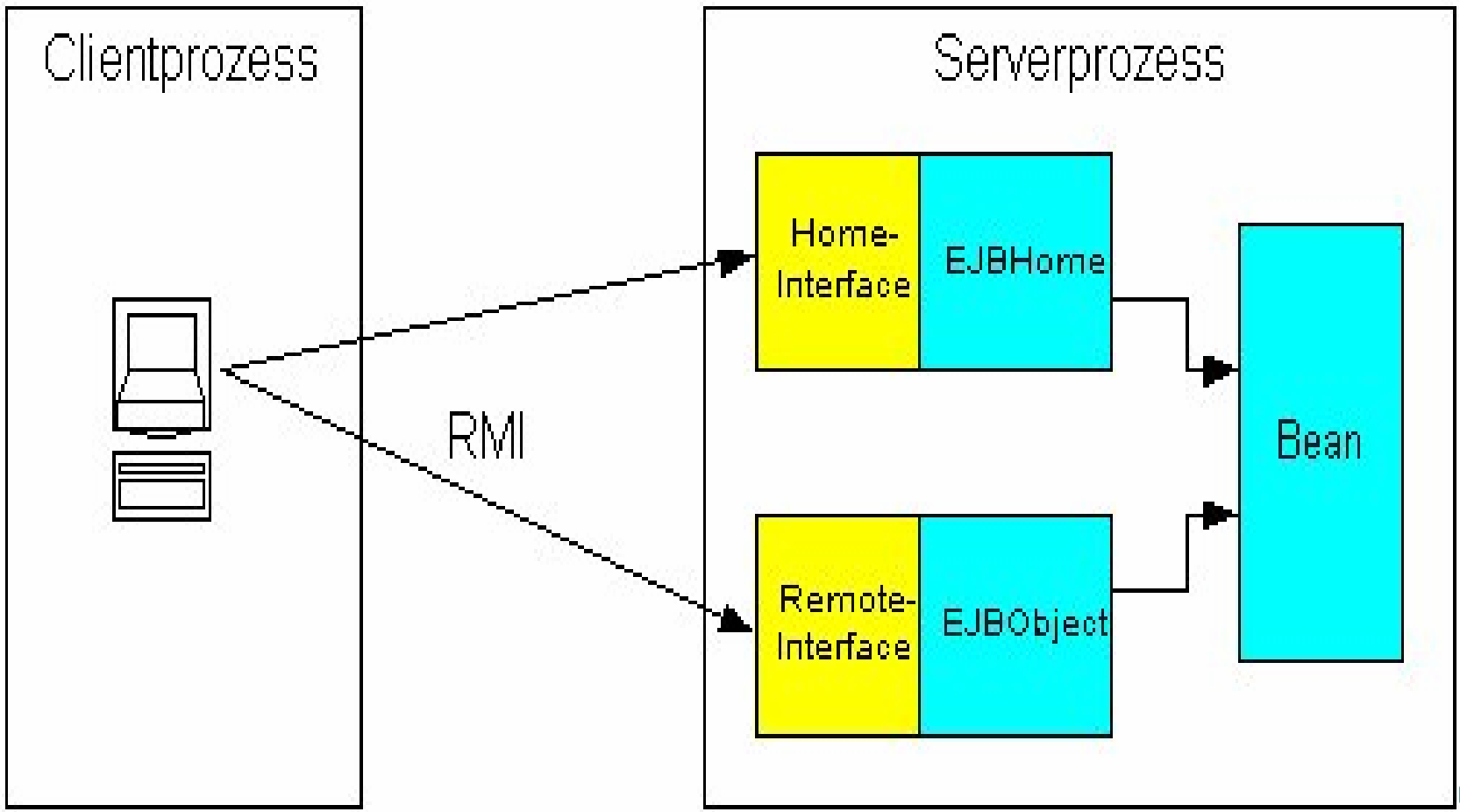
Orchester der Beans



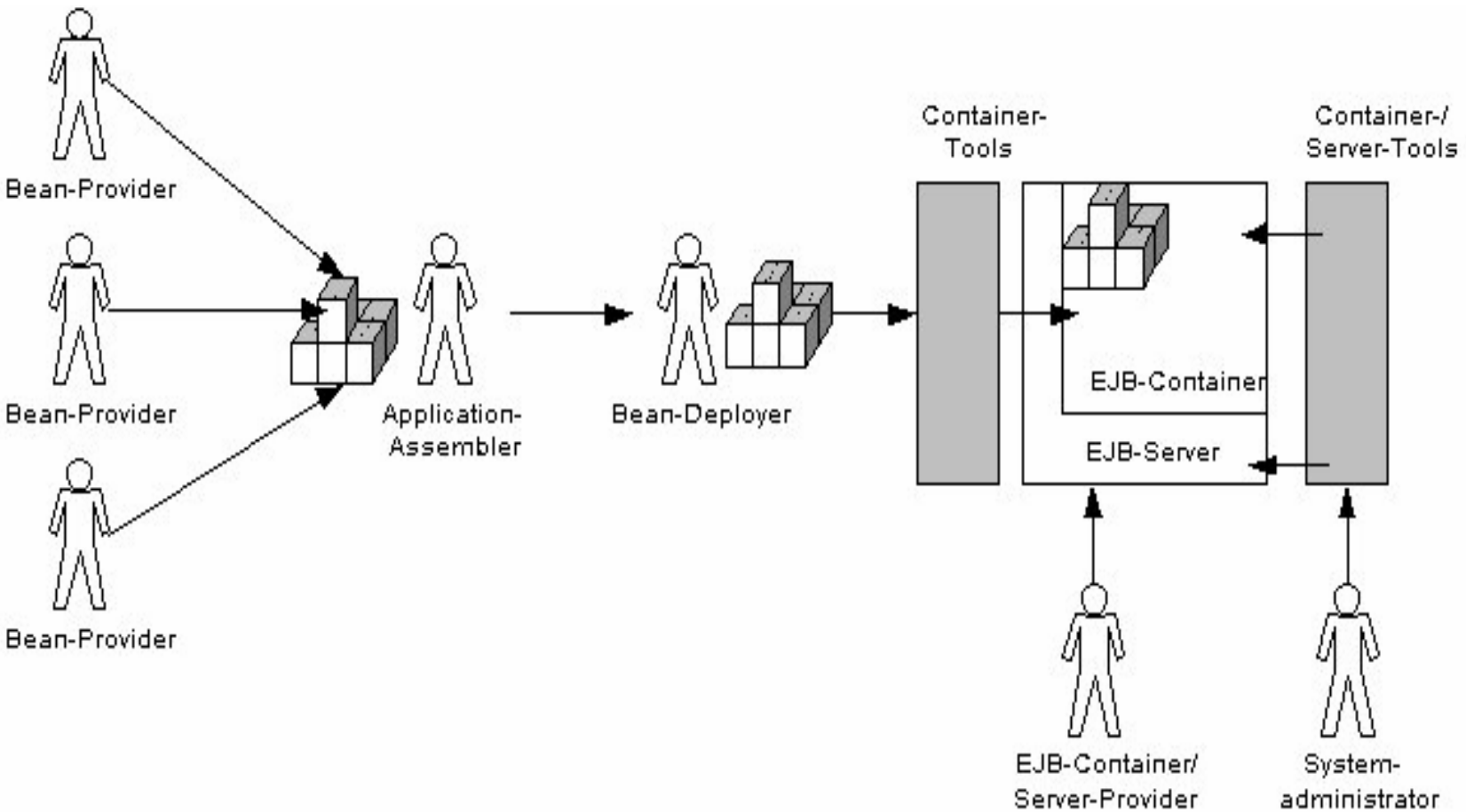
Orchester der Beans

- Die Implementierungsklassen des Remote- und Home-Interfaces heißen üblicherweise EJBObject und EJBHome. Sie sind der verlängerte Arm der Laufzeitumgebung des Containers für bestimmte Bean-Typen. Sie sind Remote-Objekte, an denen der Client die Methoden aufruft. EJBObject und EJBHome delegieren die entsprechenden Methoden an die Bean-Instanz. Die EJBHome-Klasse enthält den Code für das Erzeugen, Finden und das Löschen von Bean-Instanzen. Die EJBObject-Klasse implementiert das transaktionale Verhalten , die Sicherheitsüberprüfungen und ggf. die container-gesteuerte Persistenz.

Orchester der Beans



Die Rollen

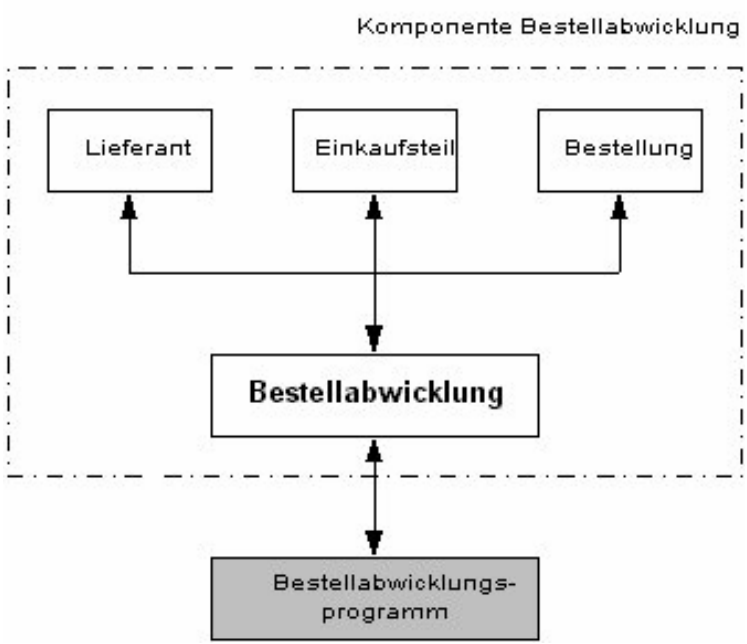

Bean-Provider
Bean-Provider
Bean-Provider
Application-
Assembler
Bean-Deployer
Container-
Tools
Container-/
Server-Tools
EJB-Container
EJB-Server
EJB-Container/
Server-Provider
System-
administrator

Rolle:Bean Provider

- Der Bean-Provider hat die Aufgabe, die eigentlich Geschäftsprozesse zu implementieren. Er verpackt die Anwendungslogik in Komponenten, durch die diese Anwendungslogik wiederverwendbar. Der Bean-Provider muss sich nur auf seine eigentlich Aufgabe konzentrieren und wird durch den EJB-Container von anderen Techniken wie z.B. Transaktionen, Multithreading Netzwerkanbindung usw. entlastet. Zusammen mit seiner Komponente liefert er einen Deployment-Deskriptor, in dem alle Informationen über seine Komponente selbst und ihre externe Abhängigkeit abgelegt sind. Diese Informationen werden benötigt, um die Komponente in einem Server zu installieren. Der Application-Assembler braucht sie auch, um Anwendungen aus verschiedenen Komponenten zusammenstellen zu können.

Rolle:Application Assembler

- Der Application-Assembler ist dafür verantwortlich, die zur Verfügung stehenden Funktionalitäten der im Container installierten Komponenten zu Anwendungen zusammenzustellen. Dazu gehört die Entwicklung von Client-Applikationen und die damit verbundene Steuerung der Kommunikation mit den Komponenten. Der Applikations-Assembler kann mehrere Komponente, die von Bean-Providern zur Verfügung gestellt haben, zu einer neuen Anwendung zusammenzufassen.



Rolle:Bean Deployer

- Der Bean-Deployer hat Aufgabe, die Komponenten in einem Container zu installieren. Dazu gehört die Bedienung des Tools des Container-Providers sowie die richtige Parametrisierung der zu installierenden Beans.

Roller:Container/Server Provider

- Der Container-Provider liefert den Komponenten eine komfortable Laufzeitumgebung. Er setzt auf den Schnittstellen des Server-Providers auf und hat die Sorge dafür zu tragen, daß der Zugriff auf die Beans ausschließlich über den Container erfolgt und ebenso die Kommunikation der Beans mit ihrer Umwelt.
- Der Server-Provider liefert einen Application-Server, der Komponenten enthält und verwaltet. Es gibt eigentlich kein Unterschied zwischen EJB-Container und EJB-Server. Es gibt einige EJB-Container-/Server-Produkte wie z.B. BEA´s Weblogic, Sun Microsystem´s NetDynamics, IBM´s WebSphere, Oracle´s Oracle 10g usw.

Rolle:Systemadministrator

- Der Systemadministrator überwacht den EJB-Server mit Hilfe der vom Hersteller gelieferten Tools zur Laufzeit und sorgt für die zum Betrieb eines Servers notwendige Infrastruktur.

Beispiel einer stateless Session Bean

- **Aufgabenstellung**
- Bei diesem Beispiel handelt es sich um eine zustandslose Session-Bean, die Clients vier Methoden zur Verfügung stellt, nämlich :
- int add(int a, int b)
- int sub(int a, int b)
- int mul(int a, int b)
- double div(int a, int b)
- Die Namen der Methoden lassen sich erkennen, daß sie den vier Grundoperationen (Addieren, Subtrahieren, Multiplizieren und Dividieren) entsprechen.

Das Remote Interface

- Das Remote-Interface definiert Methoden, die Clients zur Verfügung gestellt werden.
- import java.rmi.RemoteException;
  import java.lang.ArithmeticException;
  import javax.ejb.EJBObject;
- public interface Calculator extends EJBObject
  {
- public int add(int a, int b);
- public int sub(int a, int b);
- public int mul(int a, int b);
- public double div(int a, int b) throws ArithmeticException;
- }

Das Home Interface

- Im Home-Interface werden alle Methoden definiert, die den Lebenszyklus einer Bean betreffen. Dieses Interface wird nach außen hin veröffentlich. Der Client benutzt diese Schnittstelle, um Bean-Instanzen zu erzeugen, zu finden oder zu löschen. import javax.ejb.EJBHome;
  import javax.ejb.CreateException;
  import java.rmi.RemoteException;
- public interface CalculatorHome extends EJBHome
  {
- // erzeugt eine Bean
  public Calculator create()
  throws CreateException,
  RemoteException;
  }

Die Bean Klasse

- Die Bean-Klasse implementiert die Methoden, die im Home- und im Remote-Interface definiert wurden, ohne diese beiden Interfaces im Sinne des Schlüsselwortes **implements** tatsächlich einzubinden. import javax.ejb.SessionBean;
  import javax.ejb.CreateException;
  import javax.ejb.SessionContext;
  import java.rmi.RemoteException;
- public class CalculatorBean implements SessionBean
  {
- // Die Methoden, die einem Client zur
  // Verfügung gestellt werden
- public int add(int a, int b)
  {
  return a + b;
  }
- public int sub(int a, int b)
  {
  return a - b;
  }
- public int mul(int a, int b)
  {
  return a * b;
  }
- public double div(int a, int b) throws ArithmeticException
  {
  return a / b;
  }
-

Die Bean Klasse II

```
// notwendige Methoden, die den Lebenszyklus
    // einer Bean betreffen
    public void ejbCreate()
    {
        System.out.println("ejbCreate()");
    }
    public void ejbRemove()
    {
        System.out.println("ejbRemove()");
    }
    public void ejbActivate()
    {
        System.out.println("ejbActivate()");
    }
    public void ejbPassivate()
    {
        System.out.println("ejbPassivate()");
    }
    public void setSessionContext(SessionContext context)
    {
       System.out.println("setSessionContext()");
    }
}
```