## Was ist J2EE?

Java 2 Platform, Enterprise Edition = die Spezifikation einer Standardarchitektur für die Ausführung von Anwendungskomponenten. Die aktuelle Version der J2EE-Spezifikation ist die Version 5.0.

- Spezifikation durch den Java Community Process verabschiedet
- An J2EE 1.3 waren insgesamt 23 Unternehmen beteiligt
- Standardisierte Infrastruktur
- J2EE definiert die Umgebung und die Dienste, die einer J2EE-Anwendung zur Verfügung stehen
- Obermenge von J2SE (Standard Edition)

### J2EE ermöglicht

- Komponenten basierte Anwendungen
- zentralisierte Geschäftslogik
- Web Anwendungen
- Web Services
- mehrschichtige Software-Architkturen
- Transaktions-Handling

### J2EE standardisiert den Zugriff auf:

- Datenbanken
- Message Oriented Middleware (MOM)
- Legacy und Backend-Systeme

### Vereinfacht und vereinheitlicht Installation, Administration von Applikationen

- J2EE ist eine Architektur – versus z.B. C++ das lediglich eine Programmiersprache darstellt
- J2EE ist Platform und Hersteller unabhängig – versus MS .NET.
- J2EE ist pragmatischer und unkomplizierter als CORBA - adressiert jedoch die gleichen Architekturmodelle.
- J2EE ist von Grund auf Netzwerk und Internet fähig sowie als eine verteilte Drei Schichten Architektur ausgelegt.
- J2EE unterstützt in gleicher Weise Client- wie auch serverseitige Programmierung.
- Java selbst ist eine etablierte, sehr moderne, sichere und ausgesprochen leistungsfähige Programmiersprache.

### Die Ursprünge von J2EE

- Wechsel von einer 2-Schichten Client/Server zu einer 3-Schichten
  - Architektur Trennung der Geschäftslogik von der Präsentationslogik und dem Backend
- Idee der Software-Komponenten
- Verteilte Architekturen mittels Object Request Brokern (CORBA)
- Namensdienste
- Transaktionsmonitore
- Message Oriented Middleware
- Internet-Technologie (http, html, xml)

### Components

- Web Components
  - Servlets, JSPs, Filters, Event Listeners
- Enterprise JavaBeans
  - Session (stateful/less), Message Driven, Entity
- Resource Manager Drivers ("Connectors")
- Application Clients
- Applets

### Containers

- Web Container
- EJB Container
- Application Client Container
- Applet Container

## J2EE Application Server

J2EE-Komponenten erfordern als Laufzeitumgebung eine spezielle Infrastruktur, einen sogenannten J2EE Application Server. Dieser Server stellt technische Funktionalitäten wie

- Sicherheit (Security)
- Transaktionsmanagement
- Namens- und Verzeichnisdienste
- Kommunikation zwischen J2EE-Komponenten
- Management der Komponenten über den gesamten Lebenszyklus (inklusive Instanziierung)
- Unterstützung für die Installation (Deployment)

zur Verfügung. Des weiteren kapselt der Server den Zugriff auf die Ressourcen des zugrundeliegenden Betriebssystems (Dateisystem, Netzwerk ...). J2EE-Server werden häufig auch mit dem Begriff Application Server bezeichnet.

Ein J2EE-Server wird in diverse logische Komponenten unterteilt. Diese werden Container genannt. Die aktuelle Spezifikation erfordert die folgenden Container

- einen EJB-Container als Laufzeitumgebung für Enterprise Java Beans
- einen Web-Container als Laufzeitumgebung für Servlets und Java Server Pages (JSP)
- einen JCA-Container als Laufzeitumgebung für JCA Connectoren. Dieser ist zwar nicht explizit definiert, faktisch jedoch muss jeder Application-Server Hersteller diesen implementieren. Denn im Enterprise Java Beans(EJB) sowie im Web-Container sind restriktionen definiert, welche für die JCA-Laufzeitumgebung nicht gelten. Dabei handelt es sich beispielsweise um das Starten von Threads oder das Lesen und Schreiben in Files etc.

### Container Services

welche der SUN Application Server zur Verfügung stellt:

- Life cycle management for components
- Distribution (Corba IIOP)
- Resource management
  - Multi-threading, state management, resource pooling
- Transaction Handling
  - Container managed (EJBs) vs. component managed
- Security management
  - Authentication, authorization, secure communication
  - Declarative vs. programmatic

# Deployment

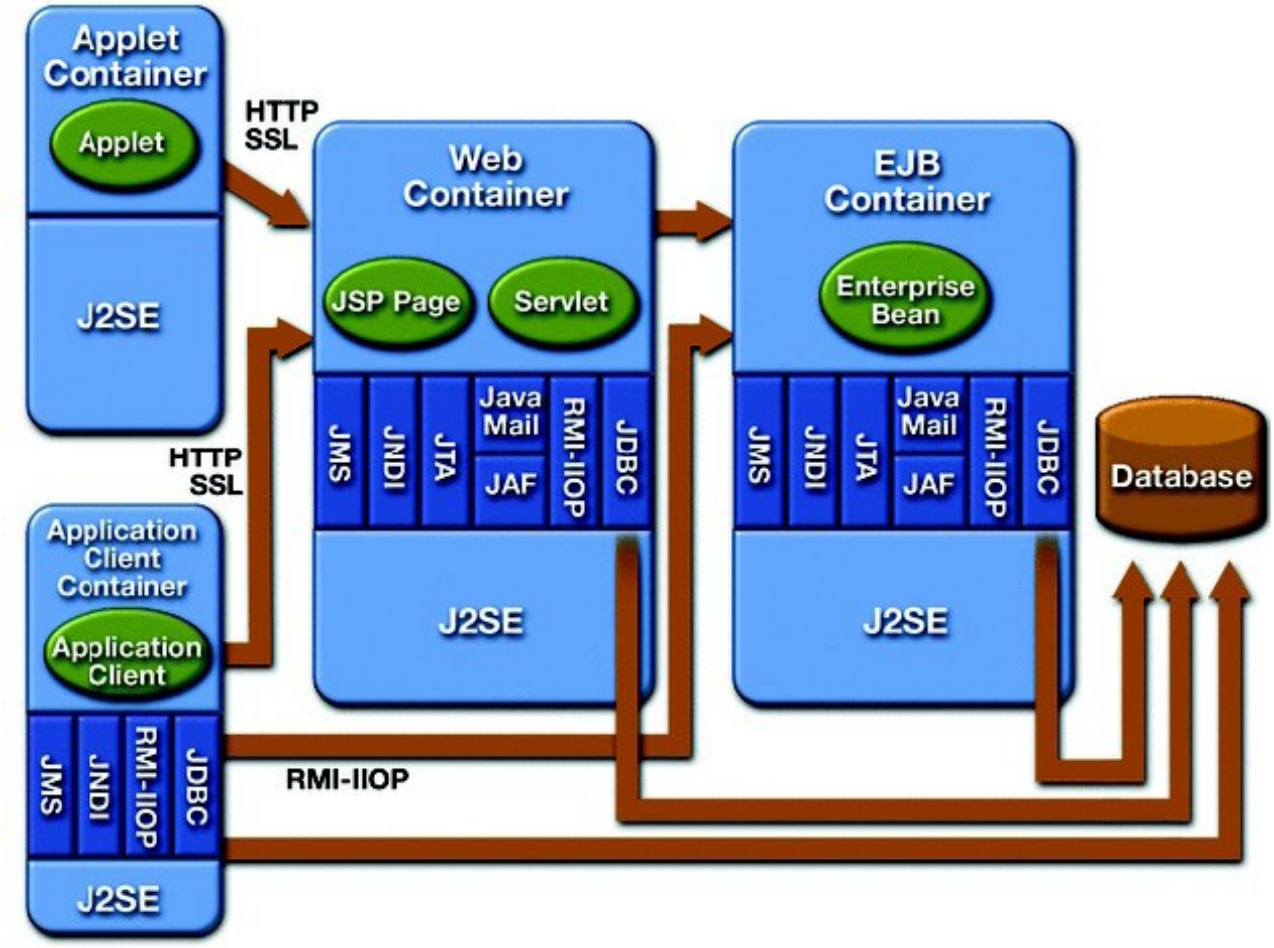
## Deployment Descriptoren

Datei im XML-Format, die ein oder mehrere EJBs und das Archiv beschreibt, zu dem die EJBs zusammengefasst sind. Hier werden alle Informationen abgelegt, die nicht im Bean-Code enthalten sind. Das sind deklarative Informationen, die beim Zusammenfassen von EJBs zu Applikationen und beim Installieren der EJBs in einen EJB-Server benötigt werden. Im Deployment Descriptor stehen Informationen zum Typ und zur Struktur des EJB und über die Abhängigkeit des EJB zu anderen EJBs oder zu Ressourcen, wie z.B.

- Name, Klasse und Schnittstellen einer EJB
- Informationen darüber, ob bestimmte Methoden unter bestimmten Arten von Transaktionen aufgerufen werden dürfen oder müssen.
- Referenzen auf Ressourcen, die der Bean vom Container bereitgestellt werden müssen, z.B. Datenquellen.
- Referenzen auf andere EJBs oder Webservices.
- Optional die Definition der Endpunkte von Webservices als die die EJBs angeboten werden sollen.
- Für Entity Beans mit Container Managed Persistence der Name ihres abstrakten Schemas sowie die Definition ihrer persistenten Felder und Beziehungen untereinander. Außerdem können Queries für bestimmte Suchmethoden (sogenannte Finders) definiert werden.

# J2EE Platform

## Kurzübersicht




***Enterprise JavaBeans (EJB) container***
: Manages the execution of enterprise beans for J2EE applications. Enterprise beans and their container run on the J2EE server.

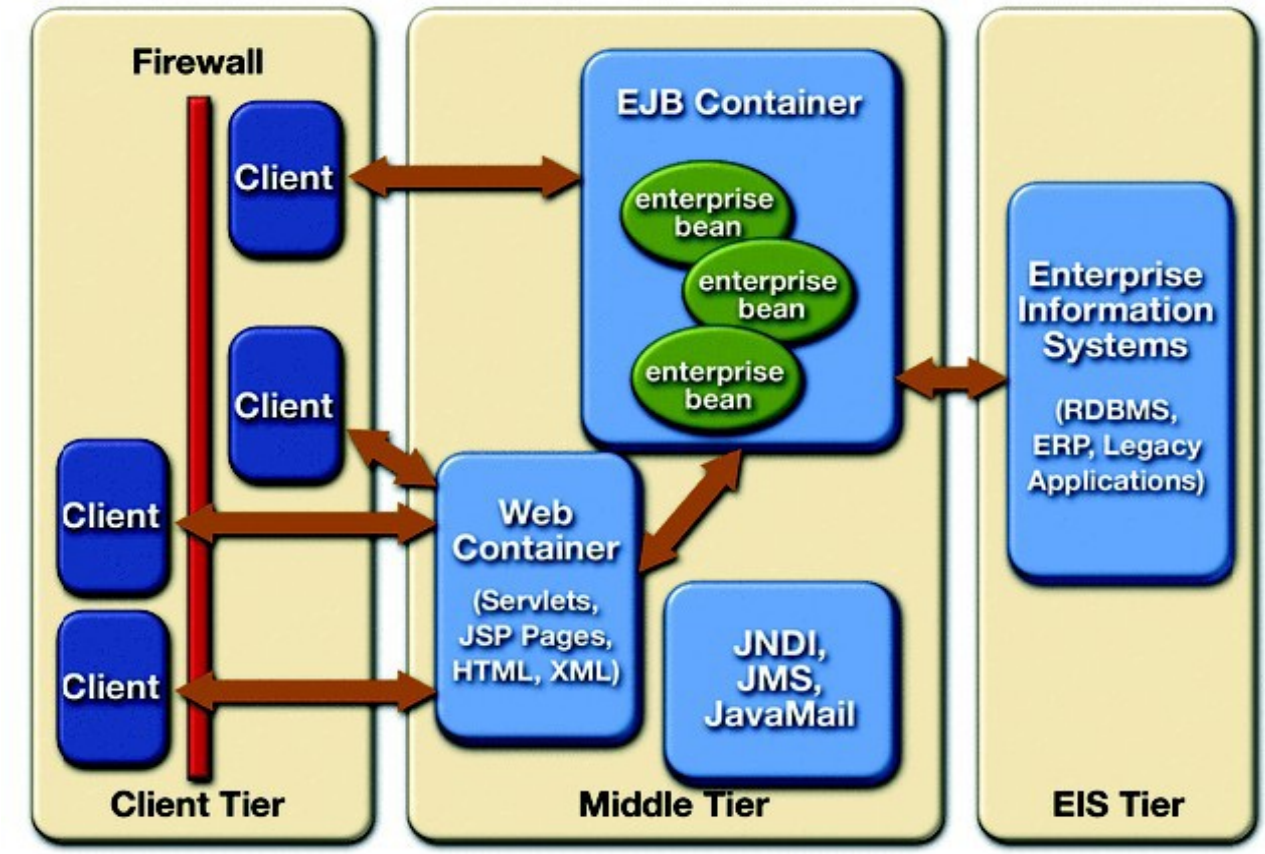
***Web container***
: Manages the execution of JSP page and servlet components for J2EE applications. Web components and their container run on the J2EE server.

***Application client container***
: Manages the execution of application client components. Application clients and their container run on the client.

***Applet container***
: Manages the execution of applets. Consists of a Web browser and Java Plug-in running on the client together.

## Typisches Application Szenario

Hier wird eine 3 Tier Architektur verwendet um ein Application Szenario aufzuzeigen.




***Client Tier***

Hier befinden sich die Clients welche auf den Middle Tier zugreifen sollen. In diesem Tier werden wiederum 2 Szenarien unterschiden:

1. Client außerhalb der Firewall (zB: Zugriff über das Internet)
2. Client befindet sich hinter der Firewall (zB: sitzt im Firmeninternen LAN)

***Middle Tier***

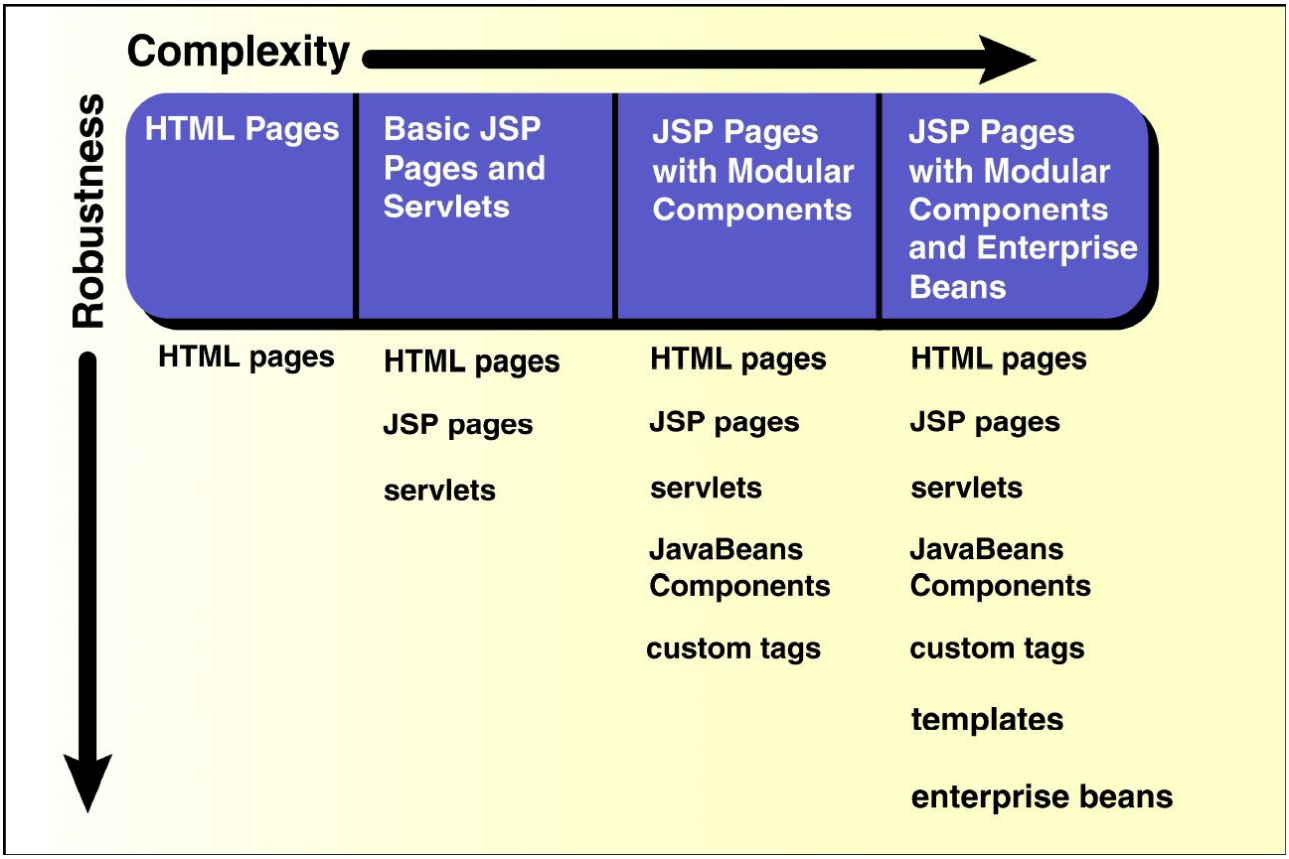
Hier befinden sich die Services bzw. Container (Web & EJB) welche die verschieden Clients ansprechen. Kurz gefasst, die Geschäftslogik.

***EIS Tier***

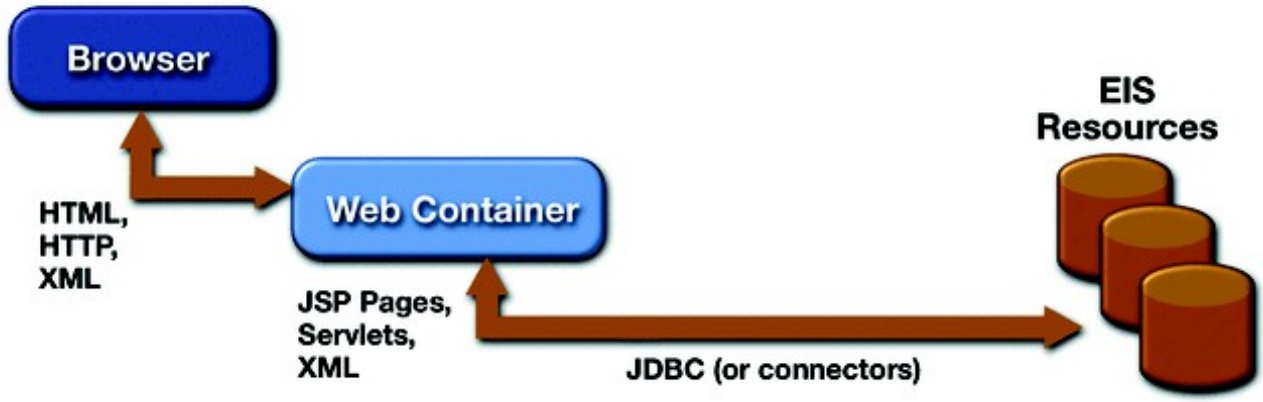
Liefert einheitliche APIs für den Zugriff auf zB: Datenbanken

## Web Applications

Je nach Komplexität und gewünschter Robustheit fällt die Entscheidung für eine Architektur bzw. eine Technologie aus. Jedoch sollte man auch wissen das je komplexer und stabiler ein Lösung sein soll, diese auch den nötigen Aufwand hervorbringt.

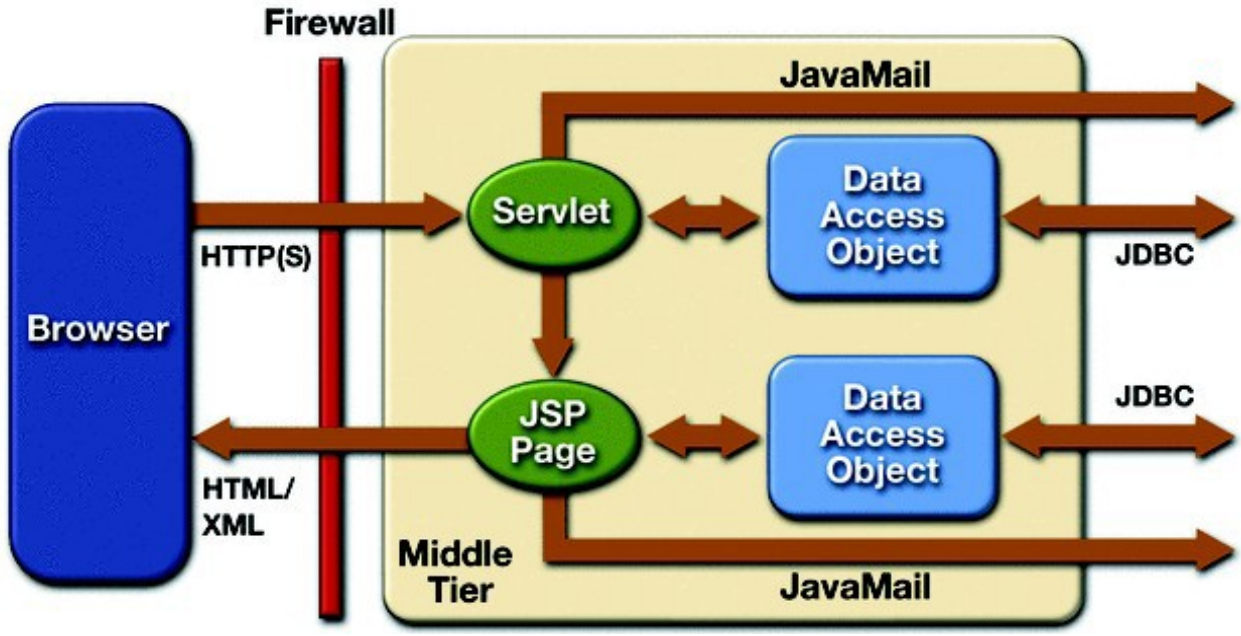



### Simple Web-Centric Application



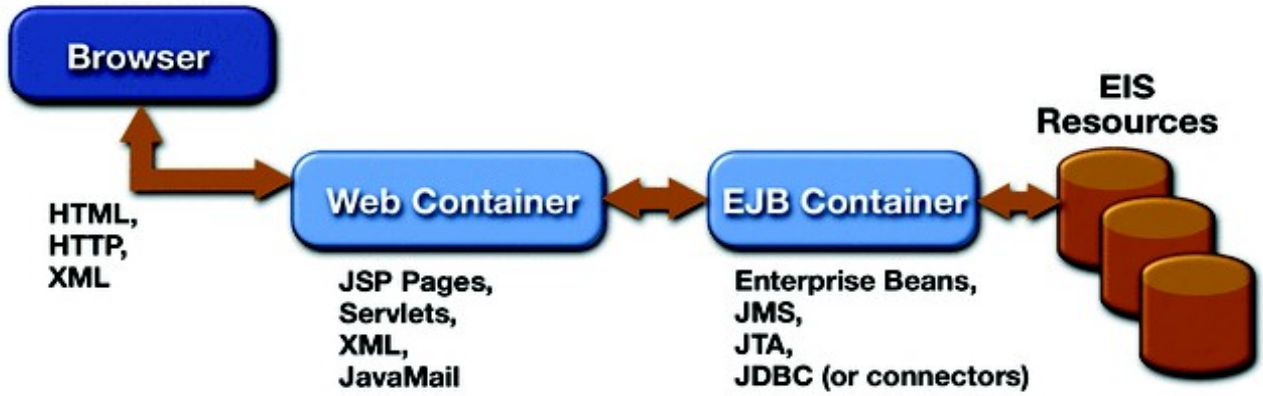

Als Beispiel sei hier eine normal JSP Site genannt inkl. Datenbankenzugriff wie wir sie schon des öfteren erstellt haben.

### Layered Web Application



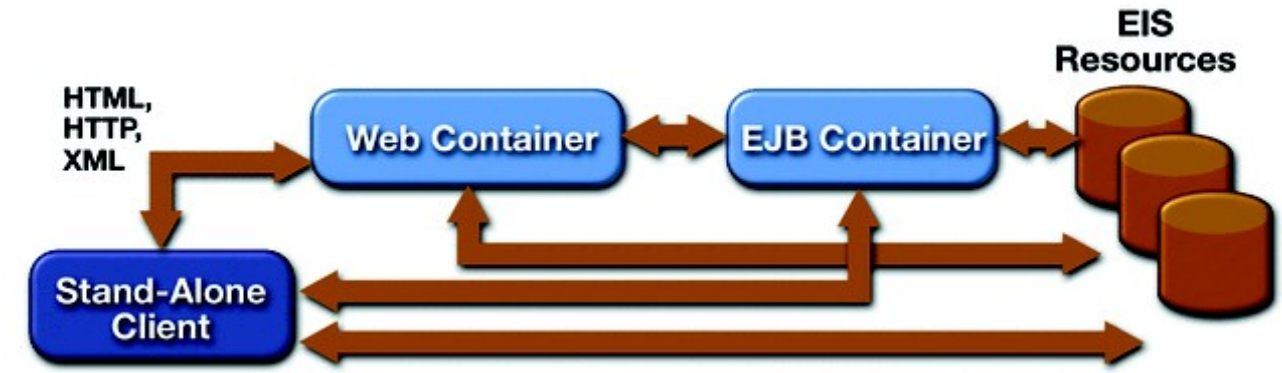

Hier wurde die Komplexität zwar gesteigert – im Endefekt besteht jedoch noch immer das selbe Szenario wie bei einer Simple Web-Centric Application. Browser fordert Daten außerhalb des LANs über die Firewall an. Wird auf das Servlet verwiesen welche dir restliche Logik übernimmt (Sternpunkt). Also Antwort bekommt der Client eine JSP Page geliefert die im Browser angezeigt werden kann. (zB: unser JavaMail Beispiel verwendet eine ähnliche Architektur)

### Web Application Using EJBs




Diese Application könnte man in einer 3 Tier Architektur implementiern (sieh oben). Hier werden einfach die Anfragen vom Browser über den Webcontainer zum EJB Container geleitet. Dieser handled die nögiten zB: Datenbankzugriffe. (zB: die letzte Übung mitn Gerald in der er uns die EJBs im Crash Kurs vorgestellt hat)

### Rich Client Distributed Application




Ein Szenario das nicht immer erwünscht bzw. nicht immer möglich ist wär in diesem Fall abgebildet. Ein Client der je nach Möglichkeit die verschieden Container nutzt oder die Datenbank zugriffe direkt vollzieht. (zB: NetBeans)

## Presentation Tier

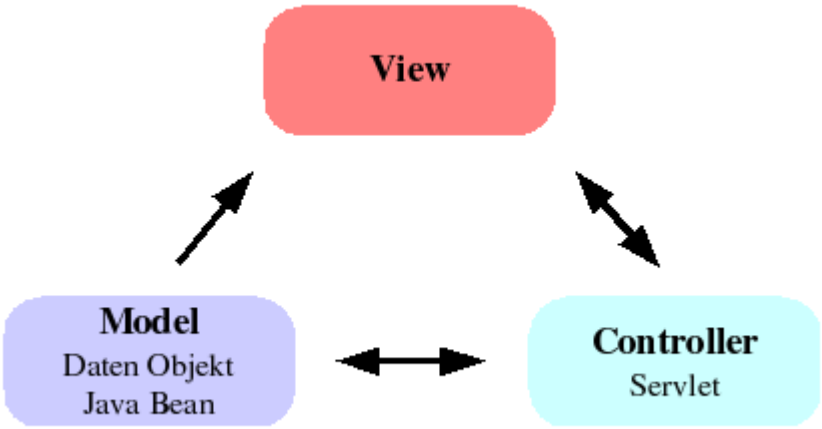
### MVC Pattern

Im Bezug auf den Presentation Tier und J2EE Technologien

***View (JSP)***
enthält nur Layout.
Ideal: Kein Scriptlet-Code,
sondern nur Actions,
Directives und Costum Tags

***Model (Java Bean)***
enthält Datenstrukturen

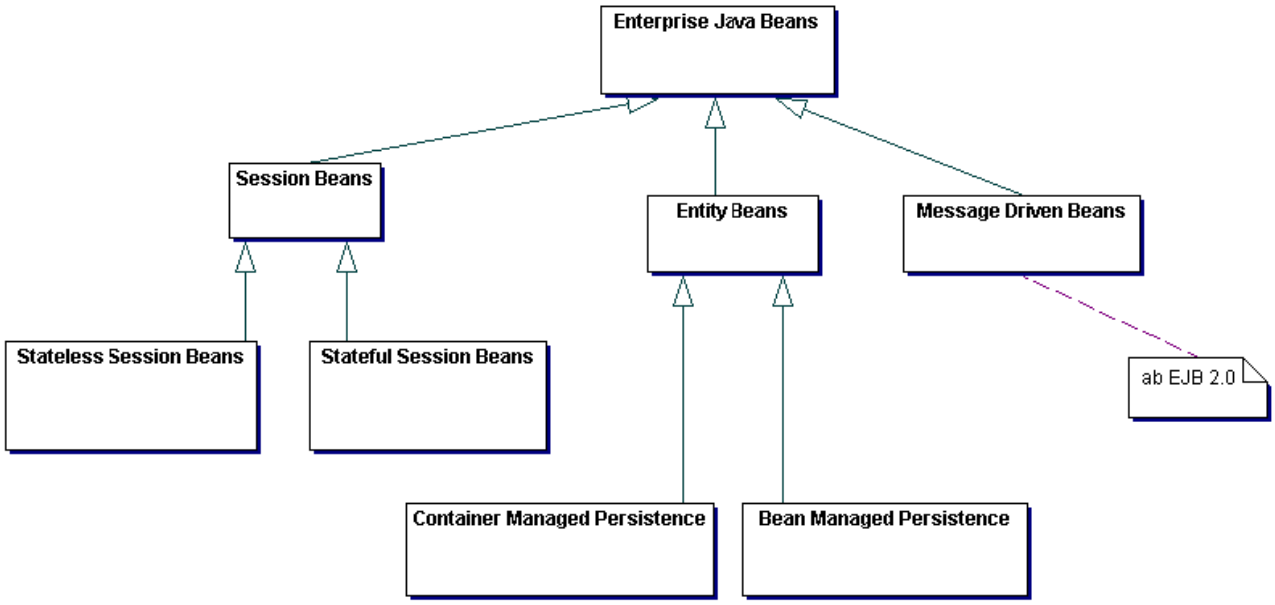
***Controller (Servlet)***
steuert ScreenFlow und gibt
keinen HTML Code aus




## Business Tier (Enterprise Java Beans)

Enterprise Java Beans (EJB) sind standardisierte Komponenten innerhalb eines J2EE-Servers (Java 2 Enterprise Edition). Sie vereinfachen die Entwicklung komplexer mehrschichtiger verteilter Softwaresysteme mittels Java. Im Gegensatz zu den Java Beans, die für die Präsentation eingesetzt werden, also z. B. als visuelle Steuerelemente (Buttons, Scrollbars), stellen Enterprise Java Beans wichtige Konzepte für Unternehmensanwendungen bereit, z.B. Transaktions-, Namens- oder Sicherheitsdienste, die für die „Geschäftslogik“ einer Anwendung benötigt werden.

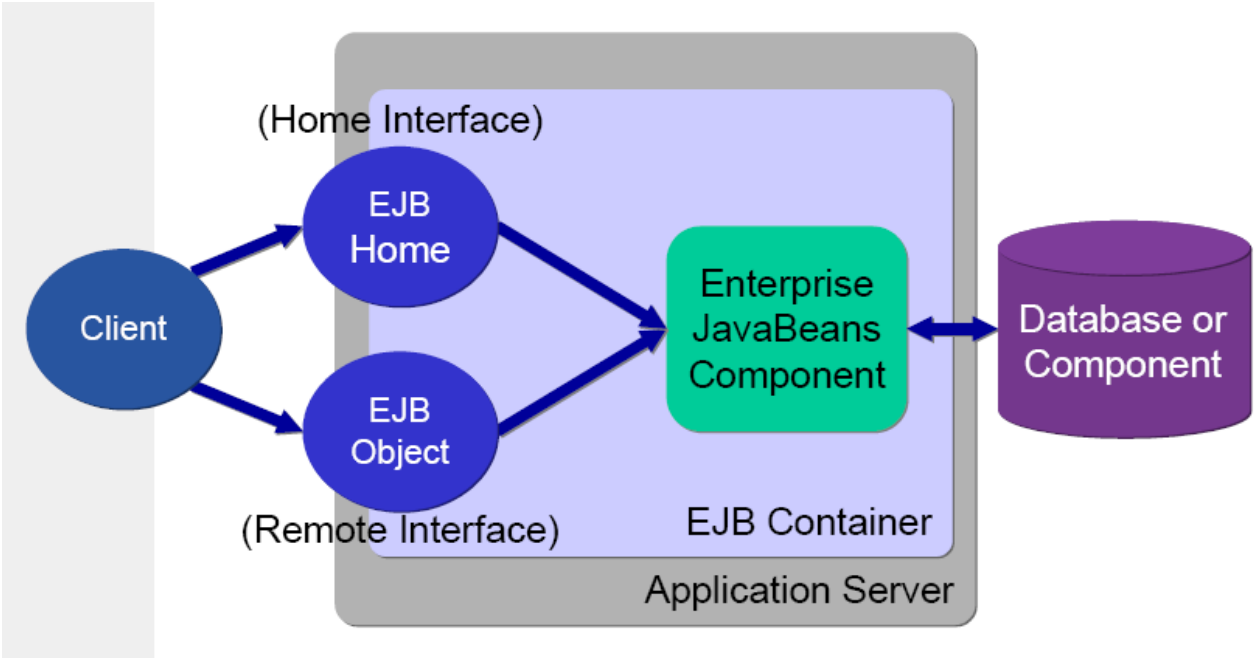
## Enterprise Java Beans Typen




## EJB Gemeinsamkeiten

- Lebenszyklus wird vom EJB-Container gesteuert
- Portierbares Deployment: “Lokation-Unabhängigkeit”
- Mindestens eine Source-Datei (Bean Implementations-Klasse)
  - Bean Implementations-Klasse (Eigentliche Implementation der Methods)
  - Remote-Interface (Deklariert Methoden für Remote-Aufrufe)
  - Home-Interface (Buchhaltung: create(), findByXXX(), remove())
  - Primary Key Klasse (nur bei Entity Beans)
  - Verbindung im Deployment-Deskriptor

## EJB Model



## EJB Container

Ein EJB-Container ist eine Software, die auf einem Server läuft und sogenannte Enterprise Java Beans (EJBs) verwaltet. Der Container kümmert sich sowohl um die Speicherung der Daten, als auch um deren Verfügbarkeit für jeden autorisierten Client. Ein EJB-Container arbeitet als Vermittlungsschicht zwischen Client und Datenbank, daher spricht man von Middleware. Der Vorteil dieser Lösung liegt darin, dass man den Client fast beliebig austauschen kann, ohne etwas an der Mittelschicht ändern zu müssen.

Der EJB Container ist eine Laufzeitumgebung für Enterprise Beans. Der Container ist dazu verpflichtet einem Enterprise Bean mindestens folgende Dienste bereitzustellen:

- API des aktuellen JDK oder das der Java2Platform
- JNDI
- JDBC
- JavaMail
- JTA

Außerdem kümmert sich der Container um

- Transaction Management,
- Persistence Management und
- Security Management.

### *Enterprise Java Beans*

- sind über ein Remote Interface ansprechbare Komponenten
- beinhalten die Geschäftslogik
- wickeln Datenbanktransaktionen ab
- reagieren auf eingehende Messages aus Message Queues

### *Component API: Interfaces and Classes*

#### *Home Interface*

erweitert javax.ejb.EJBHome
Provides remote access to create, find, remove beans

#### *Remote Interface*

erweitert javax.ejb.EJBObject
Provides remote access to business methods

#### *Bean Class*

erweitert javax.ejb.EnterpriseBean
Implements Business Logic and other methods

## EJB Home Factory

(= Service Locator)
Der Client muss für die Nutzung von EJBs deren Home Objekte kennen. Diese Erhält er über die Auflösung der Bean Namen über einen Namensdienst (JNDI).
Die EJB Home Factory wird verwendet um einmal gefundene Home Objekte zu Cachen, sodass nicht jedesmal eine neue Remote Abfrage nötig ist.

# Typen von Beans

# Session Beans

- Session Beans sind nicht persistent
- können aber auf die Datenbank zugreifen

Über Session Beans erfolgt die Kommunikation mit Komponenten außerhalb des Containers (Meistens mit JSP oder Servlet über welche der Client zugreift).

Session-Beans bilden insbesondere Vorgänge ab, die der Nutzer mit dem System durchführt. Sie bedienen sich häufig mehrerer EntityBeans, um die Auswirkungen des Prozesses darzustellen. Man unterscheidet zustandslose ("stateless") und zustandsbehaftete ("stateful") Session-Beans.

## Stateless Session Beans

Stateless Session Beans haben KEINEN eigenen internen 'Zustand', den sie sich zwischen Methodenaufrufen merken. Nacheinanderfolgende Methodenaufrufe eines Clients werden eventuell von verschiedenen Stateless-Session-Bean-Instanzen behandelt. Stateless Session Beans sind zu bevorzugen, da sie performanter, leichter vom Container zu verwalten und im J2EE Application Server besser skalierbar sind.

## Stateful Session Beans

Eine zustandsbehaftete Session Bean hat ein eigenes Gedächtnis. Sie kann Informationen aus einem Methodenaufruf speichern, damit sie bei einem späteren Aufruf einer anderen (oder der gleichen) Methode wieder zur Verfügung stehen. Die Zustandsbehaftung wird durch die Vergabe einer eindeutigen ID umgesetzt, über diese ID können die stateful Session Beans unterschieden werden.

Transaktionen werden meistens von Session Beans verwaltet. Einige Transaktionsarten können aber auch von Entity Beans oder von Clients außerhalb des EJB-Containers verwaltet werden.
Daher kann ein stateful Session Bean auch durch die Art der Transaktionsverwaltung unterschieden werden:

BMT (Bean Managed Transactions)
CMT (Container Managed Transactions)

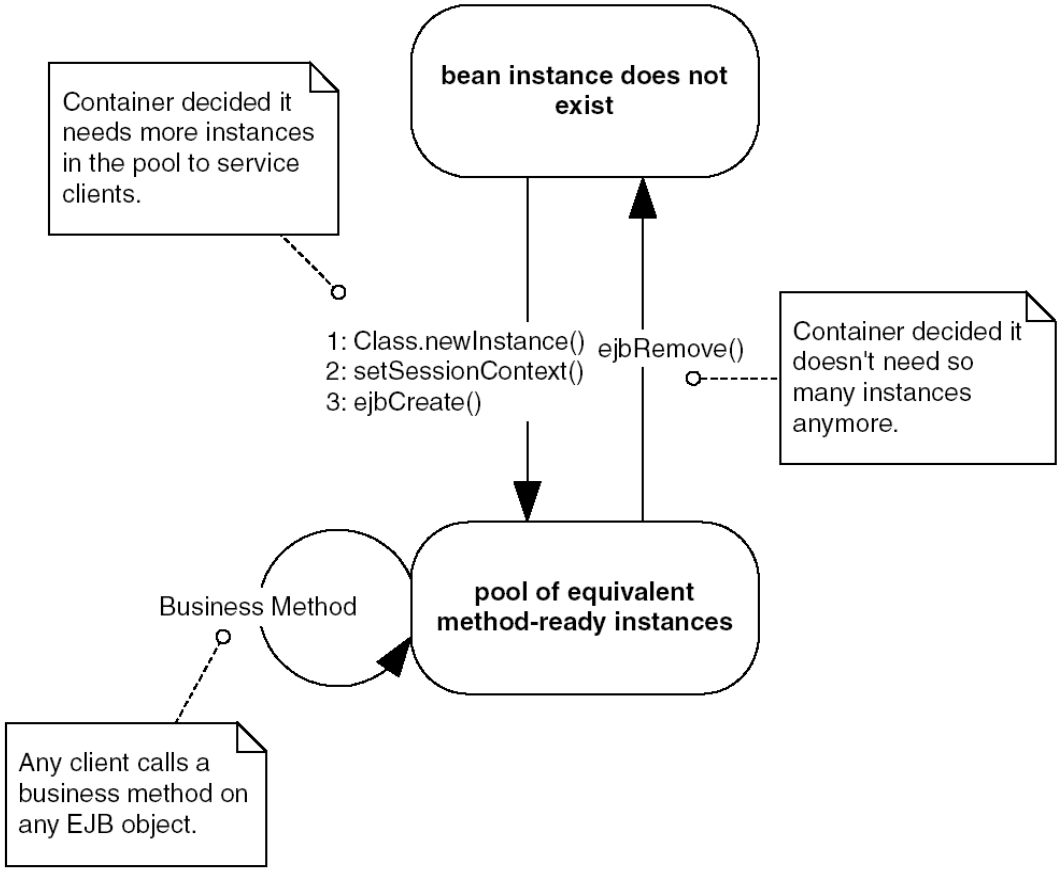
## Writing a session bean

- entscheiden zwischen stateful/stateless
- entscheiden zwischen local und/oder remote client view (oft wird nur remote verwendet)
- home interface(s) schreiben
- business interface(s) schreiben
- component interface(s) schreiben
- Bean Class schreiben
- deployment descriptor schreiben
- alles in ejb-jar file packen
- ejb-jar auf app server deployen

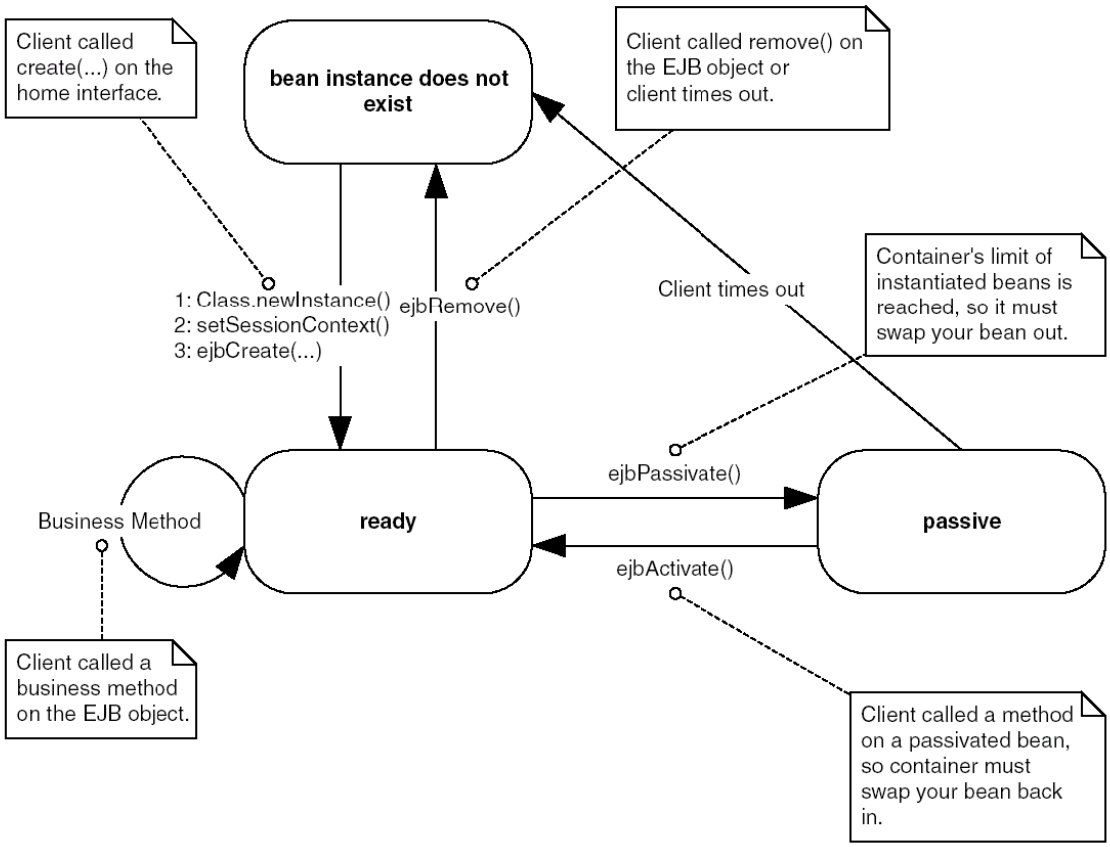
## Using a Session Bean

- Home interface mittels JNDI finden
- eine Session-Bean-Instanz des Home interface erzeugen
- Business-Methoden auf Instanz aufrufen
- Instanz zerstören

## Stateless Session Bean Life Cycle



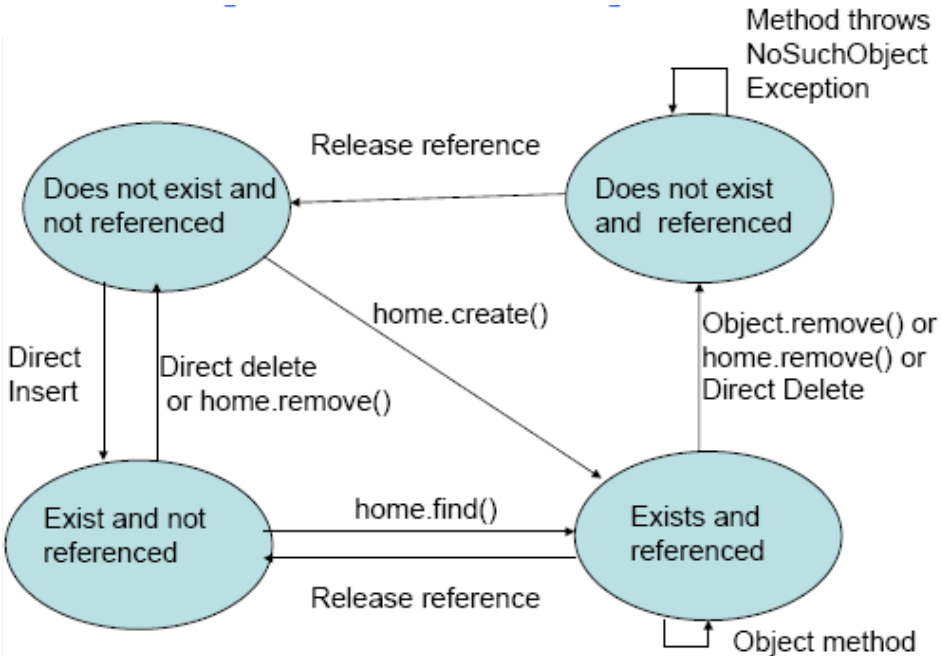
### Stateful Session Bean Life Cycle




## Entity Beans

Die Hauptaufgabe der Entity Beans ist die Persistenz, also das dauerhafte Speichern über mehrere Sitzungen hinweg (meistens in relationalen Datenbanken).
Entity Beans repräsentieren oft ein Objekt aus der realen Welt (in objektorientierter Darstellung). Die Daten einer Entity Bean entsprechen einer Zeile einer Datenbanktabelle (oder mehrerer Tabellen). Normalerweise können auch nur Operationen an genau einem Datensatz (also einer Tabellenzeile) durchgeführt werden (anders als bei SQL, wo über eine 'where'-Bedingung viele Tabellenzeilen auf einmal manipuliert werden können).
Anders als eine Session Bean kann eine Entity Bean mehrere Clients gleichzeitig bedienen (dabei sind Clients in der Regel Session Beans).

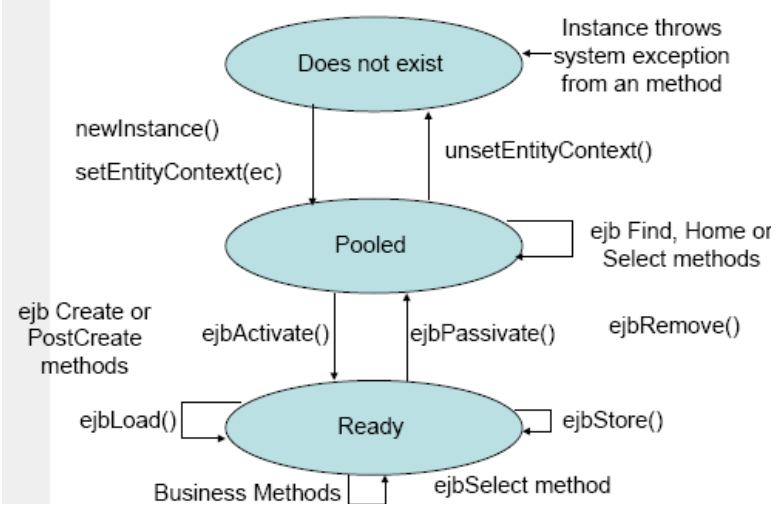
## Life Cycle




Der Lebenszyklus eines Entity Objektes ist unabhängig vom Lebenszyklus der Objektreferenzen am Client. (Ein Entity Objekt wird nicht automatisch mit der Referenz entfernt, und umgekehrt).

## Entity Bean Lifecycle States

Jede Entity Bean-Instanz hat einen Lebenszyklus, der bei der creation-time anfängt and bis zur Entfernung andauert. Der EJB-Container managed den Lebenszyklus für jede Instanz einer Entity Bean-Klasse. Gute EJB-Entwicklungstools machen es für den Entwickler überflüssig, sich um den Lebenszyklus zu kümmern.




**Does not Exist**
Container hat noch keine Instanz erzeugt, oder sie verworfen

**Pooled**
Instanz ist noch nicht mit einem bestimmten Entity-Objekt assoziiert

**Ready**
Instanz ist mit einem bestimmten Entity-Objekt assoziiert -> hat Identität (PrimaryKey)

Die Statusübergänge werden durchgeführt, wenn:

- Der Client eine Methode auslöst
- Interne Events (z.B. transaction commit oder Exceptions)

Eine neue Instanz wird durch das Keyword “new” durch den Container erzeugt. Durch „setEntityContext“ werden die Referenzen zu seinem EntityContext-Objekt gesetzt.
Aktionen des Containers beim Pooled-Status:

- Ausführen von „ejbFind“, „ejbSelect“ oder „ejbHome“-Methode
- Verknüpfen einer Instanz mit einem existierenden Entity-Objekt durch „ejbActivate“
- Mit dem Aufruf von „ejbCreate“ oder „ejbPostCreate“ der Instanz wird ein neues Entity-

Objekt erzeugt
- Durch „unsetEntityContext“ wird die Instanz verworfen

Aktionen des Containers beim Ready-Status:
- „ejbLoad“: Instanz wird von der DB in den Cache geladen
- „ejbStore“: Instanz wird vom Cache in die DB geschrieben
- Auslösen einer business-Methode durch eine vom Client ausgelöste Methode
- „ejbPassivate“: Instanz wird auf Pooled gesetzt
- „ejbRemove“ (vom Client aufgerufen): Instanz wird auf Pooled gesetzt

Wenn eine Exception von der Instanz ausgelöst und nicht abgefangen wird, fängt diese der Container und ändert ihren Status auf“Does not exist“.

### Erstellen eines Entity Objektes

Ein Client kann ein Entity Objekt erzeugen, indem er die create-Methode aufruft, welche im Home-Interface der Entity Bean definiert ist. Ein Entity Objekt kann auch direkt in der Datenbank ohne der Entity Bean und deren Container erzeugt werden.

### Suchen eines Entity Objektes

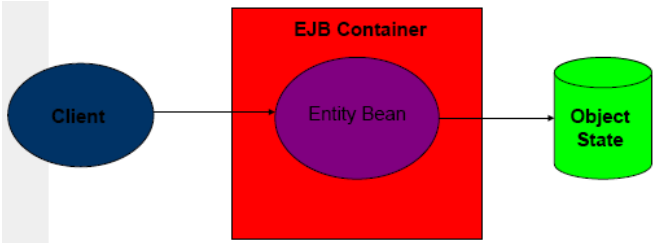
Ein Client kann ein existierendes Entity Objekt auffinden, indem er die find-Methode des Home-Interfaces verwendet.

### Aufrufen von Business Methods

Ein Client, welcher eine Objektreferenz für das Entity Objekt hat, kann Business Methods auf dem Entity Objekt aufrufen.

### Entity Objekt Status

Der Entity Objekt Status wird vom Ressourcenmanager verwaltet, auf welchen die Methoden der Entity Bean-Klassen zugreifen. Diese Verwaltung ist unabhängig vom Containermanagement der Entity Bean Instanzen.




Der Entwickler kann zwischen 2 verschiedenen Methoden wählen, um auf den Objektstatus zuzugreifen:
- Container Managed Persistence (CMP)
- Bean Managed Persistence, (BMP)

Für Entity Beans welche CMP verwenden, kann der Entwickler die EJB query language verwenden, um die find-Methode zu implementieren.

### Persistence

Das Trennen des Entity Beans-Status und der Instanz selbst hat die folgenden Vorteile:
- Das Ermöglicht dem Entity Objekt-Status persistent zu sein.
- Durch diese Trennung ist der Lebenszyklus des Entity Bean-Status unabhängig vom Lebenszyklus der Entity Bean Klassen-Instanzen und vom Lebenszyklus der JVM, in welcher die Instanzen erzeugt wurden.
- Der Ressourcenmanager ist für die ACID-Eigenschaften der Transaktionen verantwortlich. (ACID…erwünschte Eigenschaften von Transaktionen bei DBs oder verteilten Systemen)

### Container Managed Persistence

…ist eine vom Ressourcenmanager unabhängige DataAccess-API speziell für Entity Beans.

Die Vorteile gegenüber BMP sind:

- DB Access Code ist einfacher zu schreiben
- BMP benötigt meistens Implementierung von JDBC-Calls
- Bei CMP muss nur angegeben werden, welcher Status persistent gespeichert werden soll
- Der Container übernimmt die Handhabung der Persistenz
- Bei BMP ist der Entwickler für die Auswahl des Persistenz-Status und die Handhabung der Persistenz selbst verantwortlich
- CMP ermöglicht dem Container ein intelligentes Cachen des Objektstatus.

Erweiterte Portabilität

- Der Entwickler verwendet die gleiche API unabhängig vom Ressourcenmanager
- Das gleiche Entity Bean wird unabhängig von der Datenbank verwendet
- Während der Deployment-Zeit können Entwickler Entity Beans entwickeln, welche man anpassen kann, um mit verschiedenen DBs zu arbeiten
- Der Deployer passt die Entity Beans an die gewünschte DB an

CMP Beans haben eine bessere Performance:

- BMP Beans verwenden JDBC-Access
- CMP Beans können datenbankspezifische Features verwenden (Connection-Pooling, optimized Caching)
- Das CMP dem Container vollen Zugriff gewährt, kann dieser den DB-Zugriff für höchste Leistung optimieren

### Container Managed Relationships

CMRs (container managed relationships) sind Beziehungen zwischen mehreren Entity Beans. Diese werden durch CMR-Fields im Deployment Descriptor beschrieben.
Der EJB-Container ermöglicht 1:1, 1:n und n:m-Beziehungen, wobei diese Beziehungen und die referentielle Integrität automatisch gemanaged werden. Der Bean-Provider spezifiziert diese Beziehungen im Deployment Descriptor. Beim Deployen wird diese Spezifikation zur Schema-Definition und die Beziehungen können in der Beziehungs-DB geschrieben werden.

### Bean Managed Persistence

Bei BMP wird eine Ressourcenmanager-spezifische API verwendet, um auf den Status zuzugreifen. Wenn der Status in einer anderen DB gespeichert ist, verwendet das Bean eine andere API, diese ist auf den Ressourcenmanager zugeschnitten. Der größte Vorteil von BMP ist, das es das Deployen von Entity Beans vereinfacht. Es sind keine Deployment-Schritte notwendig, um die Bean an den Ressourcenmanager oder an dessen DB-Schema anzupassen.
In mancher Hinsicht können BMP Beans Datenunabhängigkeit erreichen, durch die Verwendung einer DataAccess-Komponente für den Zugriff auf den Entity Bean-Status. BMP erlaubt keine DB-Optimierung ausgehend vom Container, jedoch von der Bean selbst.

## Passivation and Activation

Gilt für stateful session beans und entity beans
Der app server (=EJB container) verwaltet den speicher mittels serialisieren/deserialisieren von bean instanzen wann immer nötig. Alle Felder eines EJB müssen serialisierbar sein (das ist Teil der EJB-Spezifikation)

ejbActivate und ejbPassivate werden vom Container aufgerufen

- ejbPassivate
  - alle offenen Verbindungen schließen (zB DB-Connections)
  - alle nicht serialisierbaren Felder auf NULL setzen
- ejbActivate
  - alle nötigen Ressourcen erneut öffnen

- alle NULL Felder re-initialisieren

## Message Driven Beans

Message Driven Beans sind diejenigen Komponenten, die EJB-Systeme für asynchrone Kommunikation zugänglich machen. Sie werden z.B. häufig für die Kommunikation mit Legacy-Systemen genutzt.

Die Message Driven Beans kann nur auf Grund eingehender JMS-Nachrichten aktiv werden. Das JMS-API (Java Message Service) beschreibt Server-Dienste zur Verwaltung asynchroner Nachrichten.
Beim Java Message Service wird unterschieden zwischen:

### *JMS Queue (Point-to-Point)*

Nachrichten werden im JMS Server in einer Nachrichtenschlange (Queue) zwischengespeichert. Nachrichten sind normalerweise für nur einen Konsumenten bestimmt. Nach Abholung befindet sich die Nachricht nicht mehr im JMS Server.

### *JMS Topic (Point-to-Multipoint, Publish/Subsribe, pub-sub, Publizieren/Abonnieren)*

Nachrichten beziehen sich auf ein Thema (Topic) und sind normalerweise für viele Konsumenten bestimmt, die sich als Abonnent registrieren müssen. Auch nach Abholung bleibt die Nachricht weiterhin im JMS Server, damit auch andere Konsumenten die Nachricht lesen können. Nach einer bestimmten Zeit kann die Nachricht auf dem JMS Server gelöscht werden (und wird häufig durch eine aktuellere Nachricht ersetzt, z.B. Börsen-Ticker).

## Remote Client View

Funktioniert sowohl für Session als auch für Entity Beans, wird für Entity Beans aber aus Performancegründen nicht verwendet.

- Ortsunabhängig/Location independent
- API ist dieselbe egal ob das EJB in der selben JVM läuft oder in einer entfernten
- Basiert auf Java-RMI (also Corba/IIOP)
- pass-by-value wird verwendet für Parameter und für Returnwerte (Objekte werden serialisiert)
- Client kann Non-Java sein solange er Corba kann (wird jedoch kaum angewandt)
- EJB Remote Client View ist erkennbar am remote home interface

### *Probleme einer Remote View*

Martin Fowler's erstes Gesetz der verteilten Objekte: "Don't distribute your objects."

Ortsunabhängigkeit ist was Tolles und macht flexibel im Deployment, aber Remote Calls haben...

- hohe Latenzzeit (Network, Marshalling/Unmarshalling von Parametern/Returnwerten ..)
- gehen ev. wegen Netzwerkproblemen schief, Server unerreichbar etc.

Daher möglichst wenige Remote Calls, jeweils mit möglichst viel Datentransport
Der Developer (allererste Rolle) entscheidet sich für Remote und/oder Local Client View.

## Local Client View

Funktioniert sowohl für Session als auch für Entity Beans.
Client und EJB werden in der selben JVM ausgeführt

- nicht Ortsunabhängig/Nicht Location independent
- pass-by-reference wird verwendet für Parameter und Returnwerte (= weniger Overhead als

bei Remote-Aufrufen)
- EJB Local Client View ist erkennbar am local home interface
- Client muss Java sein

## Web Service Client View

Funktioniert nur bei stateless session beans
- calls mittels SOAP over HTTP(s) (anstatt IIOP im remote client view)
- SOAP bringt Ortsunabhängigkeit mit sich (SOAP calls are remote calls)
- EJB Web Service Client View erkennbar am WSDL Dokument (replaces remote interface)
- Client kann Non-Java sein
- Java Clients nutzen JAX-RPC (Java API for XML-based RPC) um einen Web Service zu erreichen

## Transaktionen

### *JTS (Java Transaction Service)*

Der Java Transaction Service (JTS) spezifiziert die Funktionalität eines Transaktionsmanagers. Die zugehörige Schnittstelle heißt Java Transaction API (JTA). Der Ablauf einer globalen Transaktion wird mittels des Zwei-Phasen-Commit-Protokolls koordiniert. Das J2EE-Modell definiert zwei Möglichkeiten zur Kennzeichnung von Transaktion:
- Deklarative Transaktionsbegrenzung
- Programmierte Transaktionsbegrenzung

### *JTA (Java-Transaction-API):*

- Besteht aus javax.transaction Paketen
- Bestimmt den Vertrag zwischen Applikation, Appl-Server, Transaction-Mgr. und Resource-Mgr.

### Transaktionsmanagement

Ein wichtiger Dienst von Enterprise JavaBeans ist das Transaktionsmanagement.
- Implizites Transaktions-mgmt (deklaratives Transaktionsmgmt.).
  - Setzen von Transaktionsattributen im Deployment-Descriptor (XML-File)
  - Transaktionsattribute bestimmen den Gültigkeitsbereich einer Transaktion
  - Sie können auf ein ganzes Enterprise-Bean (Attribute gelten für jede Methode des Bean) oder auf einzelne Methoden angewendet werden.

- Explizites Transaktions-mgmt. (Programmiertes Transaktionsmgmt.)
  - Transaktionen müssen von Hand kontrolliert werden
  - Erfordert, dass Transaktions-Code direkt in die Geschäftslogik geschrieben wird.

Folgende Werte sind für ein Transaktionsattribut definiert:
- NotSupported
  Eine Methode mit diesen Attributen unterstützt keine Transaktion. Wenn ein Client diese Methode aufruft, wird die laufende Transaktion unterbrochen, bis der Aufruf dieser Methode endet.
- Required
  Ein Bean mit Required läuft immer in einer Transaktion. Wenn eine Transaktion schon existiert, wird das Bean an dieser Transaktion teilnehmen. Wenn noch keine Transaktion läuft, wird der EJB Container eine neue Transaktion für das Bean starten.
- Supports
  Wenn ein Bean mit Supports aufgerufen wird, wird es nur dann in einer Transaktion laufen, wenn der Client eine Transaktion ausführt. Wenn also eine Transaktion existiert, dann wird

das Bean an der Transaktion teilnehmen, wenn der Client keine Transaktion ausführt, läuft das Bean auch ohne Transaktion.
- RequiresNew
  Eine neue Transaktion wird immer gestartet, wenn ein Bean mit RequiresNew aufgerufen wird. Falls eine Transaktion schon existiert, wird diese zuerst unterbrochen. Und nur wenn die neue Transaktion endet, wird die alte Transaktion weiter ausgeführt.
- Mandatory
  Ein Bean mit diesem Wert wird gezwungen, in einer Transaktion zu laufen. Wenn das Bean aufgerufen wird, muss eine Transaktion schon gestartet sein. Das Bean läuft dann auch unter dieser Transaktion. Falls keine Transaktion existiert, wird der Container keine neue Transation starten und es wird eine Exception ausgelöst.
- Never
  Ein Bean mit diesem Attribut läuft niemals in einer Transaktion. Wenn ein Aufruf dieses Beans ein Teil von einer Transaktion ist, wird eine Exception ausgelöst. Andernfalls wird das Bean normal ohne Transaktion aufgerufen.

### Bean-Managed-Transaction

- Deployment Descriptor spezifiziert den Transaction typ „Bean“ für die EJB
- Im Code der EJB, Session.getUserTransaction() oder ein JNDI-Lookup für „java:comp/UserTransaction“ gibt UserTransaction objekt zurück, welche von den EJB methoden genutzt werden müssen um die transaktion abzugrenzen.
- Java-Code ist identisch zu transaction abgrenzung in Servlets/JSPs
  - Stateful Sesssion Beans: eine Transaktion muss nicht in dem methoden Aufruf beendet werden, der sie aufgerufen hat.
  - Stateless Session Beans und Message-driven-Beans müssen die Transaktion beenden innerhalb des methoden Aufrufs welche sie gestartet haben.
- Keine Option für Entity Beans.

### Application-Exceptions

- Werden verwendet um Business Logik Probleme zu melden nicht aber technische Probleme
- Kann von jeder Methode im Home od. Komponenten Interface geworfen werden
- Keine sub-Klasse von Runtime-Exception

### System-Exceptions

- Sind alle sub-Klassen von Runtime-Exception inklusive EJBexception
- Werden verwendet um unerwartete Fehler od. Fehler von welcher sich die EJB nicht erholt, passieren.
- Eine System-Exception welche von einer EJB methode geworfen wurde, wird vom EJB Container gefangen.

## Hibernate

Um den Bruch zwischen der objektorientierten Anwendungsentwicklung z. B. mit Java und relationalen Datenbanken, die den Datenbank-Standard darstellen, zu überbrücken, wurden in den letzten Jahren viele sogenannte OR-Mapper (Object Relational Mapper) entwickelt. Hibernate ist ein solcher ORMapper für Java. Er ist Open-Source.

## JDO (JavaDataObject)

- Die JDO definiert eine Standatd API für Daten, welche in lokalen Speichersystemen und heterogenen EnterpriseInformationSystemen enthalten sind.
- Typischweise wird JDO verwendet um auf Objekt Datenbanken (ODBMS)und relationale

DBs (RDBMS, über ORM) zuzugreifen
- JDO ist ein Standard (im gegensatz zu Hibernate)

## Was sind Design Patterns?

- Patterns sind keine Vorlagen (viele unterschiedliche Implementierungen)
- Patterns sind keine Frameworks (viel allgemeiner)
- Sie werden (oft) nicht von Einzelpersonen erfunden, sondern aus der Arbeit von Vielen extrahiert.

### *Behandelte Design Patterns*

- Application Controller
- Business Deligate
- Session Facade
- Data Access Object

## Application Controller

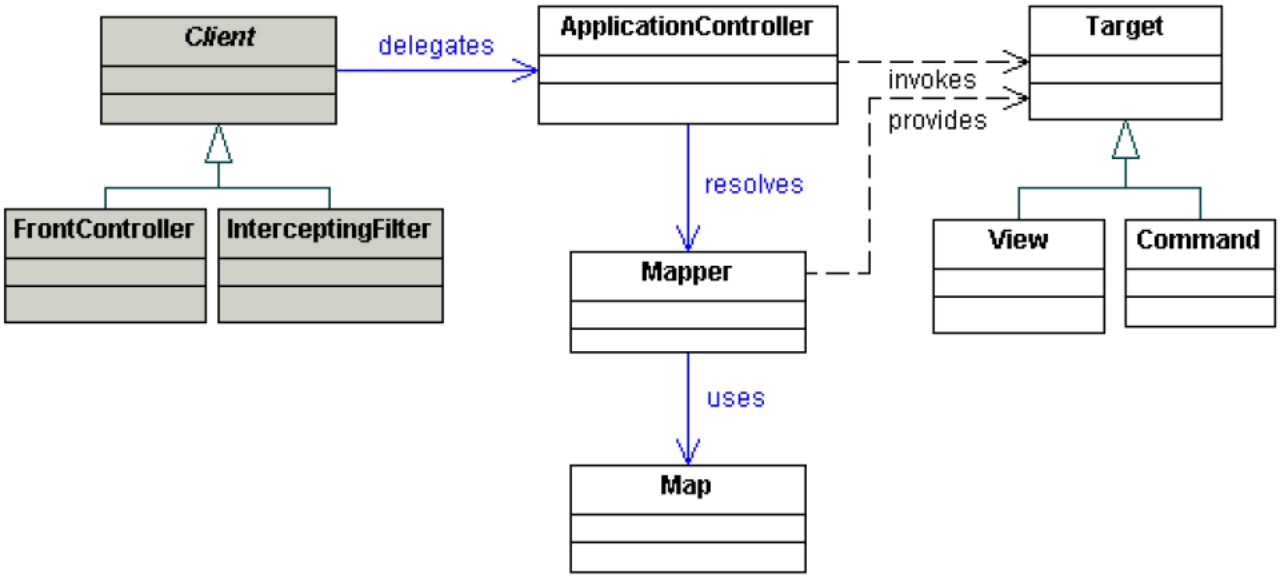
Wird verwendet, wenn man Action- und View-Management zentralisiert und modular benutzen will.

### Wann?

- Action und View-Management Code wiederverwenden
- Erleichtern der request-handling Erweiterbarkeit
- Erleichtung der Erweiterung der Applikation

### Wie?

Das Klassendiagramm in Abbildung zeigt die Struktur der beiden Patterns Front Controller und Application Controller. FrontController bildet den zentralen Zugriffspunkt f¨ur den Request des Client. Die Verarbeitung des Requests wird an den ApplicationController delegiert, welcher einen Mapper verwendet, um den Request zu einem Target aufzulösen. Targets sind Ressourcen wie Views oder Commands.



## Business Delegate

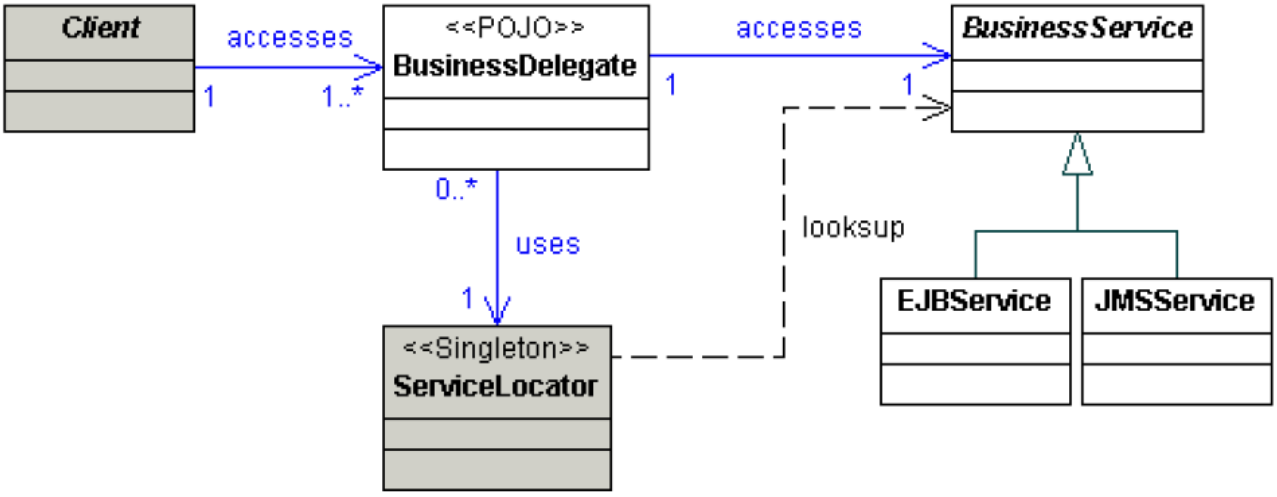
Das Business Delegate Pattern  verbirgt die Komplexität der Kommunikation mit der Service-Schnittstelle vor dem Client. Die Service-Schnittstelle wird in der Regel durch Session Facades gebildet, welche als Session-Beans implementiert sind.

### Wann?

- Um Kopplung zwischen Clients und Business Services zu minimieren.
- kapselt den Zugriff auf die Service-Schnittstelle und verbirgt dessen Implementationsdetails, insbesondere den Zugriffsmechanismus.
- Behandlung von Service-Level-Exceptions. Anstelle der Service-Level-Exceptions wirft das Business Delegate Application-Level-Exceptions, welche für den Client einfacher zu behandeln sind.

### Wie?

Verwende ein Business Delegate um Zugang zu einem Business Service zu kapseln. Business Delegate versteckt die implementierung Details von einem Business Service




### Vorteile

- Übersetzt business service exceptions
- Verbessert Verwendbarkeit
- Verbessert Performance – caching
- führt additional layer ein
- Versteckt remoteness

## Session Facade

Thematisch dem Business Delegate sehr ähnlich. Zielt darauf ab, die im Client integrierte Logik möglichst simpel zu halten und die komplexen Beziehungen der Business Objekte vor ihm zu verbergen. Die Session Facade agiert als eine Art Schnittstelle und enthält die Geschäftslogik des Systems.

### 2 Hauptaufgaben

- Kontrolle der Clientzugriffe auf die Business Objekte
- Minimierung des Netzwerktraffics zwischen Client und Business Objekten

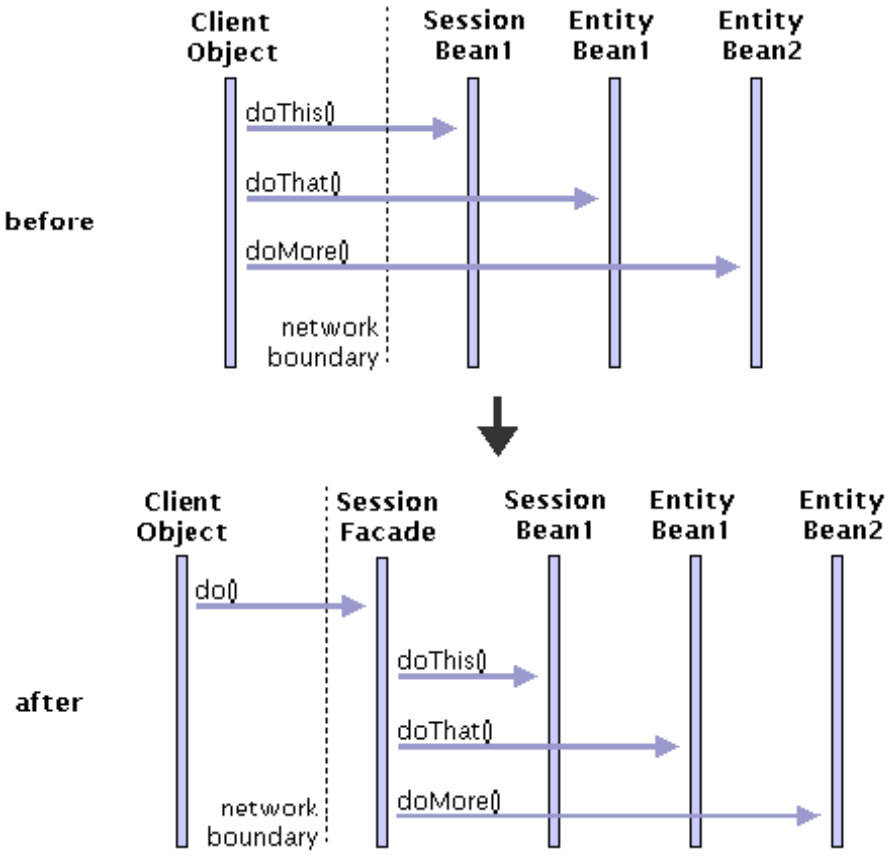
Typischerweise werden server-seitige Geschäftskomponenten als Entity-Beans realisiert. Erlaubt man einem Client, auf diese Komponenten direkt zuzugreifen, kann dies zu mehreren Problemen führen:

- Eine feste Kopplung zwischen Client und Geschäftskomponente impliziert eine direkte Abhängigkeit zwischen diesen. Ändert sich die Schnittstelle der Geschäftskomponente, so müssen Anpassungen an den Clients vorgenommen werden.

- Ein direkter Zugriff erfordert vom Client, dass er über Geschäftslogik verfügt, um die Interaktion mit den verschiedenen Geschäftskomponenten zu koordinieren. Dies untergräbt das multi-tier Modell, wo die Geschäftslogik durch den Business-Tier repräsentiert wird.
- Existieren verschiedene Typen von Clients, führt ein direkter Zugriff auf die Geschäftskomponenten dazu, dass die Logik für die Interaktion mit den Komponenten auf jedem Client existiert. Duplizierter Code auf verschiedenen Clients erschwert die Wartbarkeit des Systems.

Der zweite Punkt, mit welchem sich die Session Facade auseinandersetzt, ist die Minimierung der Netzwerkbelastung. Häufig haben Geschäftskomponenten eine feine Granularität. Greifen Clients direkt auf die Komponenten zu, verursacht dies eine Vielzahl von wiederholten Aufrufen über das Netzwerk. Dadurch wird das Netzwerk stark belastet, was unweigerlich zu einer schlechten Performanz führt.
Im Gegensatz zu den Geschäftskomponenten haben die Session Facades eine grobe Granularität.

Eine Session Facade wird als Session-Bean implementiert, welches mit allen beteiligten Geschäftskomponenten interagiert. Die Geschäftskomponenten sind häufig Entity-Beans, es können jedoch auch andere Session-Beans sein.




### Vorteile

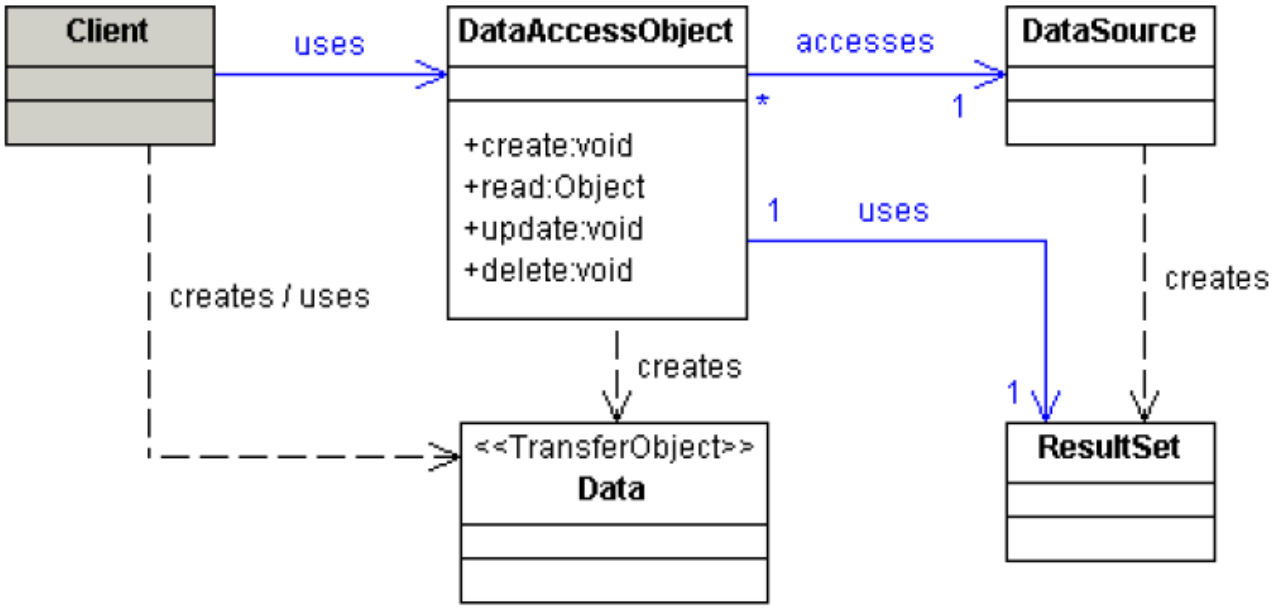
- geringe Netzwerkkosten
- geringe Kosten auf Client Ebene.
- Session Bean stellt eine Transaktions Facade dar, belässt Transaktionen Server seitig, und hält sie kurz => höhere concurrency
- geringers coupling
- Entity Beans bleiben wiederverwendbar
- Bessere Haltbarkeit:
- Saubere “verb-noun” trennung (verb => application logic, noun => persistence logic)

## Data Access Object

Wird verwendet um Daten- Zugriff und Manipulation in einem seperaten Layer zu kapseln.

### Wann?

- Datenzugriffs Mechanismen implementieren um Zugriff und Manipulation auf einem persistenten speicher zu ermöglichen
- Trennen dieser Implementierung vom Rest der Applikation
- Vereinheitlichte Datenzugriffs API für verschiedenste Datenquellen (RDBMS, LDAP, OODB, XML, etc)




### Vorteile

- Zentralisiert die Kontrolle mit loosely coupled handlers
- Ermöglicht Transparenz
- Stellt objekt orientierte Ansicht zur Verfügung, und kapselt Datnebank Schemas
- Ermöglicht einfachere Umwandlung
- Reduziert komplexität des Codes bei Clients
- Organisiert sämtlichen Datenzugriffs-Code in einem seperaten Layer

## EAI – Messageoriented Middleware

### Ein Unternehmen hat viele verschiedene Systeme wie

- ERP
- CRM
- Mainframe
- Financials
- Custom applications

Diese Systeme müssen miteinander funktionieren und effektiv integriert warden in das Unternehmen. Sobald sie integriert sind, unterstützen sie neuere Anwendungsplatformen wie:

- Web application servers
  - B2B exchange systems

### Unternehmen integrieren Anwendungen um

- Etwas besser zu machen
  - Existierende Prozesse automatisieren
- Etwas neues zu machen

- Neue Services zur Verfügung stellen

### Zwei typische Bereiche

- Business-to-business eCommerce (Save Money)
- Customer Care/CRM (Make Money)

Enterprise Application Integration (EAI) ist ein Konzept zur unternehmensweiten Integration der Geschäftsfunktionen entlang der Wertschöpfungskette, die über verschiedene Applikationen auf unterschiedlichen Plattformen verteilt sind, und die im Sinne der Daten- und Geschäftsprozessintegration verbunden werden können.

### Die unterschiedlichen Methoden

- Datenintegration / Enterprise Bus,
- Anwendungsintegration / Message Broker und
- Prozessintegration / Prozessmanagementtool

bauen aufeinander auf.

## Glossar

IIS: Microsoft Internet Information Server

XML: eXtensible Markup Language. Textbasiertes, allgemeines Datenformat, das Grundregeln für den Aufbau eigener Datenformate definiert.

CORBA: Common Object Request Broker Architecture. Konsortialstandard der Object Management Group (OMG) als Richtline zur Implementation von Objektsystemen.

CDO: Collaboration Data Objects. ActiveX Controls, die es erlauben, einfach Mail aus Anwendungen heraus zu versenden.

RDS: Remote Data Service. ADO Technologies, die es erlaubt, Recordsets über Netzwerke zu remoten.

SMTP: Simple Mail Transport Protocol. Standardisiertes Internetprotokoll zum Austausch von EMail.

SOAP: Simple Object Access Protocol. XML-basierter Standard zum Datenaustausch zwischen Systemen.

| | | |
|---|---|---|
| AWT | ... | Abstract Window Toolkit |
| EIS | ... | Enterprise Information System (=Database) |
| ERP | ... | Enterprise Resource Planning |
| IIOP | ... | Internet Inter-ORB Protocol |
| J2EE | ... | Java 2 Platform Enterprise Edition |
| JAF | ... | JavaBeans Activation Framework |
| JDBC | ... | Java Database Connectivity |
| JNDI | ... | Java Naming and Directory Interface |
| JMS | ... | Java Message Service |
| JTA | ... | Java Transaction API |
| JVM | ... | Java Virtual Machine |