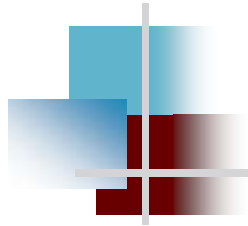


Tema 3.

Tecnologías y marcos de trabajo
para desarrollo de sistemas de
componentes distribuidos

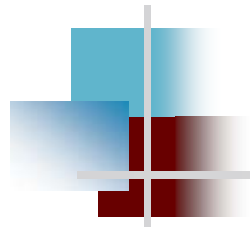
Miguel A. Laguna



Objetivos del tema

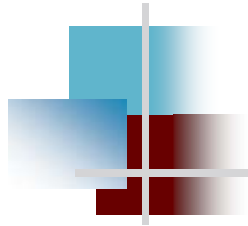
- Aprender:
 - Modelos de Componentes distribuidos
 - Frameworks de componentes distribuidos,
 - En particular JEE en detalle

- En el Laboratorio, ser capaz de:
 - Implementar un sistema utilizando las herramientas y técnicas mostradas en el curso



Contenidos

- Java: EJB y JEE
 - Características y Arquitectura de EJB 3
- Tipos de EJB
 - Stateless y Stateful Session Beans
 - Message driven Beans
- Persistencia: JPA y Entities
- Despliegue de Aplicaciones JEE
 - [Tecnologías de Presentación: JSP y JSF]
 - Despliegue: EAR, WAR y JAR
- [Microsoft DCOM, COM+ y .NET Remoting]
- [Corba Component Model (CCM)]



Bibliografía

EJB/JEE

- 📖 D. Panda, R. Rahman, D. Lane. EJB3 in Action. Manning. 2007. ISBN: 1-933988-34-7.
- 📖 Rima Patel Sriganesh, Gerald Brose, M. Silverman. Mastering Enterprise JavaBeans 3.0. Wiley, 2006.
 - 📖 disponible en <http://www.theserverside.com/>

Lecturas complementarias

- 📖 Oracle/Sun, The JEE Tutorial

.NET

- 📖 Juval Lowy, Programming .NET Components, 2nd Edition O'Reilly. 2005. ISBN-13: 978-0596102074

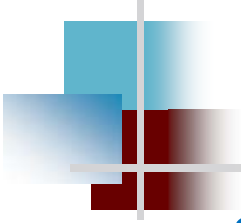
3.1

Tecnología Java: EJB y JEE



- ❑ Características
- ❑ Arquitectura de EJB 3
- ❑ Anotaciones
- ❑ Tecnologías asociadas

Enterprise Java Beans y Java Beans

- 
- Son cosas totalmente distintas!
 - La arquitectura de EJBs proporciona un formato para componentes especializados en la lógica de negocio que se implantan en un entorno JEE
 - Tienen más en común EJBs, Servlets y JSPs que los JavaBeans con cualquiera de ellos

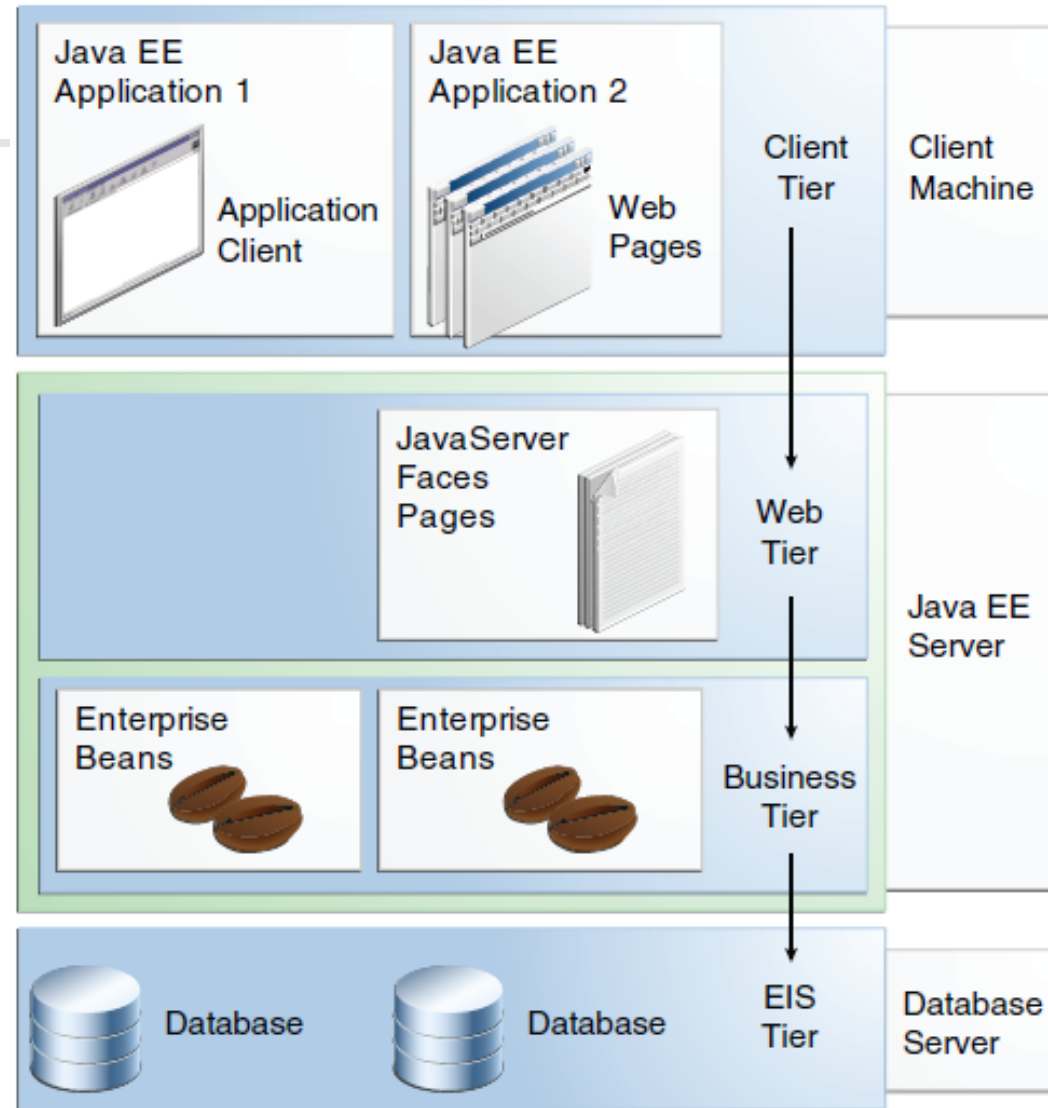


Definiciones

- “Un EJB es un componente de lado del servidor que encapsula la lógica del negocio de una aplicación”.
(Java EE Tutorial)
- “JEE es una plataforma para construir aplicaciones de negocios portables, reutilizables y escalables usando el lenguaje de programación JAVA”
(EJB 3 In action, Debu Panda)

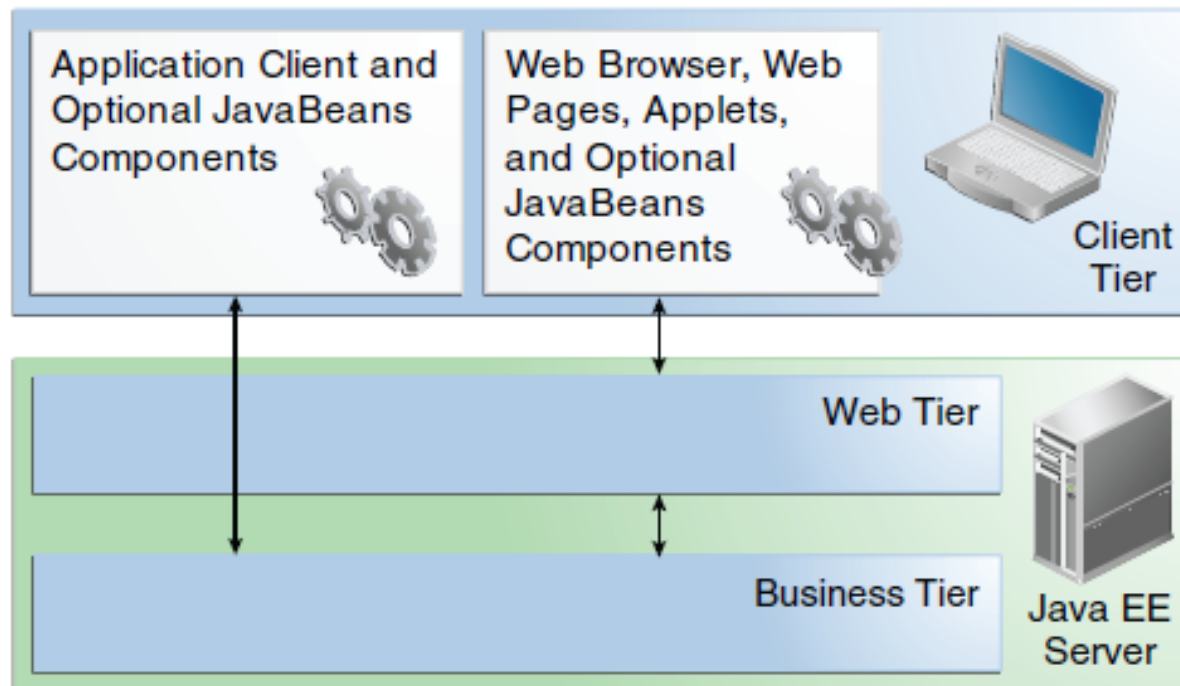
JEE

- Visión global (según Oracle/Sun)



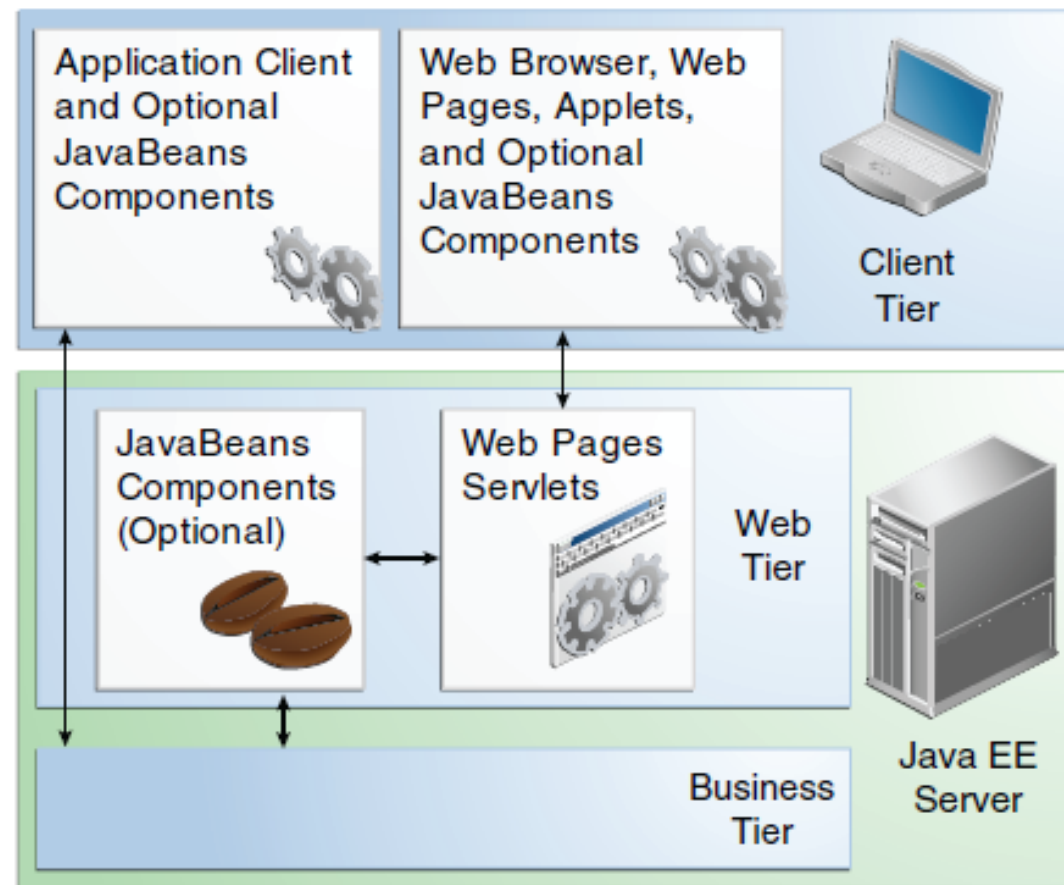
JEE

- La plataforma JEE (Java Enterprise Edition) soporta un modelo de aplicación distribuida multinivel basado en componentes escritos en Java:
 - Componentes cliente: aplicaciones de cliente, web y applets



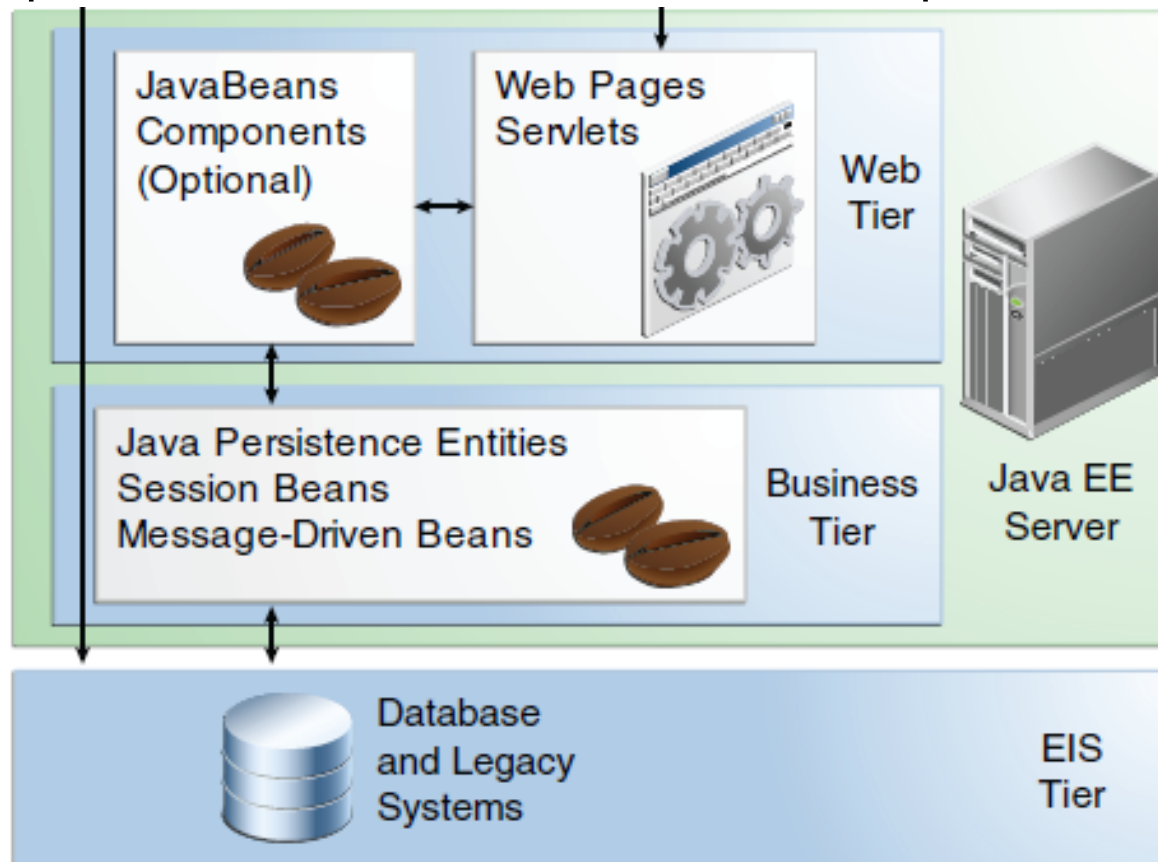
JEE

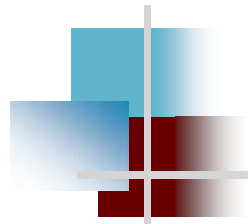
- Componentes web: Servlets y Java Server Pages (JSP/JSF)



JEE

- Componentes de negocio: Enterprise JavaBeans (EJB)
- Componentes del Sistema Información Empresarial (EIS)





Objetivos de JEE

- Definir una arquitectura de **componentes estándar** para la construcción de aplicaciones distribuidas basadas en Java
- **Separar los aspectos** de lógica de negocio de otros soportados por la plataforma: transacciones, seguridad, ejecución multihilo, pooling y otros elementos de bajo nivel
- Filosofía java: Escribir una vez y ejecutar en cualquier parte
- Cubrir los aspectos de desarrollo, despliegue y ejecución del ciclo de vida de una aplicación

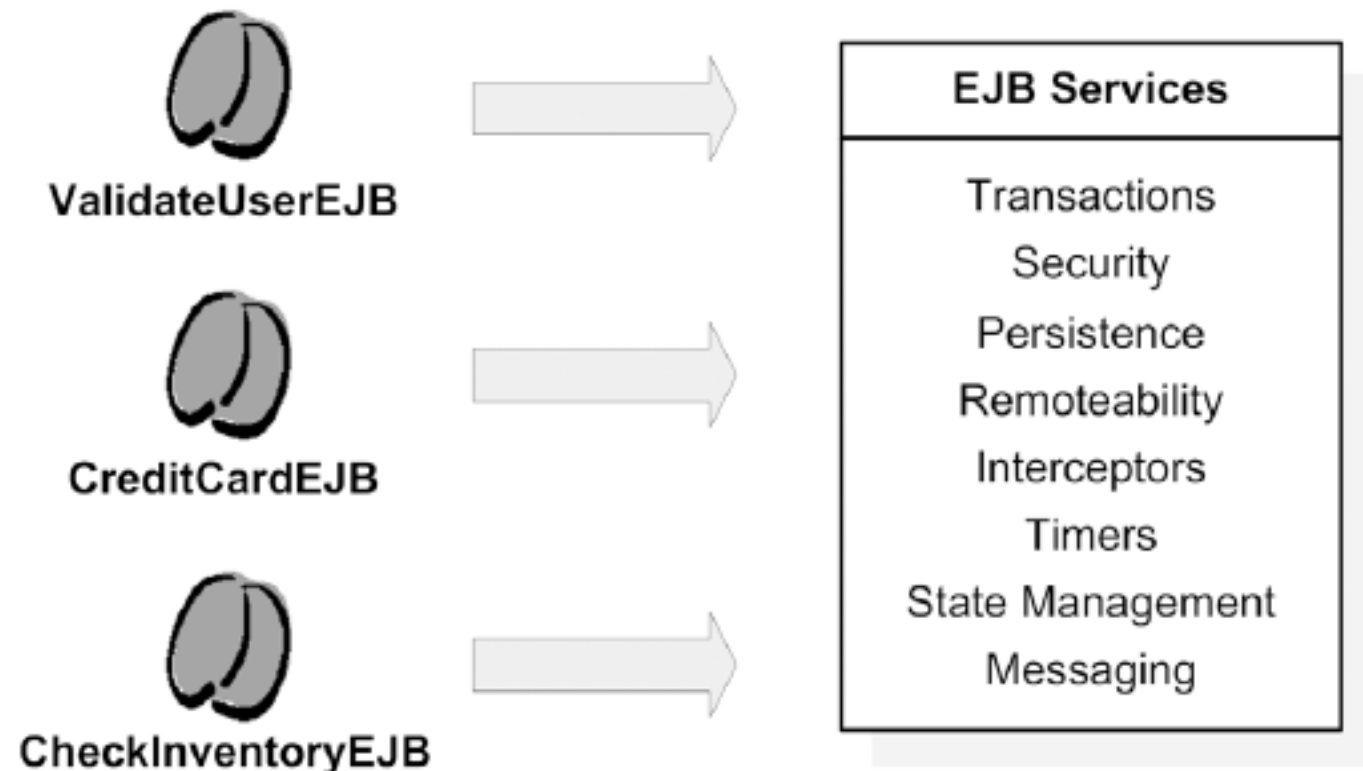


Características de los EJBs

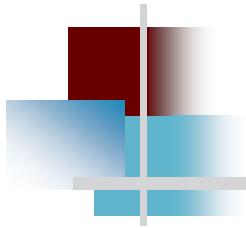
- Contienen lógica de negocio, que opera sobre los datos de la empresa
- Las instancias de un EJB son administradas en ejecución por un **contenedor**
 - El acceso del cliente es mediado por el contenedor donde el EJB está desplegado.
 - El acceso es transparente para el cliente
 - El contenedor asegura que los beans pueden ser desplegados en múltiples ambientes de ejecución sin re-compilación
- Los aspectos como transacción y seguridad, pueden ser especificados:
 - junto con la lógica del negocio de la clase EJB en forma de **anotaciones**
 - en un descriptor de despliegue **XML**

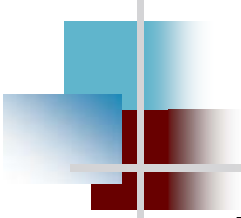
Framework EJB

- EJB (+ el contenedor) es un framework además de una plataforma de componentes...



Arquitectura de EJB

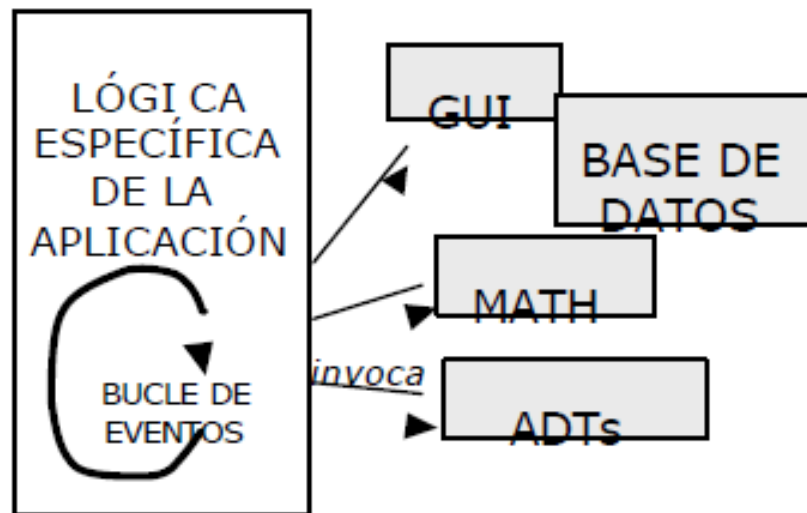




Arquitectura basada en contenedores

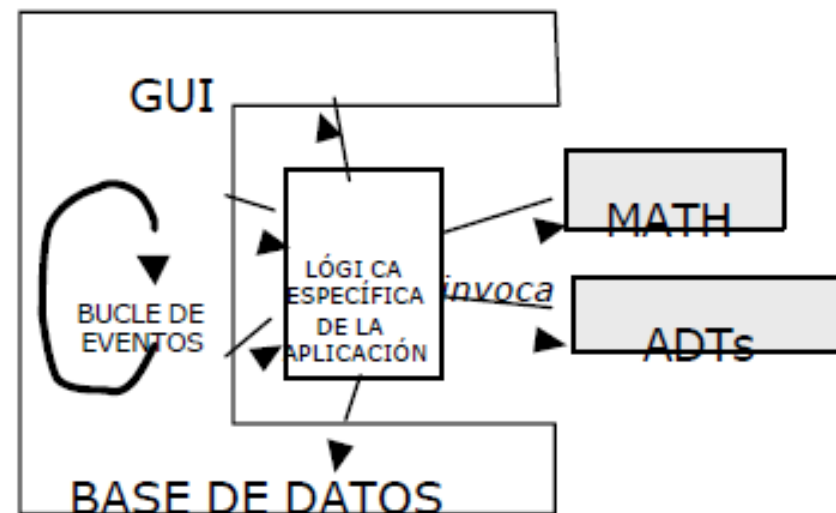
- Un contenedor **es un proceso** donde se ejecutan los componentes
 - Gestiona los componentes de la aplicación (Ciclo de vida)
 - Proporciona acceso a servicios de la plataforma: transacciones, seguridad, conectividad
- El desarrollador tiene que especificar dos tipos de elementos:
 - Los componentes de la aplicación
 - JSPs (Java Server Pages) [o JSF], Servlets
 - EJBs (Enterprise Java Beans)
 - Los descriptores de despliegue (deployment)
 - Ficheros XML que describen los componentes de aplicación

Bibliotecas y contenedores/frameworks



Arquitectura basada en biblioteca de clases

→ reutilización de código

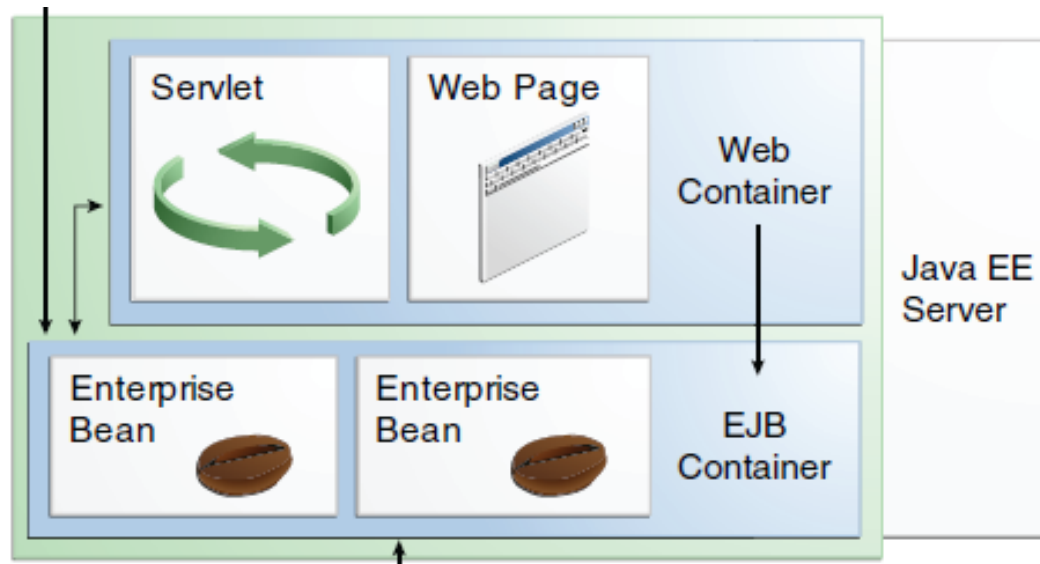


Arquitectura basada en Contenedor (armazón)

→ reutilización de diseño y código

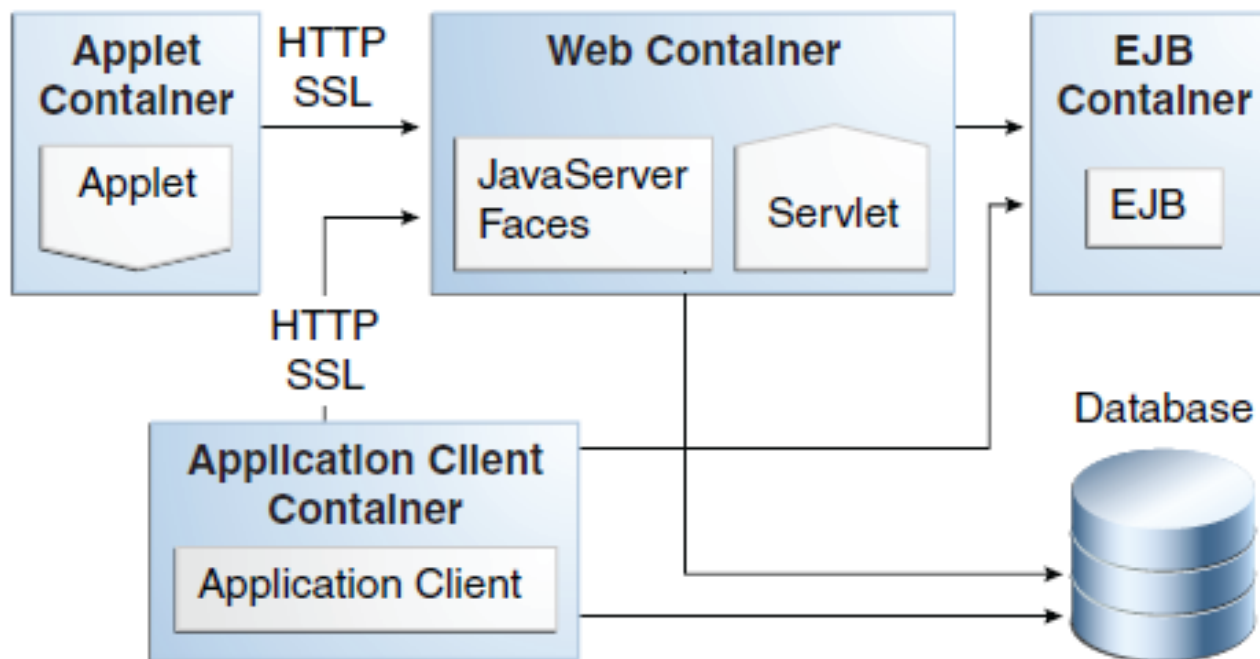
Contenedores Web y EJB

- Dos tipos:
 - Contenedores Web
 - Contenedores EJB

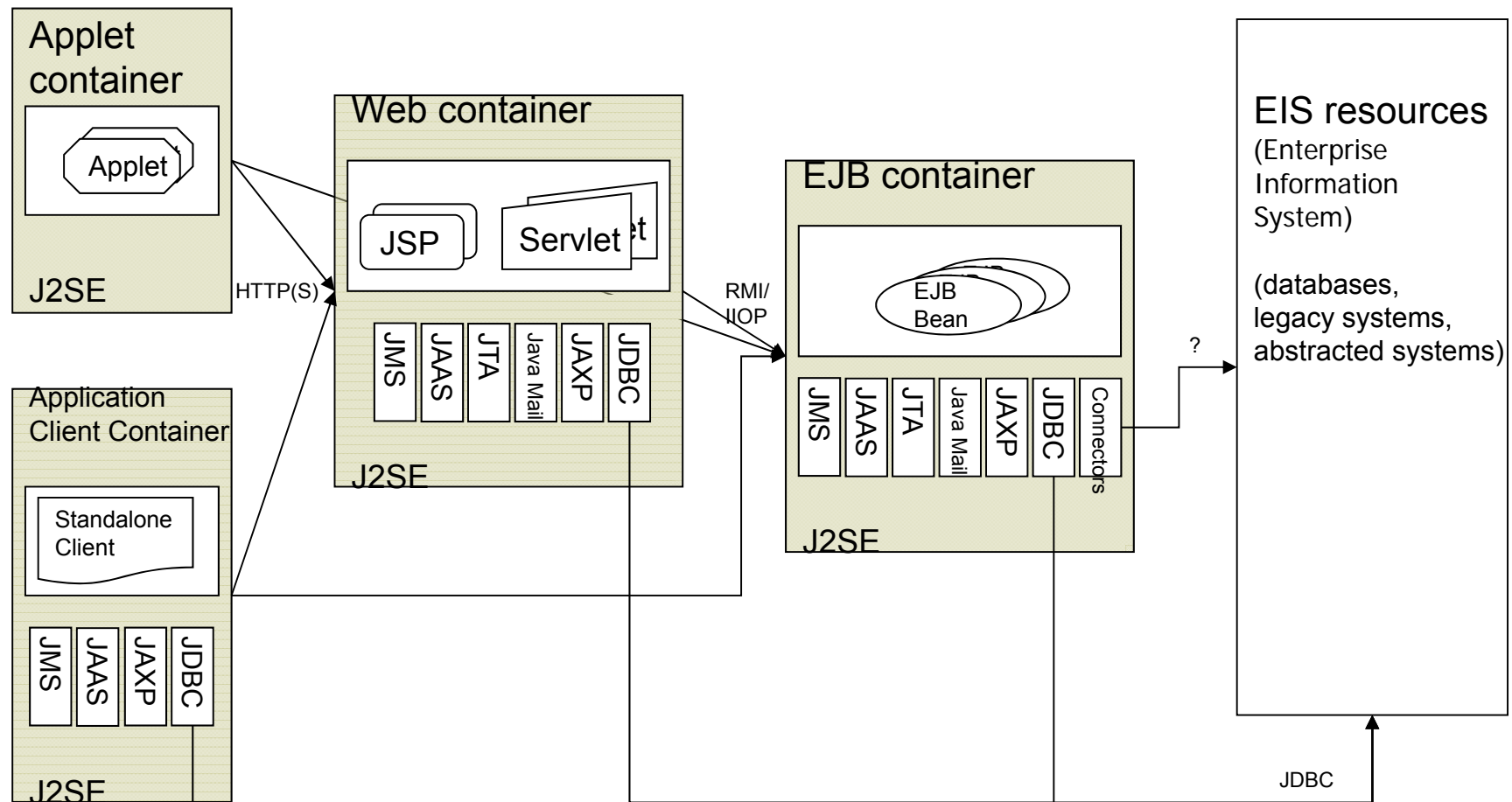


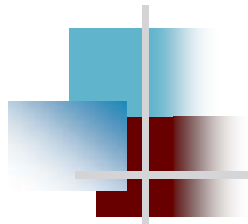
Conexión entre contenedores

- JMV y los navegadores son también contenedores



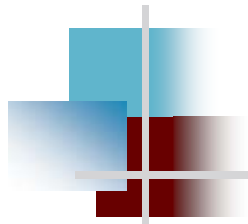
Contenedores JEE: APIs Java



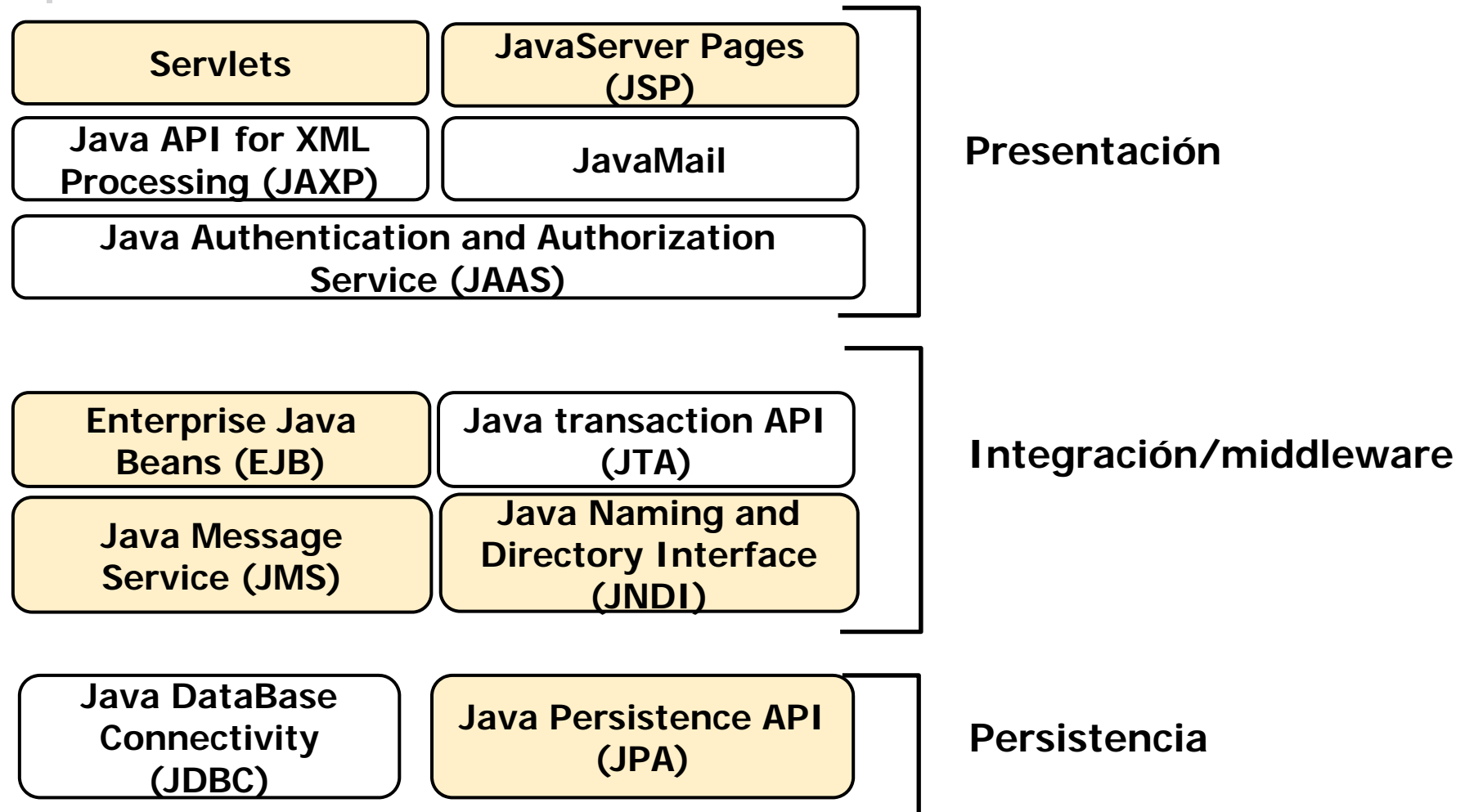


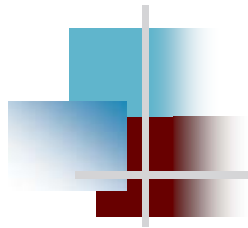
Servicios Java EE

- Para cada contenedor Java EE proporciona una serie de servicios (en forma de APIs), como por ejemplo:
 - Java Beans Active Framework (JAF)
 - Servicios Web (JAXWS)
 - Java EE Connector Architecture
- Java Transaction API (JTA)
- Java Persistence API (JPA)
- Java Message Service (JMS)
- Java Naming Direct Interface (JNDI)
- JavaMail
- Java API for XML Procesing (JAXP)
- Java Authentication and Authorization Service (JAAS)

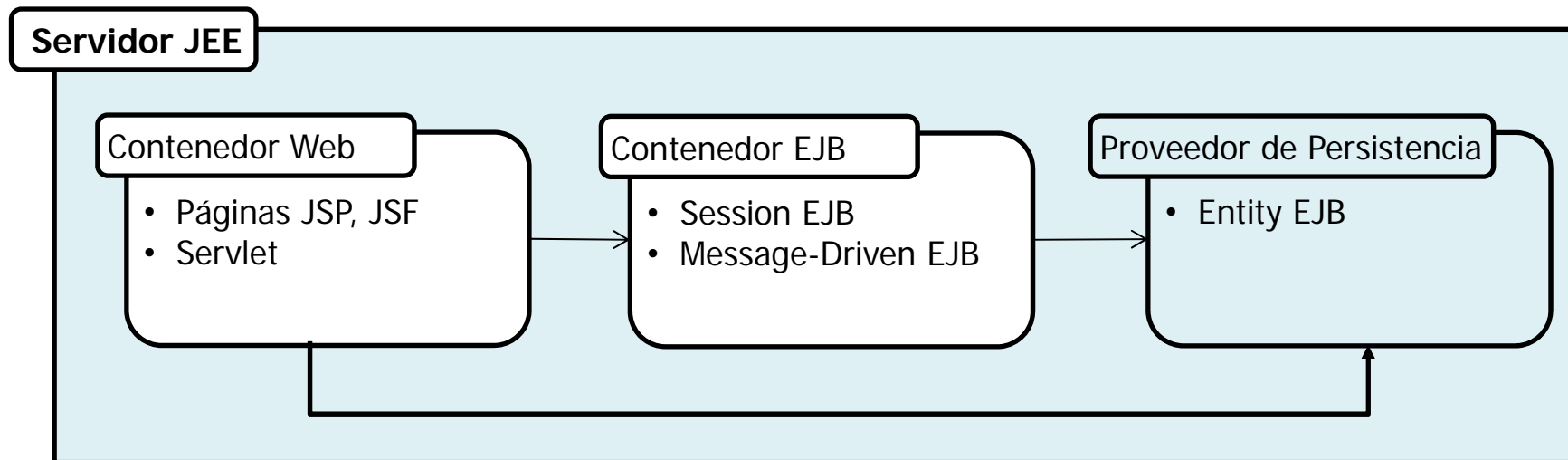


Java APIs



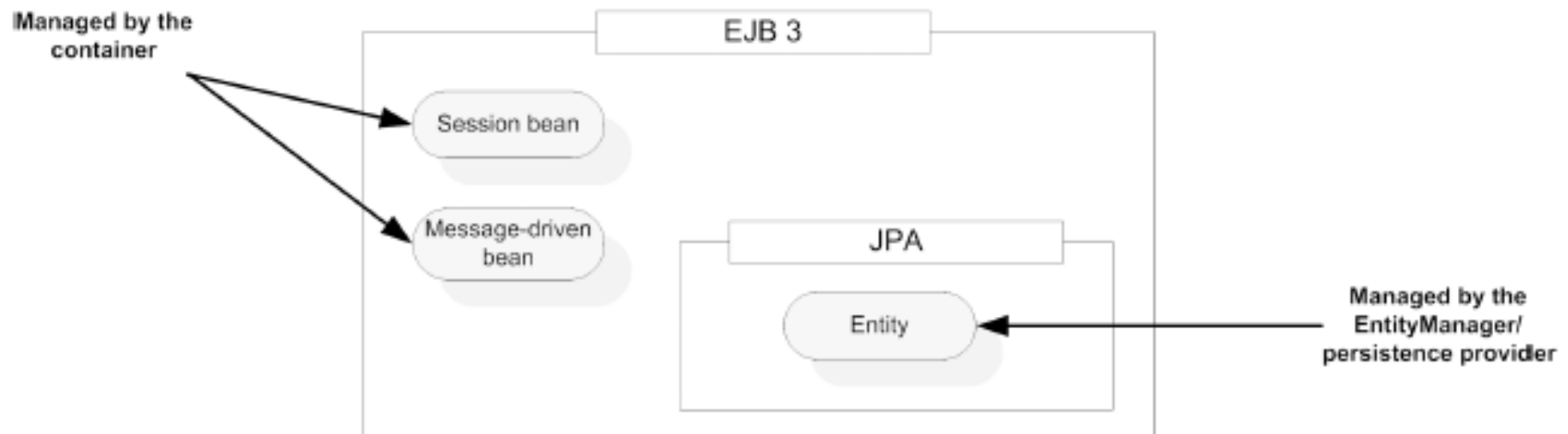


Servidor tipo: GlassFish



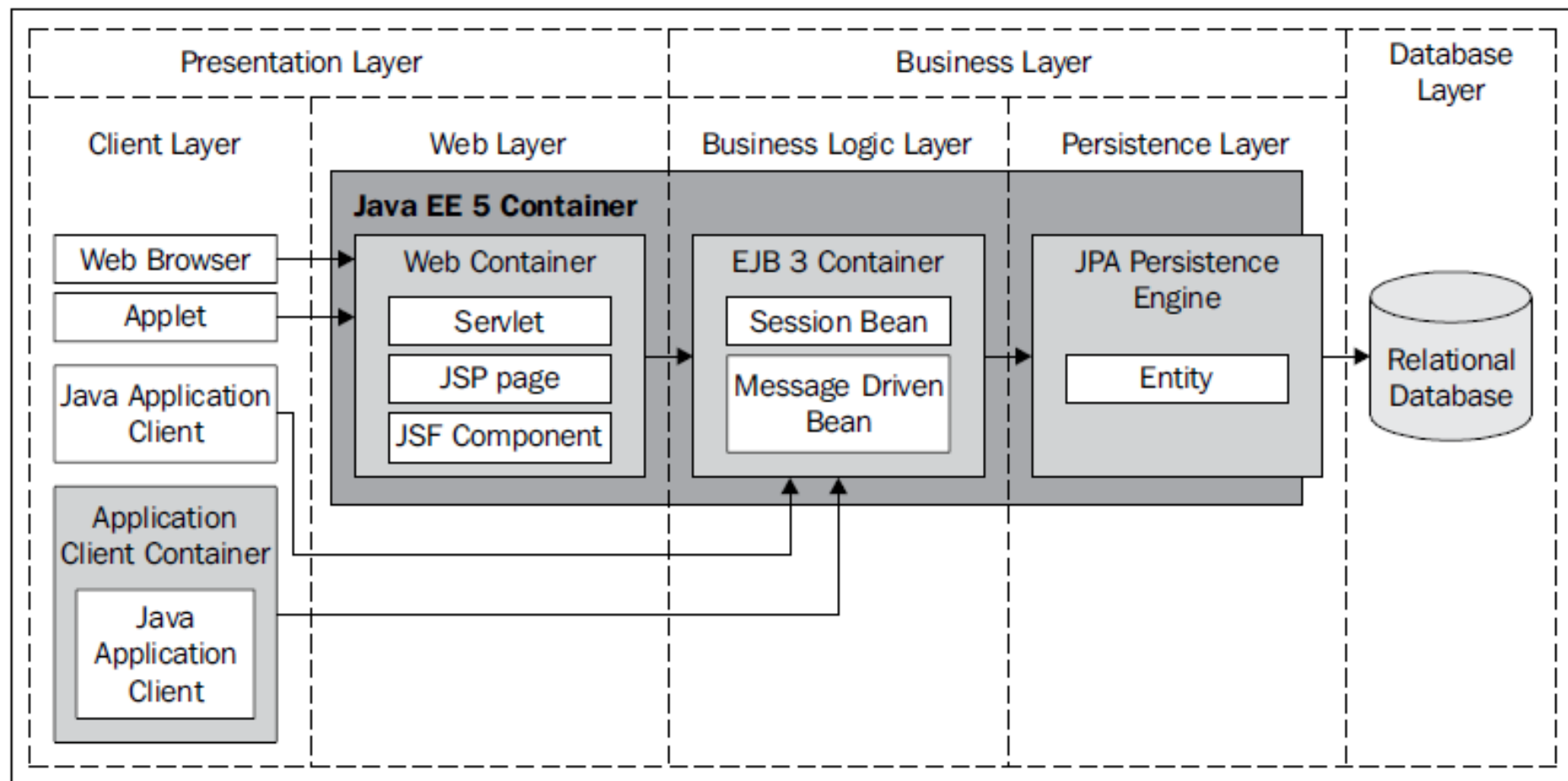
JPA: Proveedor de Persistencia

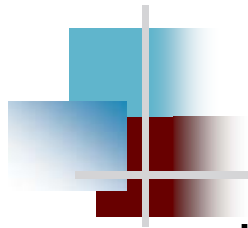
- A partir de la versión 3 se recomienda usar la API de persistencia estándar de Java en lugar de EJB de tipo entity



Niveles:

Recomendación Sun





Estructura de despliegue

- Una aplicación EJB debe contener:
 - Componentes EJB
 - Interfaces que definen los métodos que implementan los componentes
 - Clases "helper": clases java de utilidad requeridas por los EJB (cálculos, DTOs, ...etc)
- Se empaquetan en un archivo .jar, son portables y pueden ser empaquetados a su vez [posiblemente junto con archivos Web en un .war] en un archivo .ear.



Anotaciones y meta-datos



EJB 2.1: Despliegue con descriptores

```
public class CartEJB implements SessionBean
{
    protected Collection items = new ArrayList();
    public void add(String item)
    {
        items.add(item);
    }
    public Collection getItems()
    {
```

```
        <session>
            <display-name>Shopping Cart</display-name>
            <ejb-name>MyCart</ejb-name>
            <home>CartHome</home>
            <remote>Cart</remote>
            <ejb-class>CartEJB</ejb-class>
            <session-type>Stateful</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    }
```

EJB 3: anotaciones

- POJO (Plain Old Java Object)



POJO



Annotation



EJB

@Stateless

@Stateful

@MessageDriven

@Entity



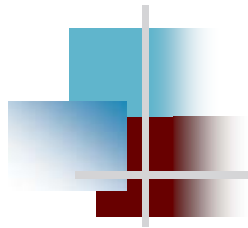
El mismo ejemplo

@Remote

```
public interface Cart {  
    public void addItem(String item);  
    public void completeOrder();  
    public Collection.getItems();  
}
```

@Stateful

```
public class CartBean implements Cart {  
    private ArrayList items;  
  
    public void add(String item) {  
        items.add(item);  
    }  
  
    public Collection.getItems() {  
        return items;  
    }  
    @Remove  
    public void completeOrder()  
    {  
    }  
}
```



Anotaciones

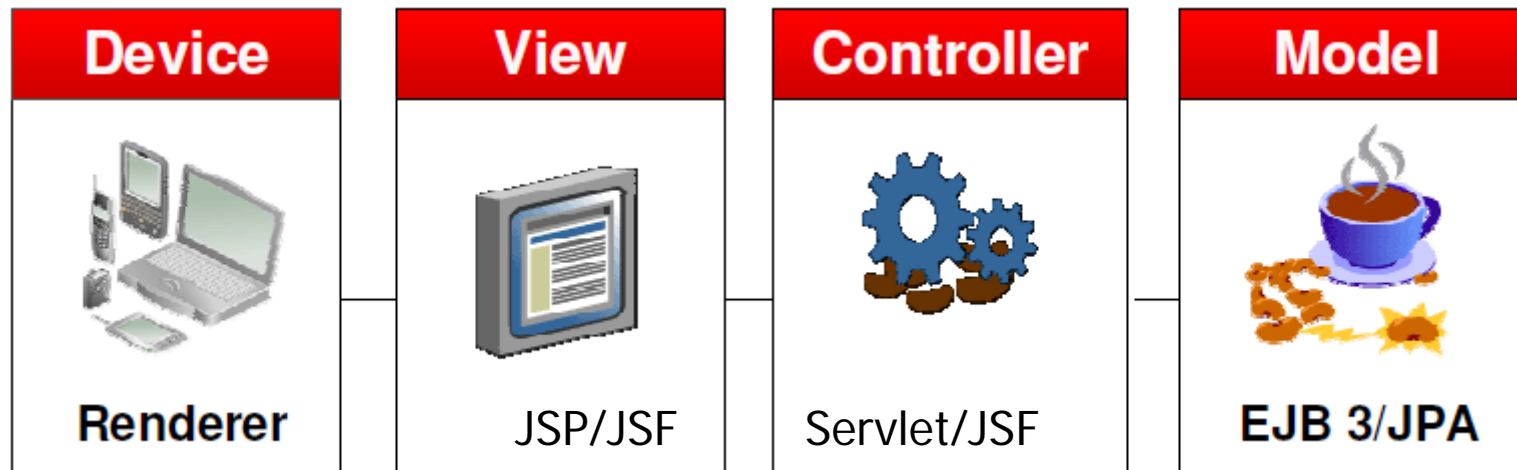
- Forma de simplificar la definición del EJB.
 - `@Stateful`
 - `@Stateless`

```
@Stateless
public class PlaceBidBean implements PlaceBid {
    @Interceptors(ActionBazaarLogger.class)
    public void addBid(Bid bid) {
        ...
    }
}
```

- Callbacks: Posibilidad de emplear AOP
 - `@PostConstruct`
 - `@PreDestroy`
 - `@PreActivate`
 - `@PostActivate`

Tecnologías asociadas: MVC

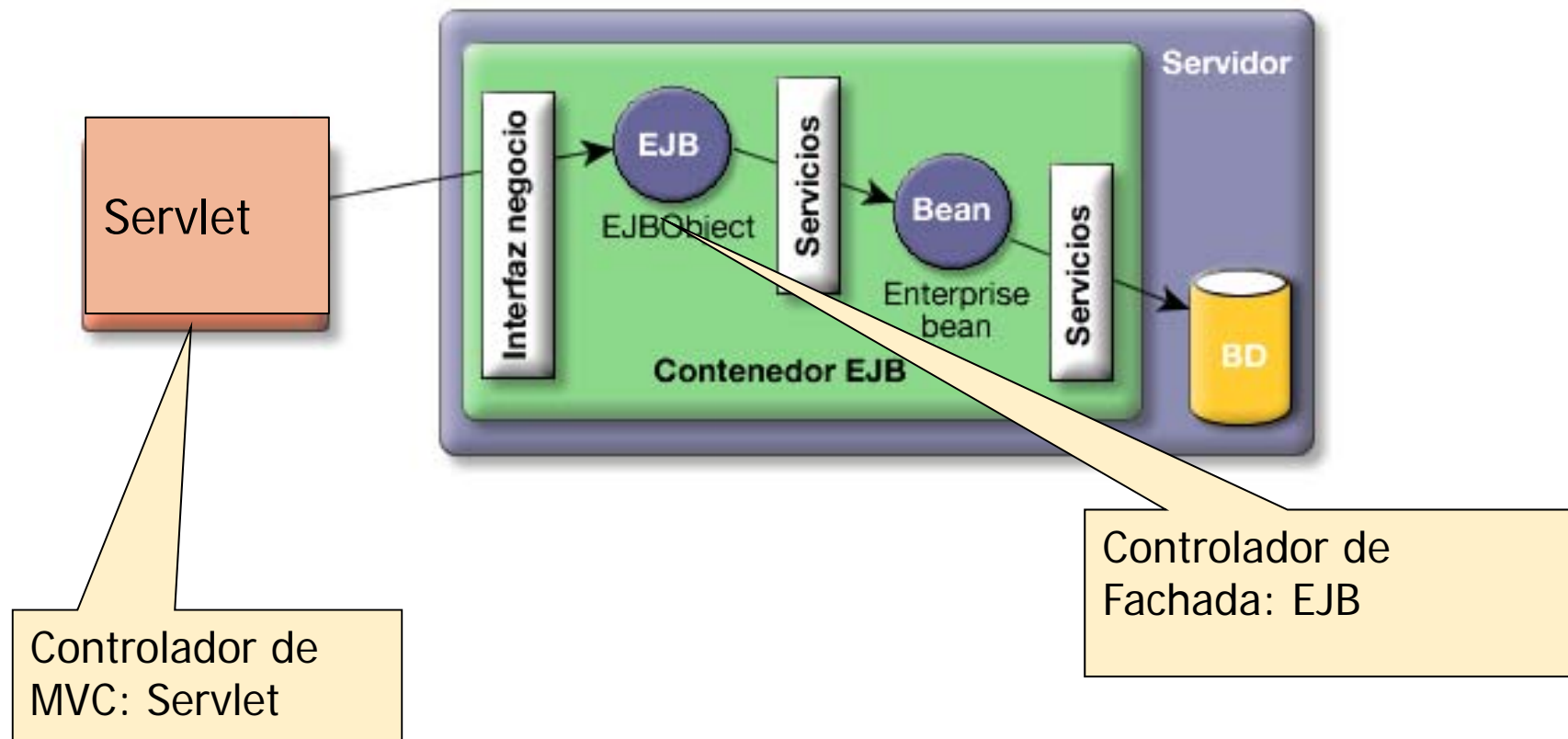
- Visión MVC: Modelo Vista Controlador
 - Vista: JSP/JSF
 - Controlador: Servlet/JSF [Framework Struts]
 - Modelo: EJB y JPA



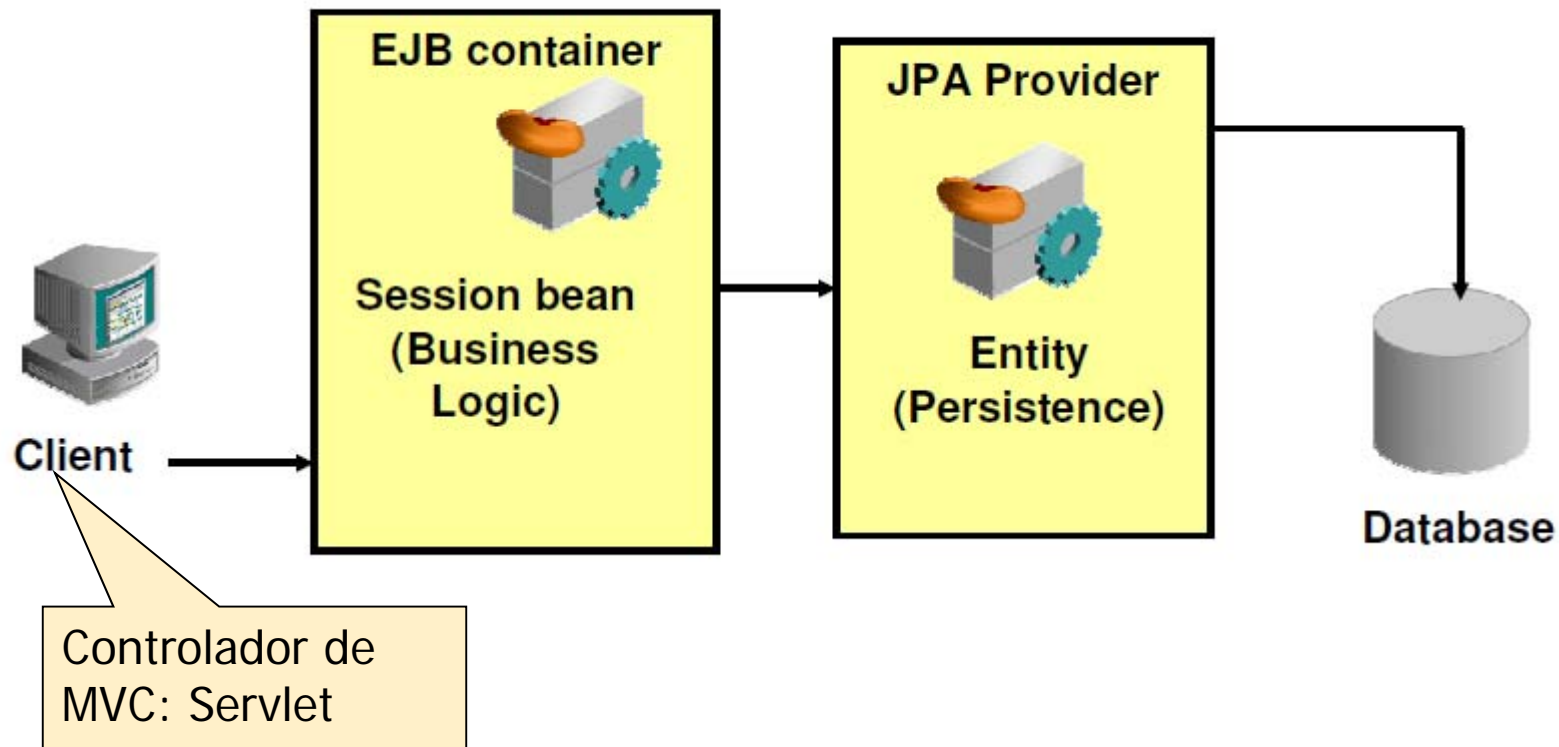
Modelo:

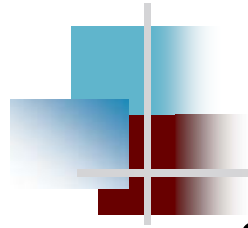
Controlador de fachada: EJB

- La conexión con el modelo es a través de un controlador (Larman) de fachada que implementa la interfaz pública del componente



Modelo: Persistencia



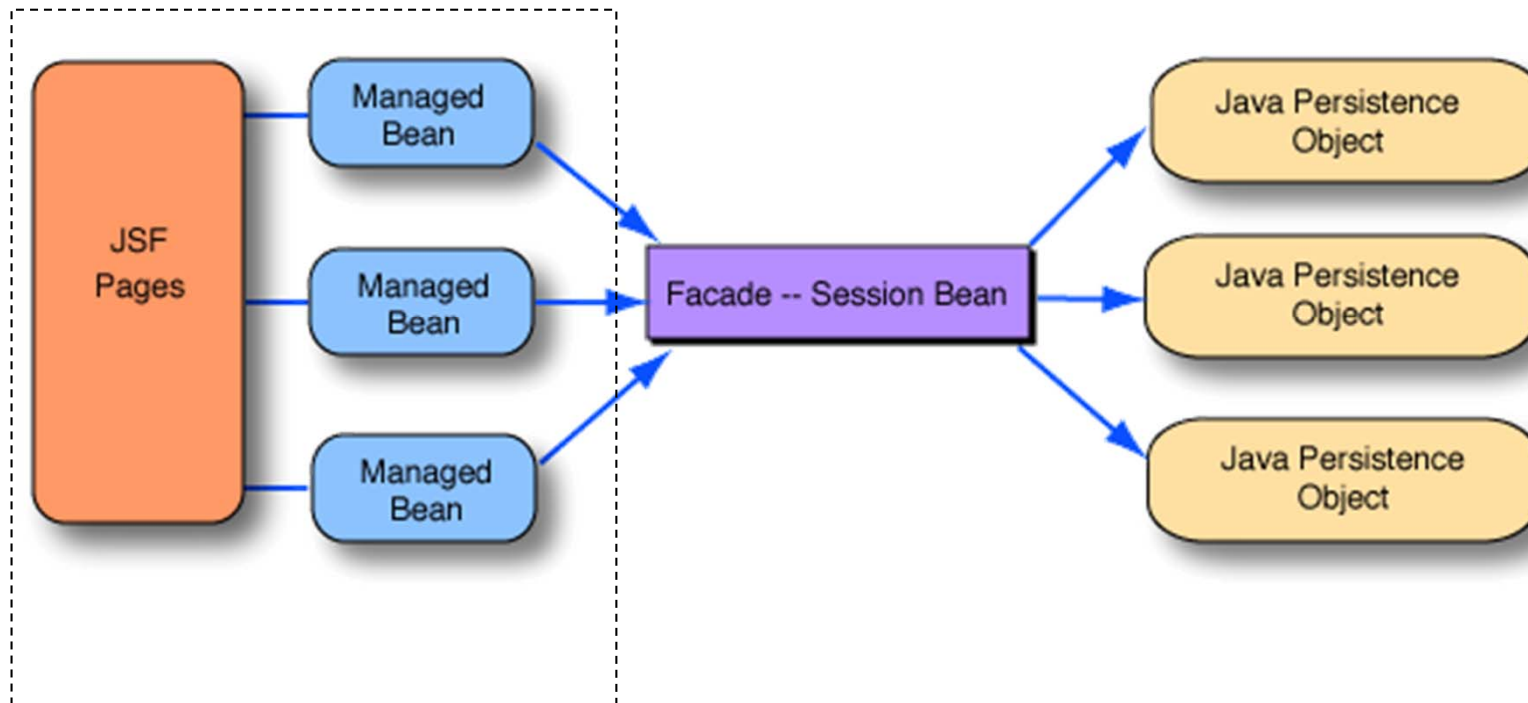



JPA (Java Persistence API)

- Simplificación de persistencia gestionada por contenedor
 - Enfoque POJO / JavaBeans
 - Permitir el uso de las entidades fuera del contenedor
 - Contenedor web
 - Java SE
- Soporte para el modelo de dominio
 - Herencia y polimorfismo
 - Metadatos para el mapeo objeto-relacional

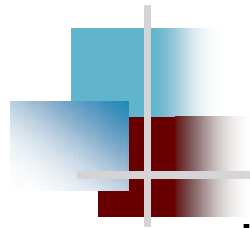
Tecnologías Java EE

- JSF: propuesto por Sun como alternativa a JSP/Servlet (Dominio y Persistencia son comunes)



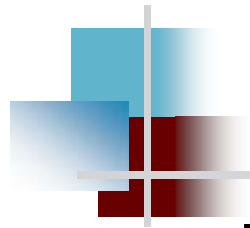


Ventajas de los EJB



Ventajas

- Los EJB son componentes portables, reutilizables y pueden ser desplegados en servidores que usen los estándares de las APIs JEE
- Simplifican desarrollo, el contenedor EJB es responsable de la administración de transacciones y autorizaciones de seguridad
- La lógica del negocio reside en los EJBs y no en el lado del cliente, permitiendo que el desarrollo del lado del cliente esté desacoplado de la lógica del negocio



Ventajas

- Distribución de componentes a través de múltiples máquinas
 - Pueden residir en diferentes servidores y pueden ser invocados por un cliente remoto.
 - Multi-usuario, locales y remotos
- Aplicaciones escalables
- Aseguramiento de integridad de los datos de las transacciones.
 - Los EJBs soportan transacciones y el mecanismo que administra el acceso concurrente de objetos compartidos

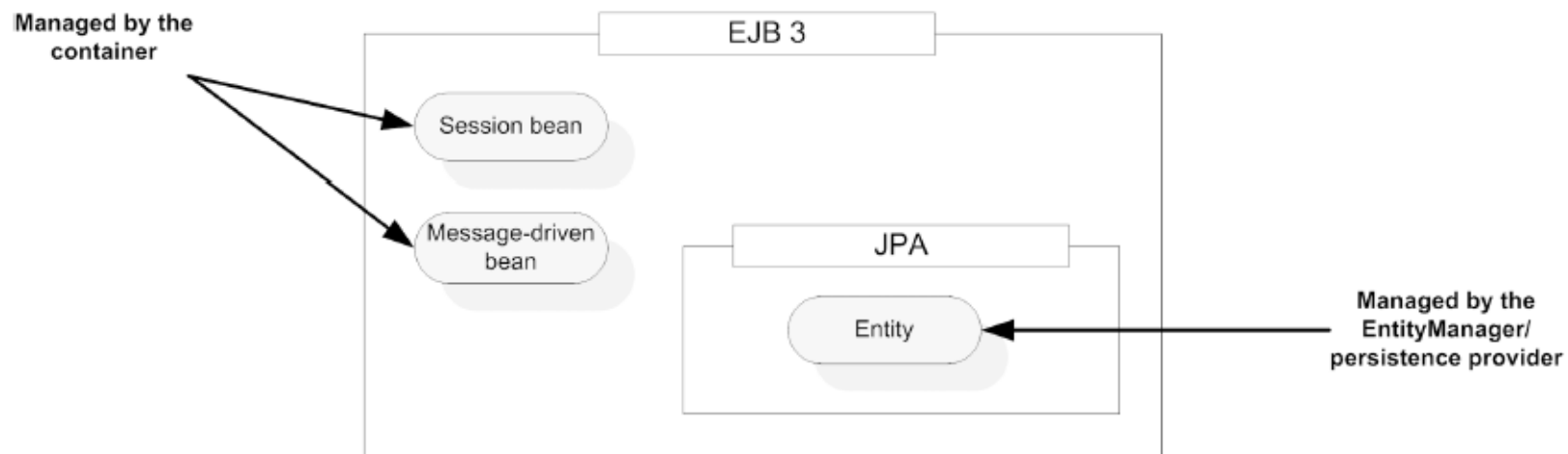
3.2.

Tipos de componentes EJB

- Session beans
- Message-driven beans
- [Entity]

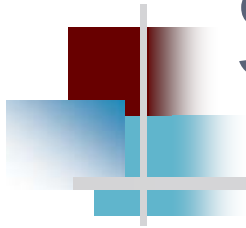
Tipos de componentes EJB 3

- Session beans
- Message-driven beans
- [Entity]



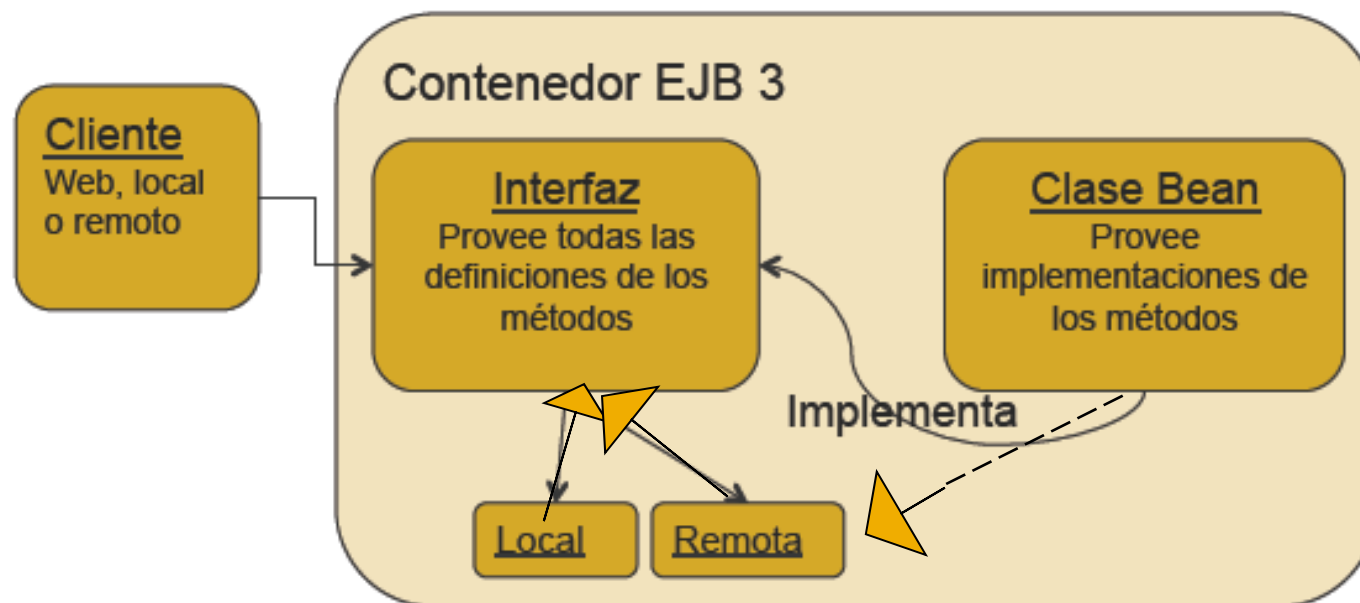
3.2.1

Session Beans



Session Bean

- Objetivo: encapsular los procesos de negocio (la interfaz del sistema en los DSS encontrada a partir de los casos de uso)



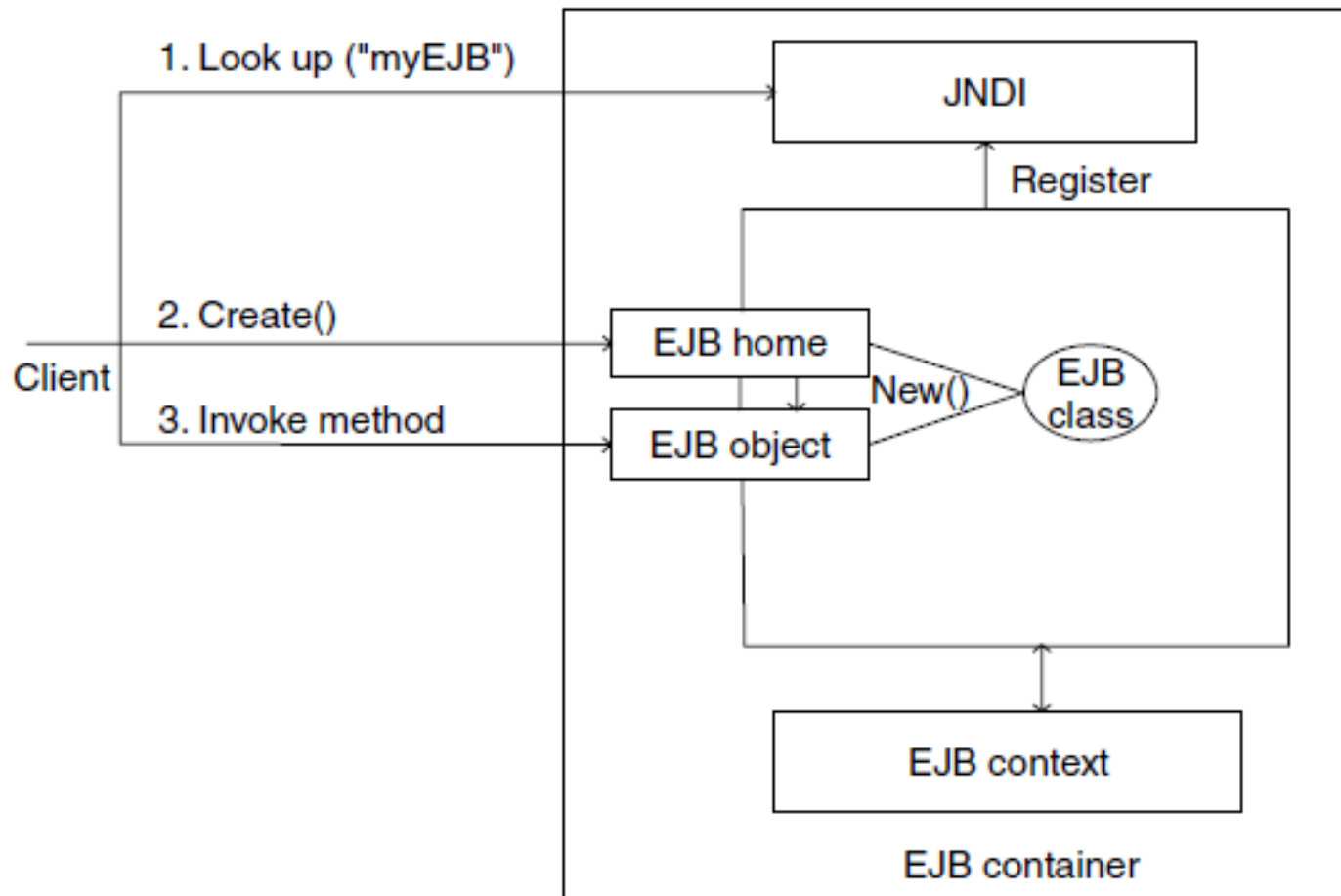
Session Bean:

Características

- Vida corta, si el servidor falla la sesión se pierde
- No se comparte entre clientes
- No manejan directamente la persistencia
- Pueden actualizar y crear "entities", estas últimas son persistentes
- Un cliente (Servlet o aplicación Java) interacciona con un Session Bean a través de la invocación de sus métodos (la invocación se llama sesión)

Middleware: JNDI y contexto

- 1 y 2 son transparentes usando anotaciones en EJB 3.0



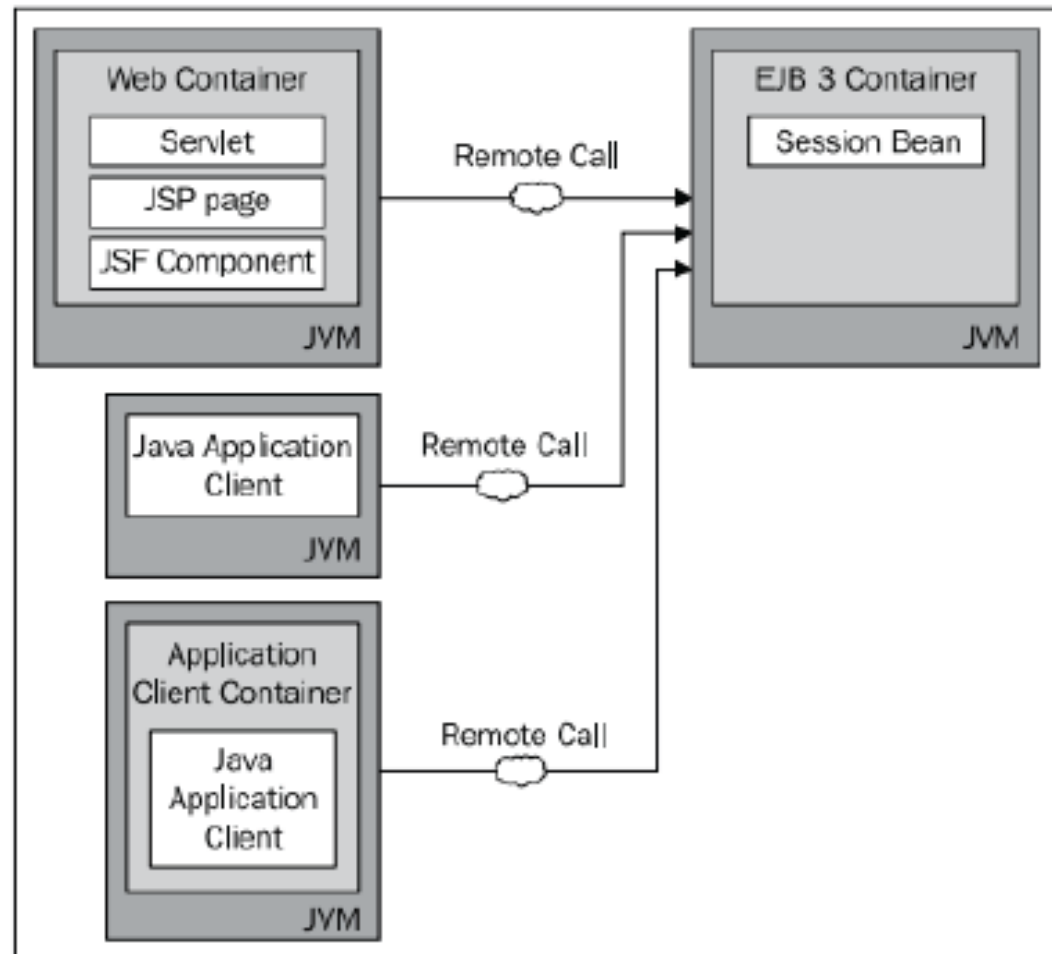


Session Bean: Características

- Un Session Bean está compuesto por una o más interfaces y una clase de implementación (POJO!)
- Un cliente puede acceder a un Session Bean solamente a través de métodos definidos en la interfaz del Bean.
 - La interfaz define la vista del cliente de un Session Bean
- Un Session Bean puede ser invocado a través de una interfaz:
 - Local
 - Remota, a través de RMI
 - Web Service !

Invocación remota

- El cliente remoto se ejecuta en una JVM diferente
- Un cliente remoto puede ser un componente web, una aplicación cliente u otro EJB.
- Para el cliente remoto la ubicación del EJB es transparente





Definición de interfaz remota

- Definir la interfaz anotada con @Remote
- Definir la clase @Stateless o @Stateful que implementa la interfaz

@Remote

```
public interface HelloUser {  
    public void sayHello(String name);  
}
```

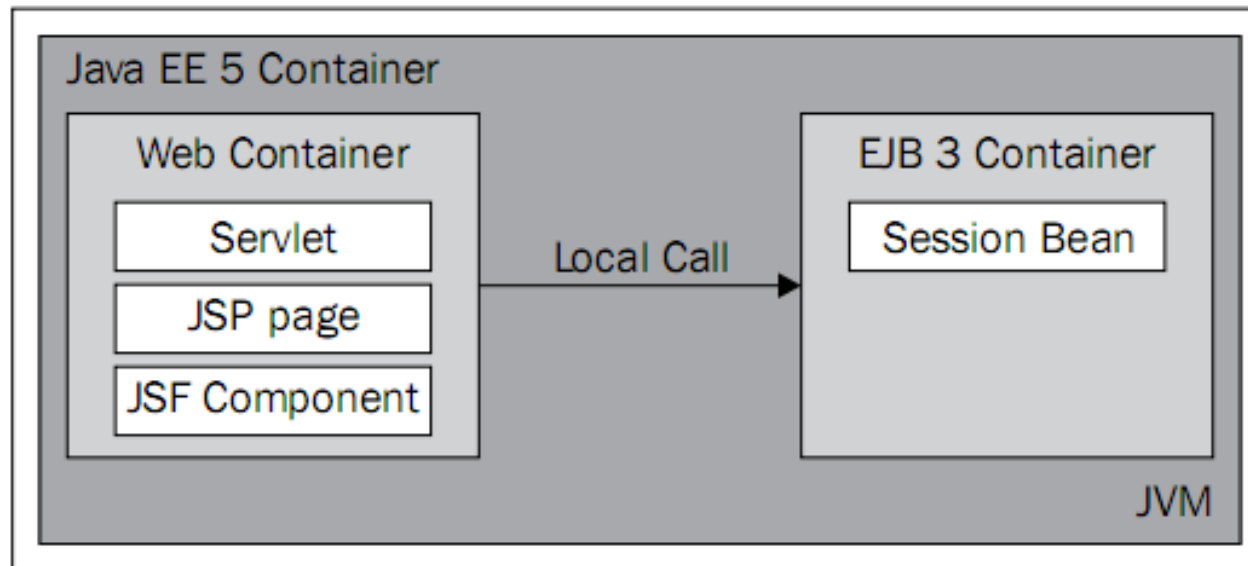
import javax.ejb.Stateless;

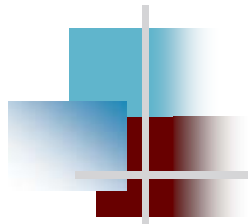
@Stateless

```
public class HelloUserBean implements HelloUser {  
    public void sayHello(String name) {  
        System.out.println("Hello " + name + ", welcome");  
    }  
}
```


Invocación Local

- El cliente reside en la misma instancia del Session Bean.





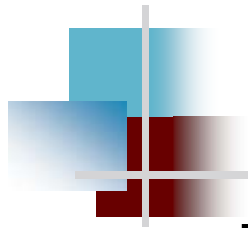
La interfaz local

- La interfaz por defecto es local
- El cliente local debe correr en la misma JVM que los EJBs a los que accede
- El cliente local puede ser un componente web u otro EJB
- Definición de un Session Bean con interfaz local.
 - Definir la interfaz anotada con `@Local`
 - Definir la clase `@Stateless` o `@Stateful` que implementa la interfaz



Estado de un Session Bean

- El estado de un objeto se compone de los valores de sus variables de instancia.
 - Los valores de las variables representan el estado de una única sesión cliente/Bean.
- El estado de la interacción del cliente con el Bean es llamado **estado conversacional**.
- Modos de estado conversacional
 - Stateful
 - Stateless

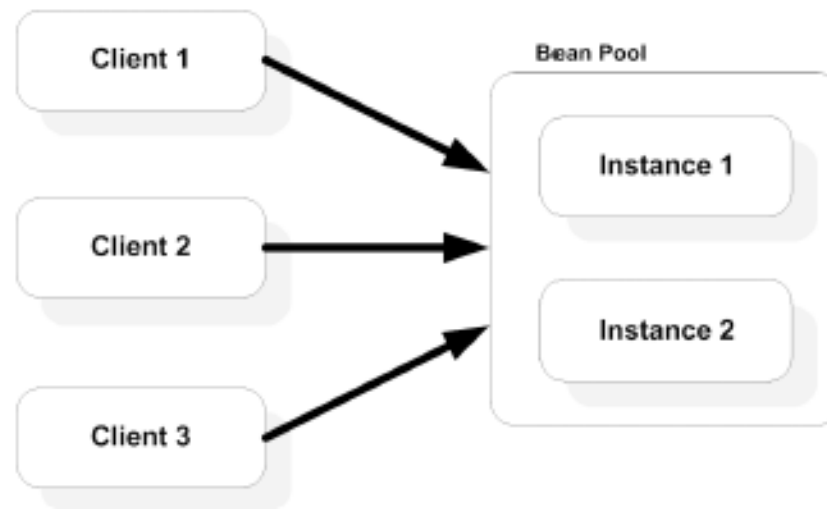


Stateless Session Bean

- No mantiene el estado conversacional con el cliente.
 - Cuando un cliente invoca los métodos de un Stateless Bean, las variables de instancia del Bean pueden contener un estado específico del cliente durante la invocación.
 - Si el método finaliza, el estado para el cliente específico se pierde
- Ofrecen mejor escalabilidad para aplicaciones con gran cantidad de clientes
- Puede servir para implementar un Web Service

Stateless Session Bean

- Las instancias están compartidas por los clientes.
- El contenedor tiene un pool de instancias:
 - cuando el cliente invoca un método se asigna una instancia
 - cuando termina la libera y vuelve al pool.



Ciclo de vida: Stateless Bean

- El cliente inicia el ciclo de vida obteniendo una referencia al Session Bean





Callback Session Bean

- Los métodos Callback son métodos del Bean no expuestos en la interfaz que reflejan una transición del ciclo de vida de un Bean
 - Cuando el evento ocurre el contenedor invoca al método Callback correspondiente y los métodos pueden ser utilizados para mejorar rendimiento
- Estos métodos son marcados con anotaciones como `@PostConstruct` y `@PreDestroy` (para los stateful Session Bean se agregan `@PrePassivate` y `@PostActivate`)

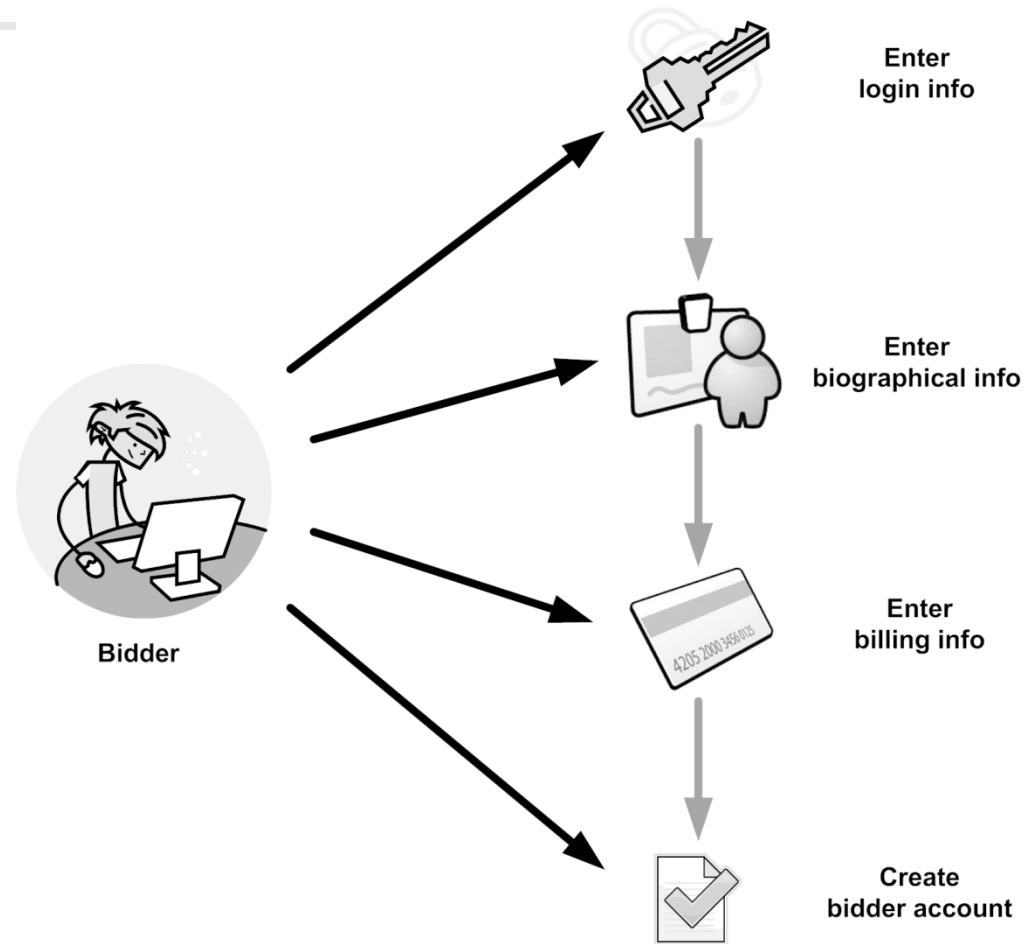
Stateful Session Bean

- El estado se mantiene durante la sesión del cliente con el Bean
- La instancia es reservada para el cliente y cada una almacena la información del cliente.
- La sesión finaliza si el cliente elimina el Bean (metodo @Remove) o finaliza su sesión (Time Out).

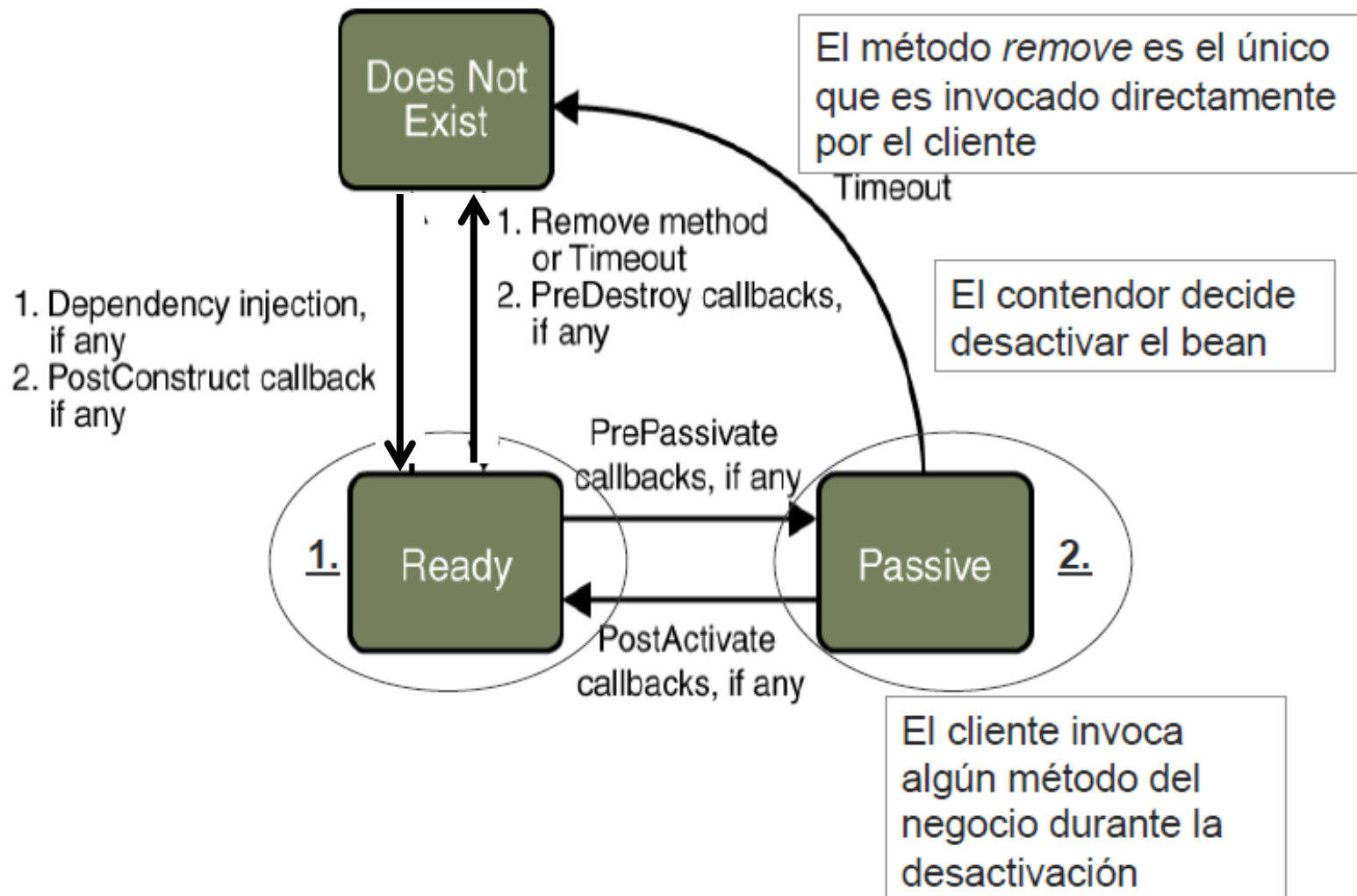


Stateful Session Bean

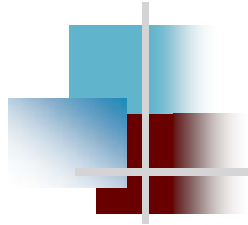
- Llamadas consecutivas a métodos del mismo Bean
- (Similar a las transacciones en Bases de Datos)



Ciclo de vida Stateful Bean



Resumen de Anotaciones Callback



@PostConstruct: invocado sobre una instancia recientemente creada después de la inyección (o JNDI lookup) de todas las dependencias y antes de la invocación del primer método (`javax.annotation.PostConstruct`)

@PreDestroy: invocado luego de que un método anotado con `@Remove` ha terminado y antes de que el contenedor elimine la instancia del bean (`javax.annotation.PreDestroy`)

@PrePassivate: invocado antes de que el contenedor desactive (passivate) el bean, el contenedor elimina temporalmente el bean y lo salva en memoria secundaria (`javax.ejb.PrePassivate`)

@PostActivate: invocado después de que el contenedor mueve el bean de memoria secundaria a estado activo (active) (`javax.ejb.PostActivate`)



Ejemplos de uso

@Remove

```
public void cancelAccountCreation() {...}
```

@Remove

```
public void createAccount() {...}
```

@PrePassivate

@PreDestroy

```
public void cleanup() {...}
```

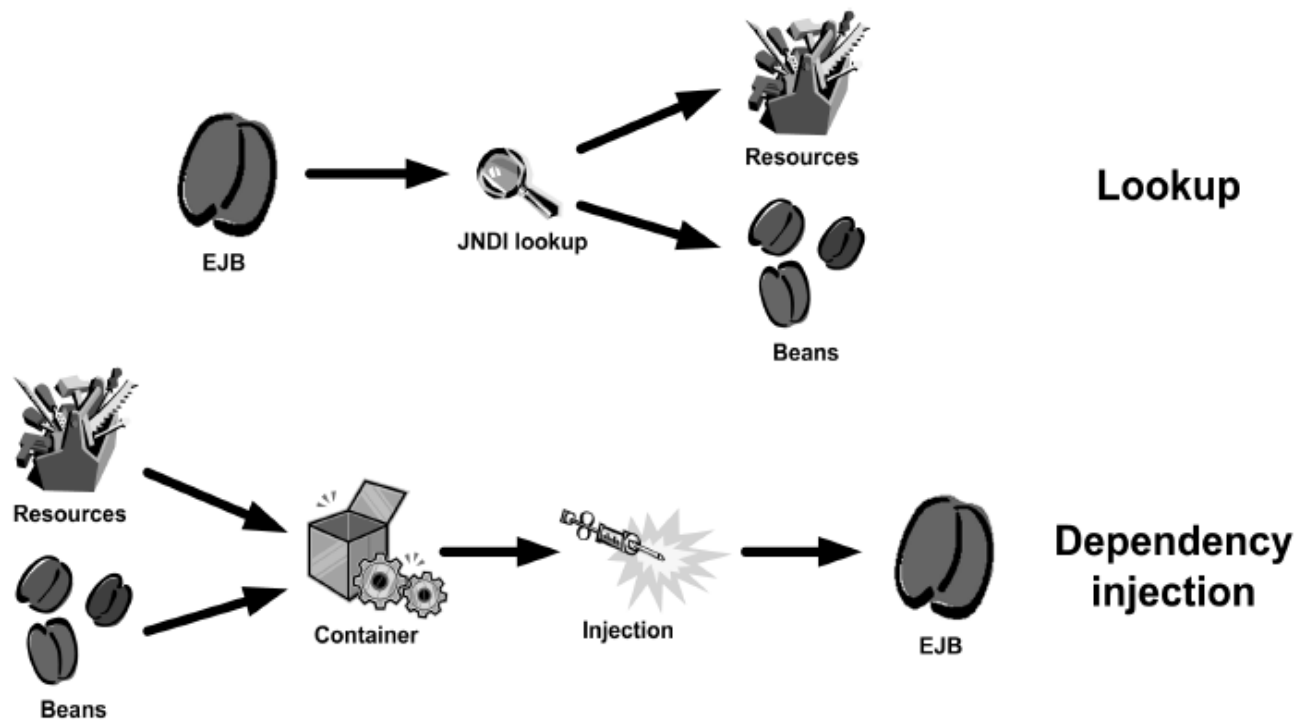
@PostConstruct

@PostActivate

```
public void openConnection() {...}
```

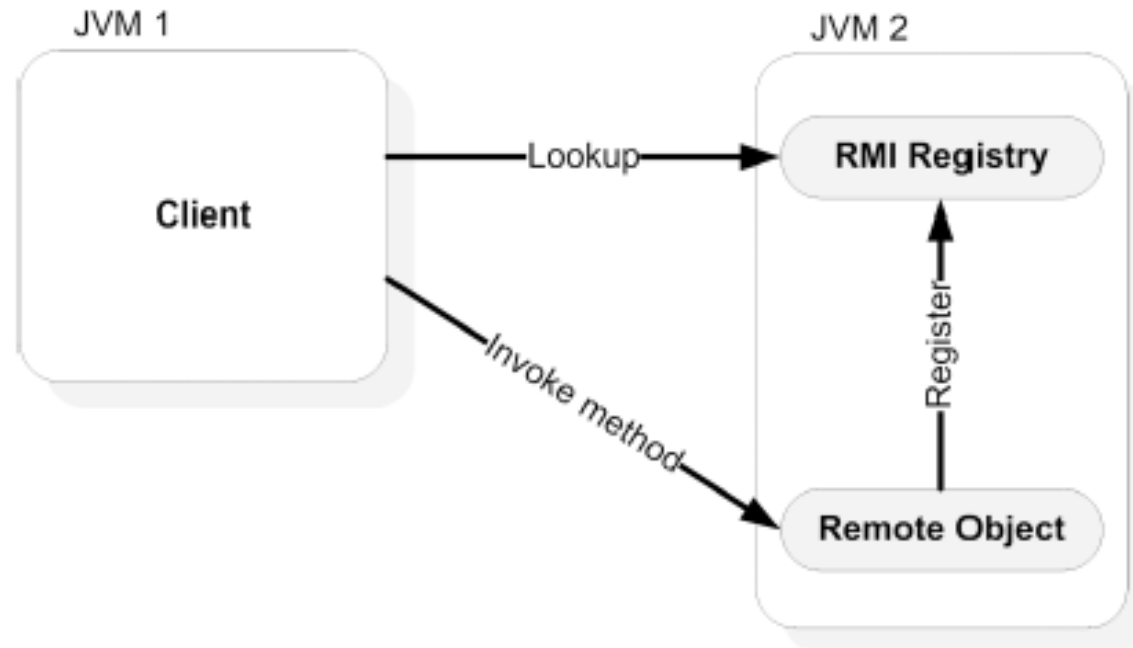
JNDI e Inyección de dependencias

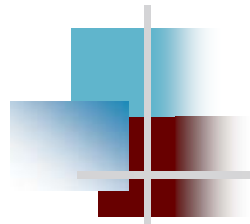
- Con JNDI es responsabilidad del cliente localizar y obtener la referencia a un objeto, componente o recurso
- Con EJB 3 y la inyección de dependencia, el contenedor localiza un objeto basándose en la declaración



JNDI (Java Naming and Directory Interface)

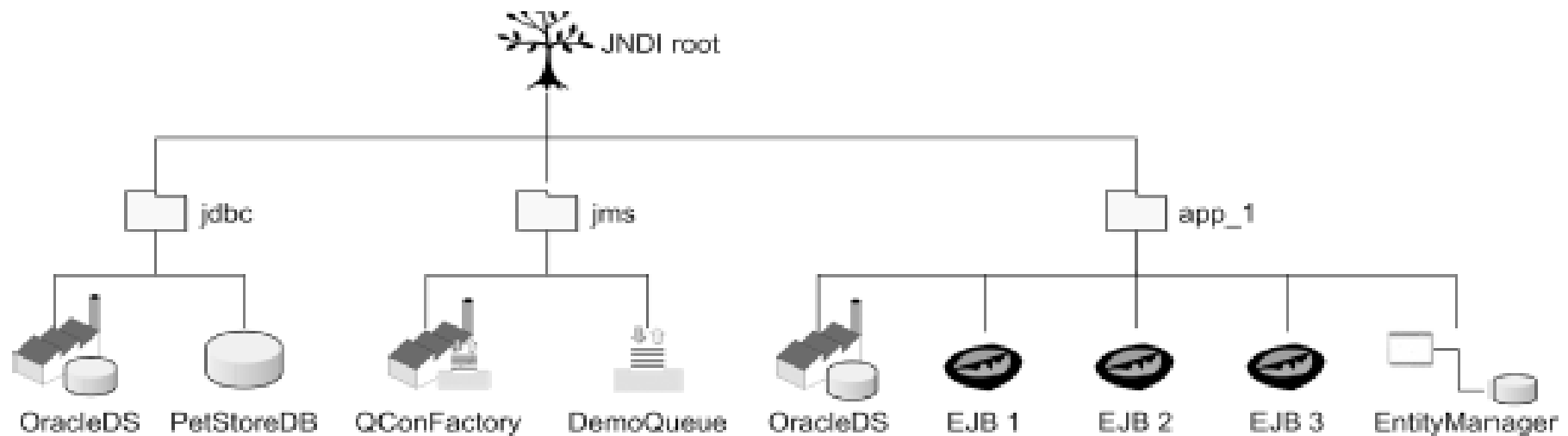
- JNDI es un servicio de nombres que permite a un componente localizar otros componentes o recursos (e.g., bases de datos via JDBC)





JNDI

- Habilita a las aplicaciones para acceder múltiples servicios de nombres y de directorios. NDS, DNS, NIS



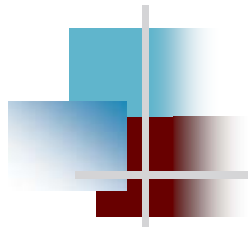
- El uso de JNDI en aplicaciones Java EE permite acceder a cualquier objeto Java, recurso (o sistema legado)



Localización (lookup)

- La localización de un recurso se puede realizar definiendo explícitamente la búsqueda con JNDI

```
Context context = new InitialContext();  
//JDBC  
DataSource dataSource = (DataSource) context.lookup  
    ("java:comp/env/jdbc/ActionBazaarDS");  
Connection connection = dataSource.getConnection();  
Statement statement = connection.createStatement();  
  
//Bean  
PlaceBid placeBid = (PlaceBid)context.lookup  
    ("java:comp/env/ejb/PlaceBid");
```

Inyección

```
import javax.ejb.EJB;

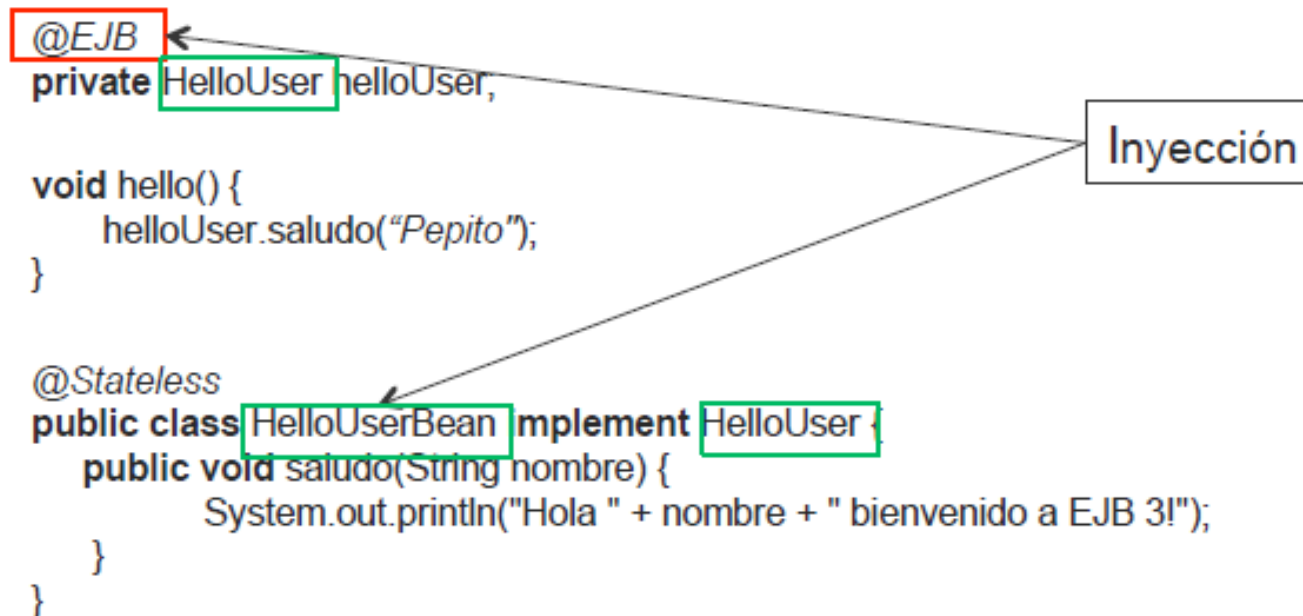
public class PlaceBidServlet extends HttpServlet {

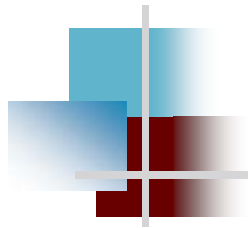
    @EJB
    private PlaceBid placeBid;

    public void service(HttpServletRequest request,
        HttpServletResponse response) throws
        ServletException,
        IOException {...}
```

Inyección

- Un cliente de aplicación JEE puede referirse a un EJB con la anotación `@EJB`.
- El contenedor EJB es el que inyecta en cada objeto los EJBs según las anotaciones que incluya.





Anotaciones y descriptores

- EJB 3 permite utilizar descriptores XML y anotaciones en el código
- Con los descriptores XML podemos cambiar la configuración sin tocar el código original

```
<enterprise-beans>
  <session>
    <ejb-name>HelloUserBean</ejb-name>
    <local>ejb3inaction.example.Hello</local>
    <ejb-class>ejb3inaction.example.HelloUserBean</ejb-
class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
  </session>
</enterprise-beans>
```



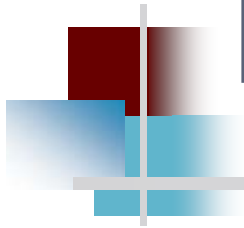
Resumen de Anotaciones

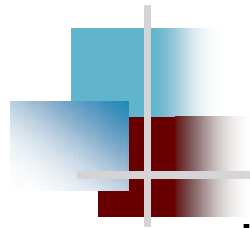
- Interface
 - `@Remote`: indica que se trata de una interfaz de negocio remota
 - `@Local`: acceso al bean de forma local únicamente

- Clase
 - `@stateless` similar a `@stateful` /* bean de sesión con estado */
 - Indica que el bean de sesión es sin estado
 - Dependiendo del contenedor
 - Se crea el stub del bean
 - Registra en JNDI el bean con el nombre lógico "java:comp/env/ejb/nombreBean" o con un nombre dado en la anotación
 - `@EJB`
 - Aplica para interfaces remotas únicamente
 - Genera el lookup del bean de sesión en JNDI con el nombre "java:comp/env/ejb/nombreBean"
 - En `serviciosCliente.java`
`@EJB(name = "co.com.uniandes.ejemplo.servicios.IServiciosProducto")`
`private IServiciosProducto serviciosProducto;`
 - `@Resource`
 - Inyección de recursos
 - `@Resource(name="jdbc/sqetestDB",type=javax.sql.DataSource.class)`
`public javax.sql.DataSource myTestDB;`

3.2.2

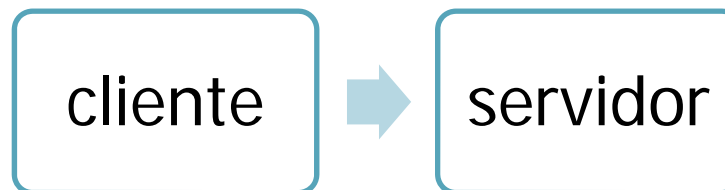
Message-Driven Beans





¿Porqué mensajes?

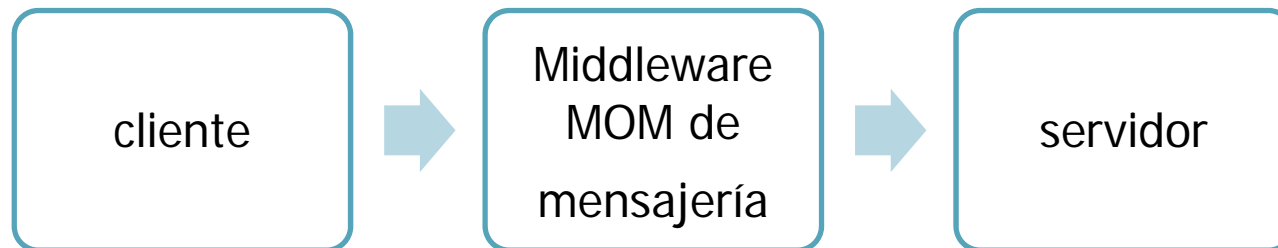
- Los Session Beans son síncronos
 - ¿Bloqueos? ¿Esperas?
- Los Session Beans deben conocer el servidor y dependen de él
 - Acoplamiento
 - ¿Qué pasa si se cae el servidor?
- Los Session Beans no están pensados para múltiples clientes y servidores
 - Un cliente y un servidor en cada llamada

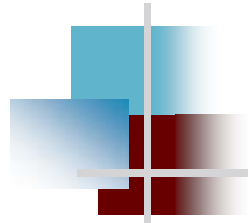




Utilizando mensajes...

- Podemos tener procesamiento sin bloquear la IU
 - Amazon.com “compra en un click”
- Desacoplamiento
 - Se envían los mensajes sin conocer el receptor
- Fiabilidad
 - Aunque esté caído el servidor se garantiza la entrega
- Multi-cast
 - Múltiples clientes y servidores en cada llamada



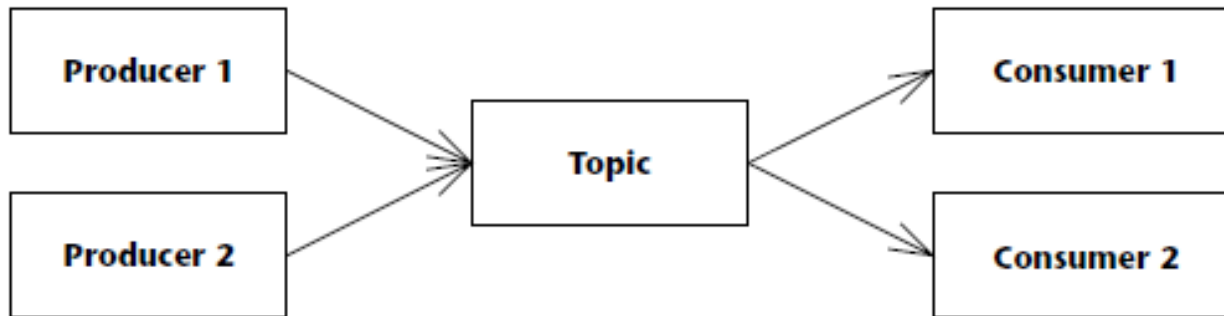


Message-oriented middleware

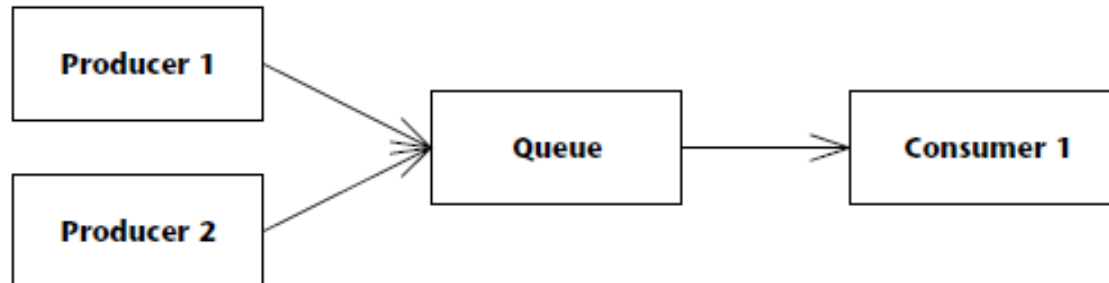
- (Software intermediario orientado a mensajes)
- Ejemplos de MOM:
 - Tibco Rendezvous
 - IBM Web-Sphere MQ
 - BEA Tuxedo/Q
 - Sun Java System Messaging Server
 - Microsoft MSMQ
- Java Message Service (JMS)
 - Estándar de mensajería para evitar usar APIs propietarias, similar a JDBC

Dos modelos

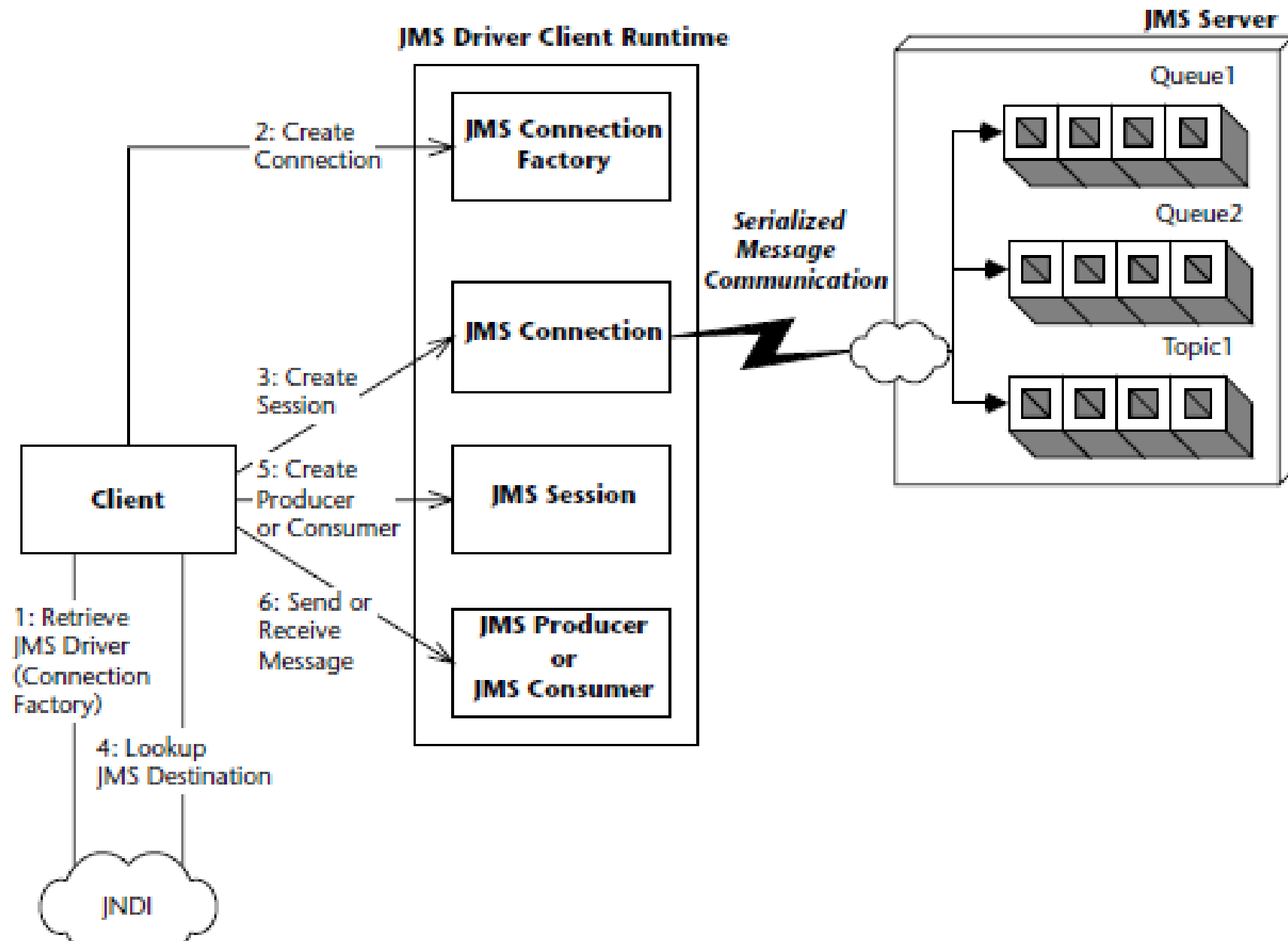
- Publish/Subscribe (eventos y “listeners”):

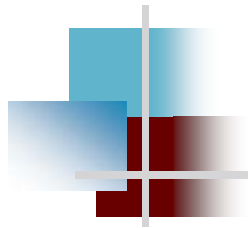


- Punto a punto:
 - La cola solo admite un consumidor para cada mensaje



Vista del cliente



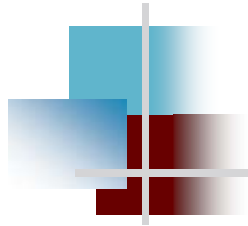


Un ejemplo simple...

```
// 1: localizar la factoría de conexiones con el contexto
TopicConnectionFactory factory =(TopicConnectionFactory)
    ctx.lookup ("jms/TopicConnectionFactory");

// 2: la factoría de conexiones  crea la conexión JMS
TopicConnection connection =
factory.createTopicConnection();

// 3: la conexión crea  la sesión
TopicSession Session = connection.createTopicSession
    (false,Session.AUTO_ACKNOWLEDGE);
```



Un ejemplo simple ...

```
// 4: localizar destino
```

```
Topic topic = (Topic)ctx.lookup("jms/Topic");
```

```
// 5: crear el emisor de mensajes
```

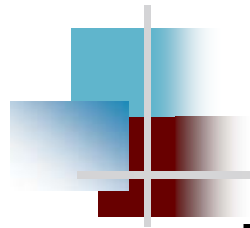
```
TopicPublisher publisher = session.createPublisher(topic);
```

```
// 6: crear y publicar un mensaje
```

```
TextMessage msg = session.createTextMessage();
```

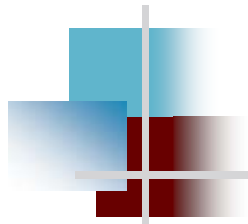
```
msg.setText("mensaje de prueba");
```

```
publisher.send(msg);
```



JMS + Session Bean

- Basta con añadir una clase Java que reciba los mensajes y los reenvíe a los Beans ya existentes...
- Es posible pero...
 - Requiere escribir el código del observador (listener)
 - Puede que haya que escribir código multi-hilos
 - La clase auxiliar no se beneficia de los servicios del contenedor EJB
 - La clase auxiliar no es reutilizable (depende de los detalles de JMS...)

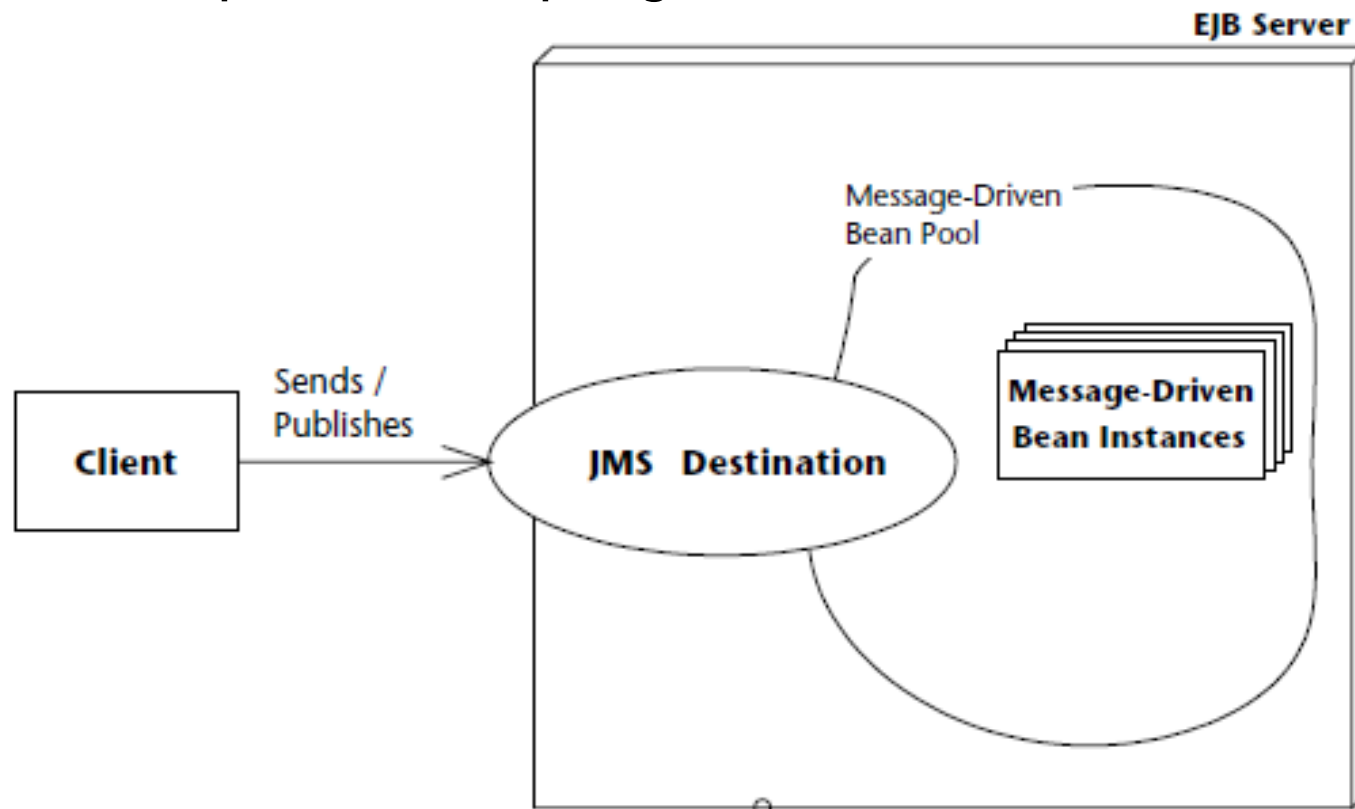


Message-Driven Bean

- Un Message-Driven Bean es un componente EJB especial que puede recibir mensajes JMS
 - Es invocado por el contenedor EJB a la llegada de un mensaje en el destino
 - Está desacoplado del productor de mensajes
 - Sin estado
 - No tiene una interfaz local o remota
 - No se le envían mensajes utilizando el estilo OO clásico
 - Implementan métodos listener genéricos para el mensaje entregado
 - Los métodos listener no devuelven valores de retorno ni lanzan excepciones para el cliente

Servidor

- El contenedor EJB consume los mensajes que llegan a un destino JMS según esta especificado en el descriptor de despliegue



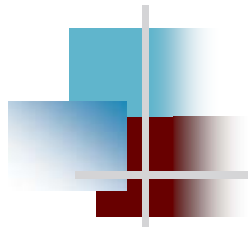


Implementación

- Dos interfaces:

```
public interface javax.jms.MessageListener {  
    public void onMessage(Message message);  
}
```

```
public interface javax.ejb.MessageDrivenBean  
    extends javax.ejb.EnterpriseBean {  
  
    ...  
  
    public void  
        setMessageDrivenContext(MessageDrivenContext ctx)  
  
        throws EJBException;  
}
```

Ejemplo: productor

@Stateful

```
public class GestorPedidosBean implements GestorPedidos{

    @Resource(name="jms/QueueConnectionFactory")

    private ConnectionFactory connectionFactory;

    @Resource(name="jms/ColaFacturacion")

    private Destination colaFact;

    ...

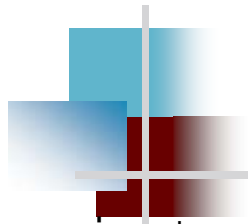
    @Remove

    public Long confirmarPedido() {

        ... facturar(pedido);

    }

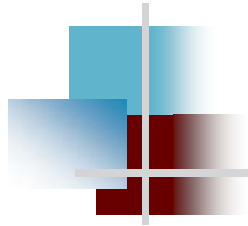
}
```



Ejemplo: productor

```
private facturar(Pedido pedido) {
    try {
        Connection connection = connectionFactory.createConnection();
        Session session = connection.createSession(false,
            Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer = session.createProducer(colaFact);
        ObjectMessage message = session.createObjectMessage();

        message.setObject(pedido);
        producer.send(message);
        producer.close();
        session.close();
        connection.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



Ejemplo: MDB consumidor

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName="destinationName",
        propertyValue="jms/ColaFacturacion")
})

public class FacturacionMDB implements MessageListener {
    ...
    public void onMessage(Message message) {
        try {
            ObjectMessage objectMessage = (ObjectMessage) message;
            Pedido pedido = (Pedido) objectMessage.getObject();
        }
    }
}
```

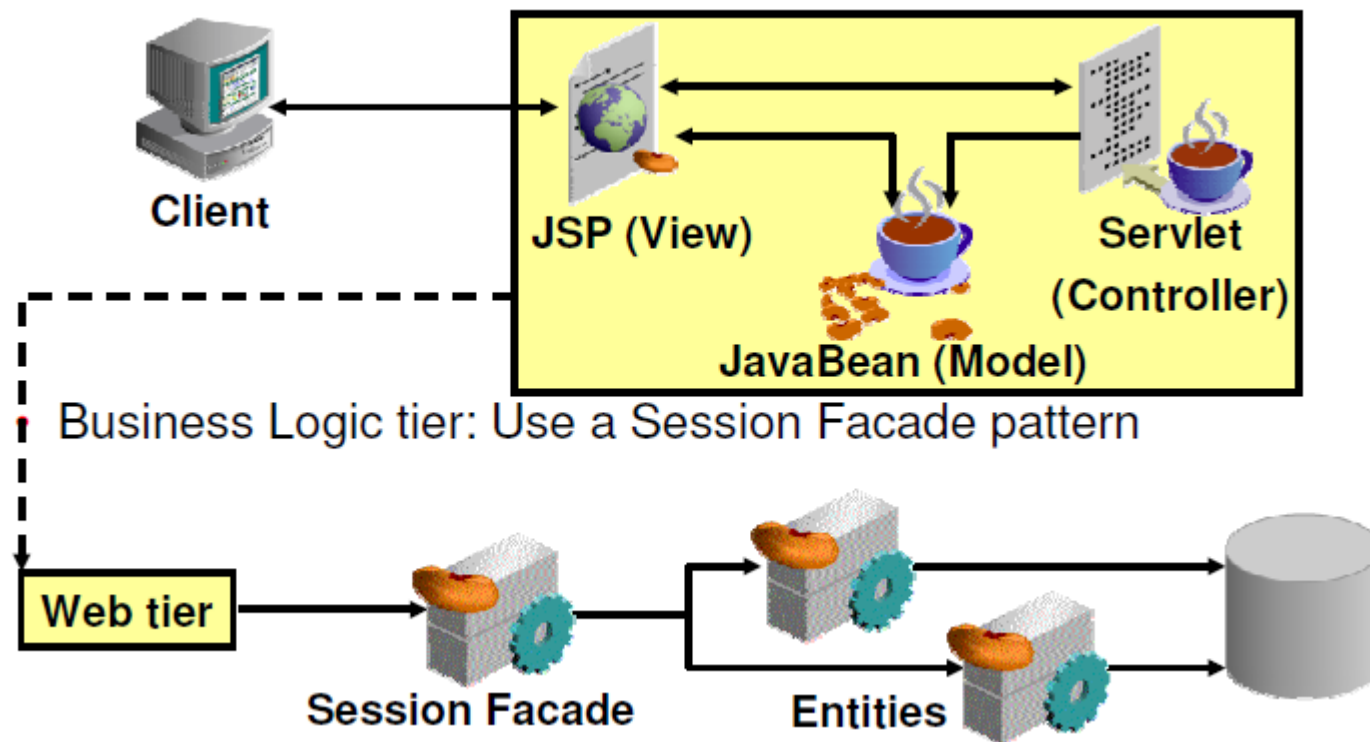
3.3

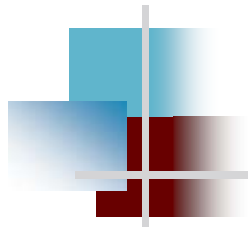
Persistencia



- JPA: Características y ejemplos
- JPA Entities y EntityManager

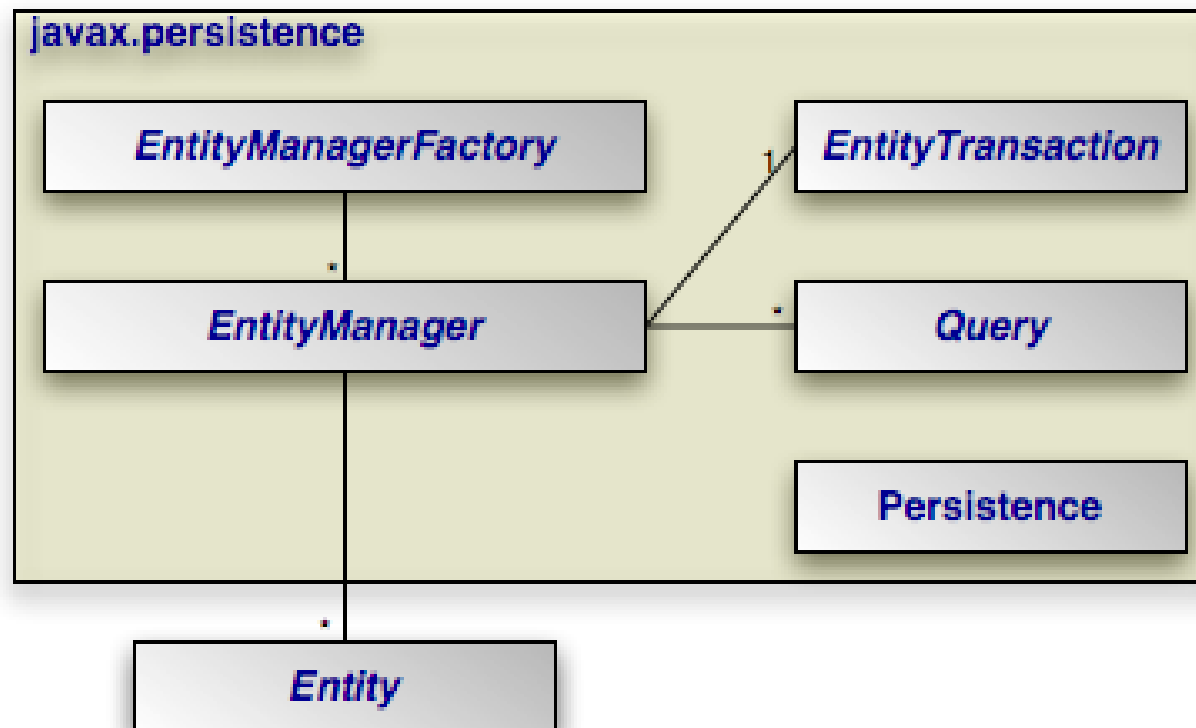
Propuesta de diseño

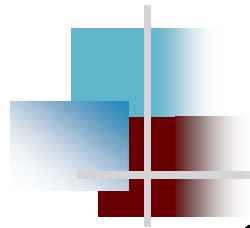




Mapping Objeto Relacional

- Muchos proyectos diferentes de ORM.
 - [Hibernate](#), IBatis, JDO, TopLink,...
- Necesaria unificación: JPA.



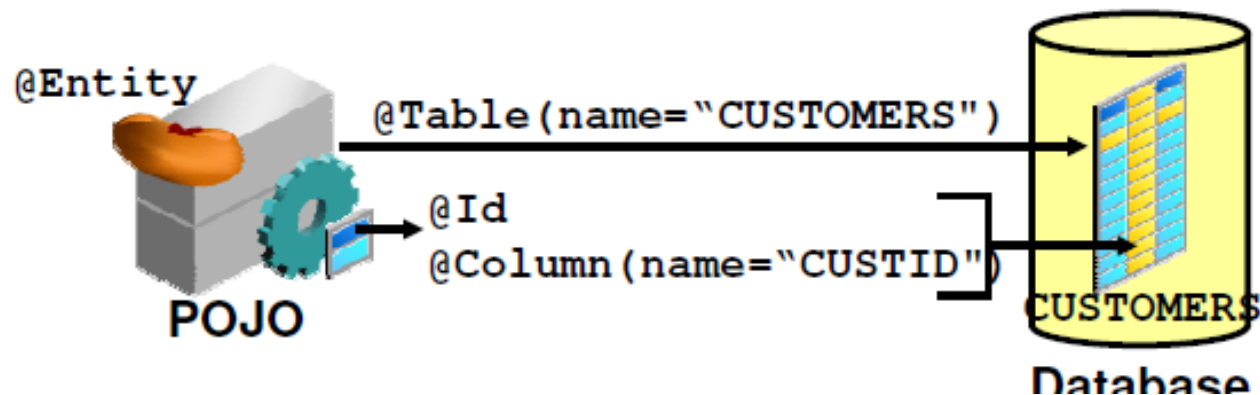


JPA (Java Persistence API)

- Simplificación de persistencia gestionada por contenedor
 - Enfoque POJO / JavaBeans
 - Permitir el uso de las entidades fuera del contenedor EJB
 - Contenedor web
 - Java SE
- Soporte para el modelo de dominio
 - Herencia y polimorfismo
 - Metadatos y anotaciones para el mapeo objeto-relacional
- Eliminación de la necesidad de objetos de transferencia de datos (DTO)

Entidad JPA (Entity)

- Clase ligera que gestiona los datos persistentes
- Marcada con la anotación `@Entity` (sin interfaces)
- Debe implementar la interfaz `java.io.Serializable`
 - Las instancias se pasan por valor a una aplicación remota
- Se asigna a una base de datos mediante anotaciones



Ejemplo de anotaciones JPA

```
@Entity
@Table(name="CUSTOMERS")
public class Customer {
    @Id
    @Column(name="CUSTID")
    private Long id;
    private String name;
    private Address address;
    private HashSet orders = new HashSet();

    public Long getId() {
        return id;
    }

    protected void setId (Long id) {
        this.id = id;
    }
    ...
}
```

CUSTOMERS
CUSTID (PK)
NAME



Mapeo de relaciones

```
// In the Order class

@ManyToOne
@JoinColumn(name="CUSTID")
public Customer getCustomer() {
    return customer;
}
...
// In the Customer class
@OneToMany(mappedBy="customer")
public Set<Order> getOrders() {
    return orders;
}

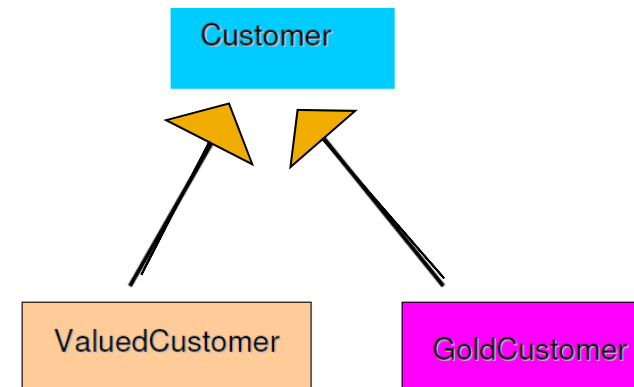
public void setOrders(Set<Order> orders) {
    this.orders = orders;
}

// other business methods, etc.
}
```

Herencia

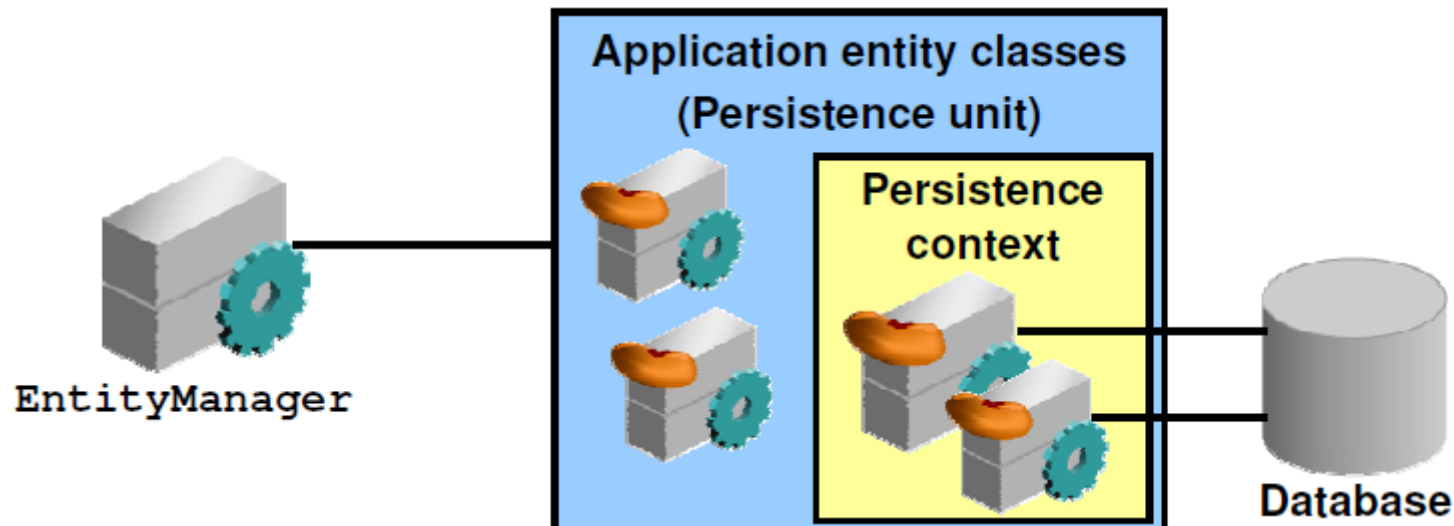
```
@Entity
@Table(name="CUSTOMERS")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE),
@DiscriminatorColumn(name="CUST_TYPE",
                    discriminatorType=STRING)
public class Customer {
    ...
}

@Entity
@DiscriminatorValue(value="V")
public class ValuedCustomer extends Customer{...}
```



EntityManager

- Gestiona el ciclo de vida de instancias entidad
- Está asociado con un contexto de persistencia
- Una instancia de EntityManager gestiona un conjunto de entidades definido por una unidad de persistencia





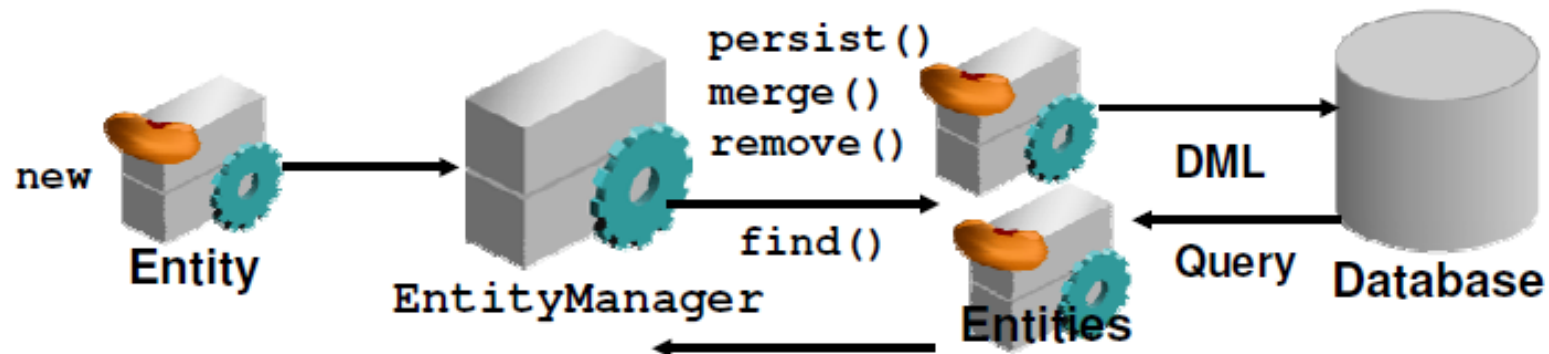
Contextos de Persistencia

- Transacciones JTA: Empleada en Servidores de Aplicaciones Java EE
- Transacciones RESOURCE_LOCAL: Empleada en Aplicaciones C/S.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
version="1.0">
  <persistence-unit name="defaultPU" transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>jdbc/NombreDataSource</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="validate"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.default_schema" value="NOMBRE"/>
```

Gestión de la persistencia

- Cada instancia de tipo entidad se crea con `new` o se recupera con una búsqueda en la BD (`find()`)
- La entidad se inserta, actualiza o borra (`persist()`, `merge()`, `remove()`) a través de la instancia de `EntityManager`





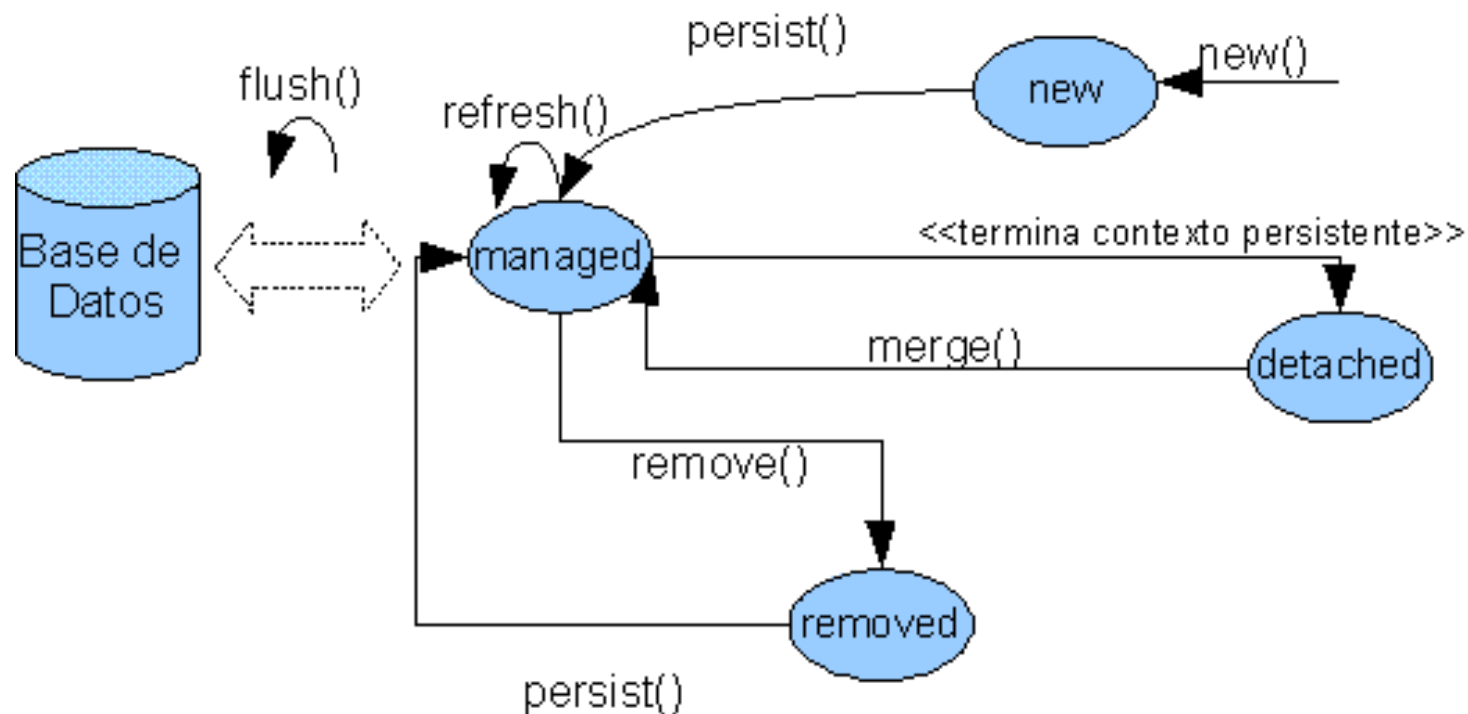
Inserción

- Crear una nueva instancia de la entidad.
- Llamar al método `persist ()` de `EntityManager`

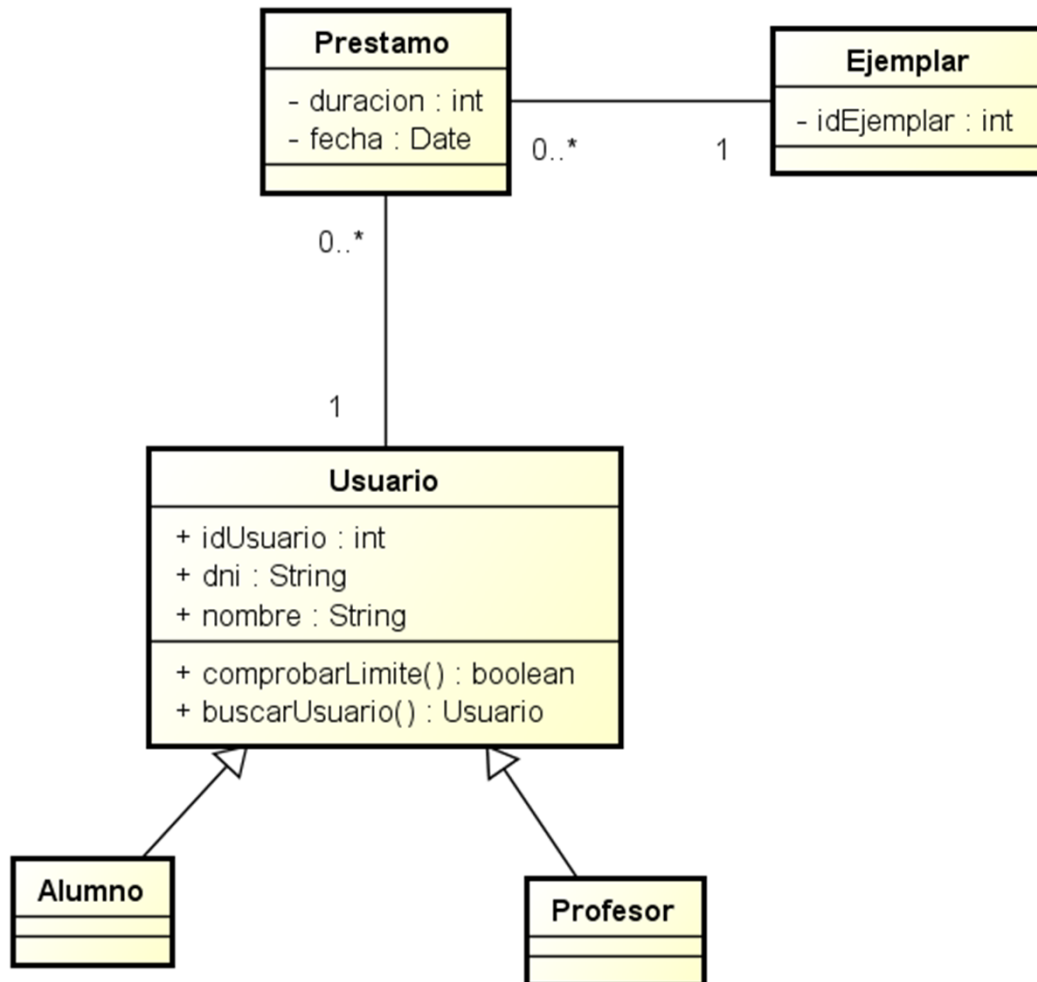
```
@PersistenceContext
private EntityManager em; //inyección del EntityManager
...
public void crearUsuario(String nom, String ape) {
    Usuario usuario = new Usuario();
    usuario.setFirstName(nom);
    usuario.setLastName(ape);
    em.persist(usuario);

    // Cuando el método devuelve el control usuario está
    // en estado "persistido" con los valores autogenerados
}
```

Ciclo de Vida de una Entity



Mapping OR: anotaciones



TABLAS:

USUARIOS

ID_USUARIO PK

DNI

NOM_APE

TIPO_USUARIO

EJEMPLAR

ID_EJEMPLAR PK

PRESTAMO

ID_PRESTAMO PK

ID_USUARIO FK

ID_EJEMPLAR FK

DURACION

FECHA

Anotaciones Básicas:

Entidades, atributos y claves

`@Entity`

`@Table(name="USUARIOS")`

`public class Usuario{`

`@Id`

`@GeneratedValue(strategy = GenerationType.IDENTITY)`

`@Column(name="ID_USUARIO")`

`protected Long idUsuario;`

`@Column(name="DNI")`

`protected String dni;`

`@Column(name="NOM_APELLIDO")`

`protected String nombre;`

`@Lob`

`@Column(name="IMAGEN")`

`@Basic(fetch=FetchType.LAZY)`

`protected byte[] imagen;`

`public Long getIdUsuario() {`

`// versión simplificada`

`@Entity`

`public class Usuario{`

`@Id`

`protected Long idUsuario;`

`protected String dni;`

`protected String nombre;`

`...`

`}`



Anotaciones Básicas: Relaciones

- Tipos:
 - @OneToOne
 - @OneToMany
 - @ManyToOne
 - @ManyToMany
- Pueden ser bidireccionales (mappedBy)
- Pueden ser obligatorias (optional="false")
- Los parámetros cascade (borrado en cascada) y fetch (cómo se recuperan de la BD) completan la información



Relaciones @OneToOne

@Entity

```
public class Usuario{  
    @Id  
    protected Long idUsuario;  
    protected String dni;  
    @OneToOne  
    protected InforFacturacion inforFac;  
    ...  
}
```

@Entity

```
public class InforFacturacion {  
    @Id  
    protected Long idFact;  
    protected String tarjetaCredito;  
  
    @OneToOne(mappedBy="inforFac", optional="false")  
    protected Usuario usuario;  
}
```

Relaciones

@OneToMany/@ManyToOne

@Entity

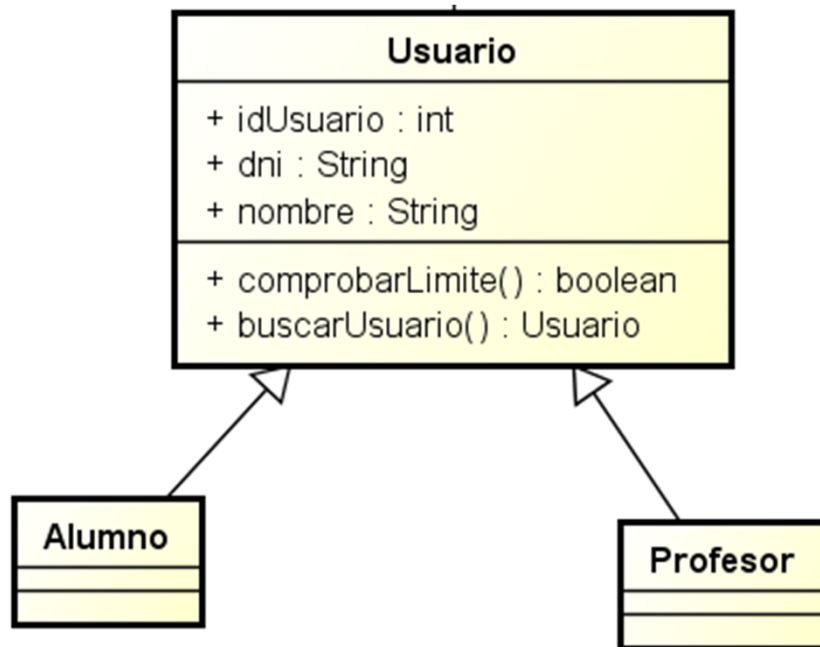
```
public class Usuario{  
    @Id  
    protected Long idUsuario;  
    protected String dni;  
    @OneToMany (mappedBy="usuario")  
    protected set<Prestamo> prestamos;  
    ...  
}
```

@Entity

```
public class Prestamo{  
    @Id  
    protected Long idPrestamo;  
    protected int duracion;  
    protected Date fecha;  
    @ManyToOne(optional="false")  
    protected Usuario usuario;  
}
```



Relaciones de herencia



TABLAS:

USUARIOS

ID_USUARIO PK

DNI

NOMBRE

TIPO_USUARIO



Relaciones de herencia

```
@Entity
@Table(name="USUARIOS")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TIPO_USUARIO",
discriminatorType=DiscriminatorType.STRING, length=1)
public abstract class Usuario{
    @Id
    protected Long idUsuario;
    protected String dni;
    protected String nombre;

    ...

@Entity
@DiscriminatorValue(value="A")
public class Alumno extends Usuario
```



Query API

- Las consultas se pueden expresar en JPQL o SQL nativo



EntityManager

→ `createQuery(String jpql)`
→ `createNamedQuery`
 `(String jpql)`
→ Native Query methods

Query instance methods:



`setParameter(String, Object)`
`setParameter(int, Object)`



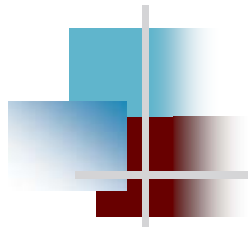
`Object getSingleResult()`

`List getResultList()`



`Query setMaxResults(int)`

`Query setFirstResult(int)`



JDBC y JPA

JDBC y SQL	JPA y JPQL
Obtener una conexión JDBC a la BD	Obtener una instancia de EntityManager
Crear una sentencia SQL	Crear una instancia de consulta
Ejecutar la sentencia	Ejecutar la consulta
Recuperar los resultados (formato registro)	Recuperar los resultados (objetos tipo "Entity")



Named/dinamic Query

```
@PersistenceContext em;

public List findAllCategories() {
    Query query = em.createQuery("SELECT c FROM Category
c"); ...
    return query.getResultList();
}
```

```
@Entity
```

```
@NamedQuery(name = "findAllCategories",
    query = "SELECT c FROM Category c WHERE c.categoryName
    LIKE :categoryName ")
```

```
public class Category implements Serializable {
    //en un método getCategories() de la interfaz de
    //persistencia:
```

```
Query query = em.createNamedQuery("findAllCategories");..
```



Automatización (NetBeans)

```
@Entity
@NamedQueries({
    @NamedQuery(name = "Manufacturer.findAll", query =
"SELECT m FROM Manufacturer m"),
    @NamedQuery(name =
"Manufacturer.findByManufacturerId", query = "SELECT m
FROM Manufacturer m WHERE m.manufacturerId =
:manufacturerId"),
    ...})
```

```
public class Manufacturer implements Serializable {
    @Id
    @NotNull
    private Integer manufacturerId;
    private String name;
    ...
}
```

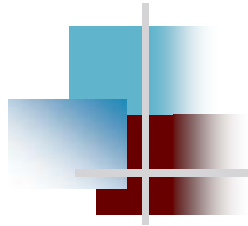


Automatización (NetBeans)

- Adaptación del patrón DAO
- No solo genera la clase `@Entity` sino también un EJB de sesión y su interfaz CRUD de persistencia

`@Local`

```
public interface ManufacturerFacadeLocal {  
    void create(Manufacturer manufacturer);  
    void edit(Manufacturer manufacturer);  
    void remove(Manufacturer manufacturer);  
    Manufacturer find(Object id);  
    List<Manufacturer> findAll();  
}
```

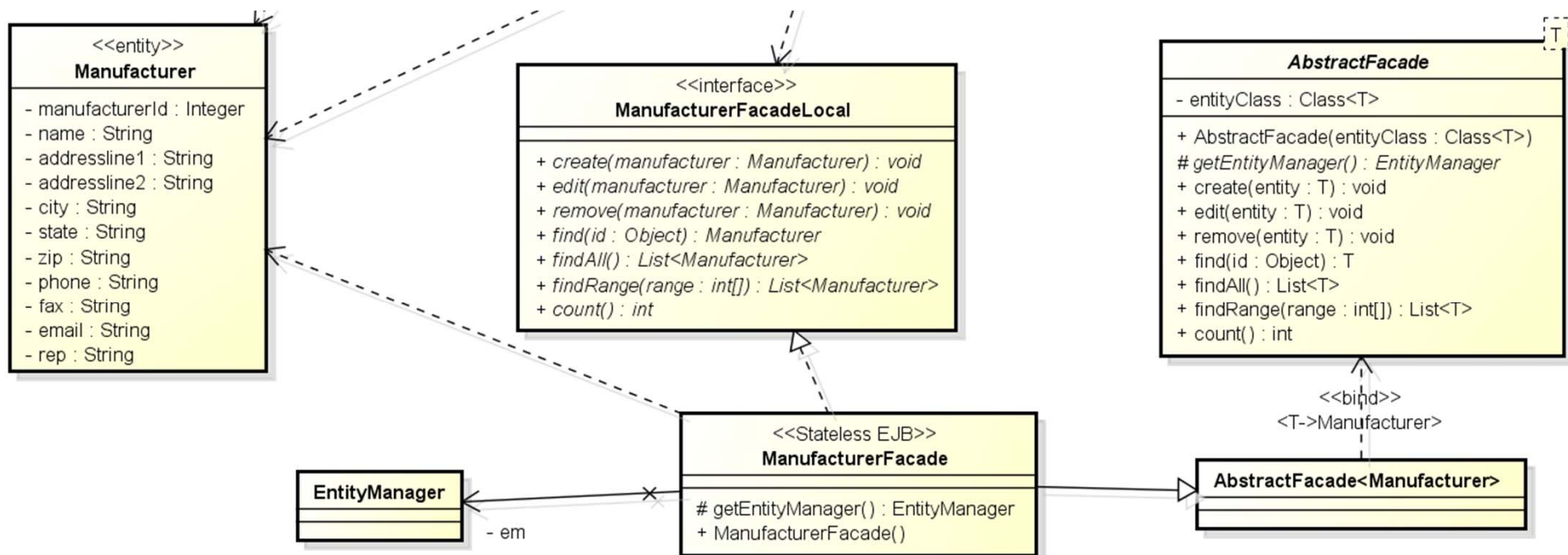


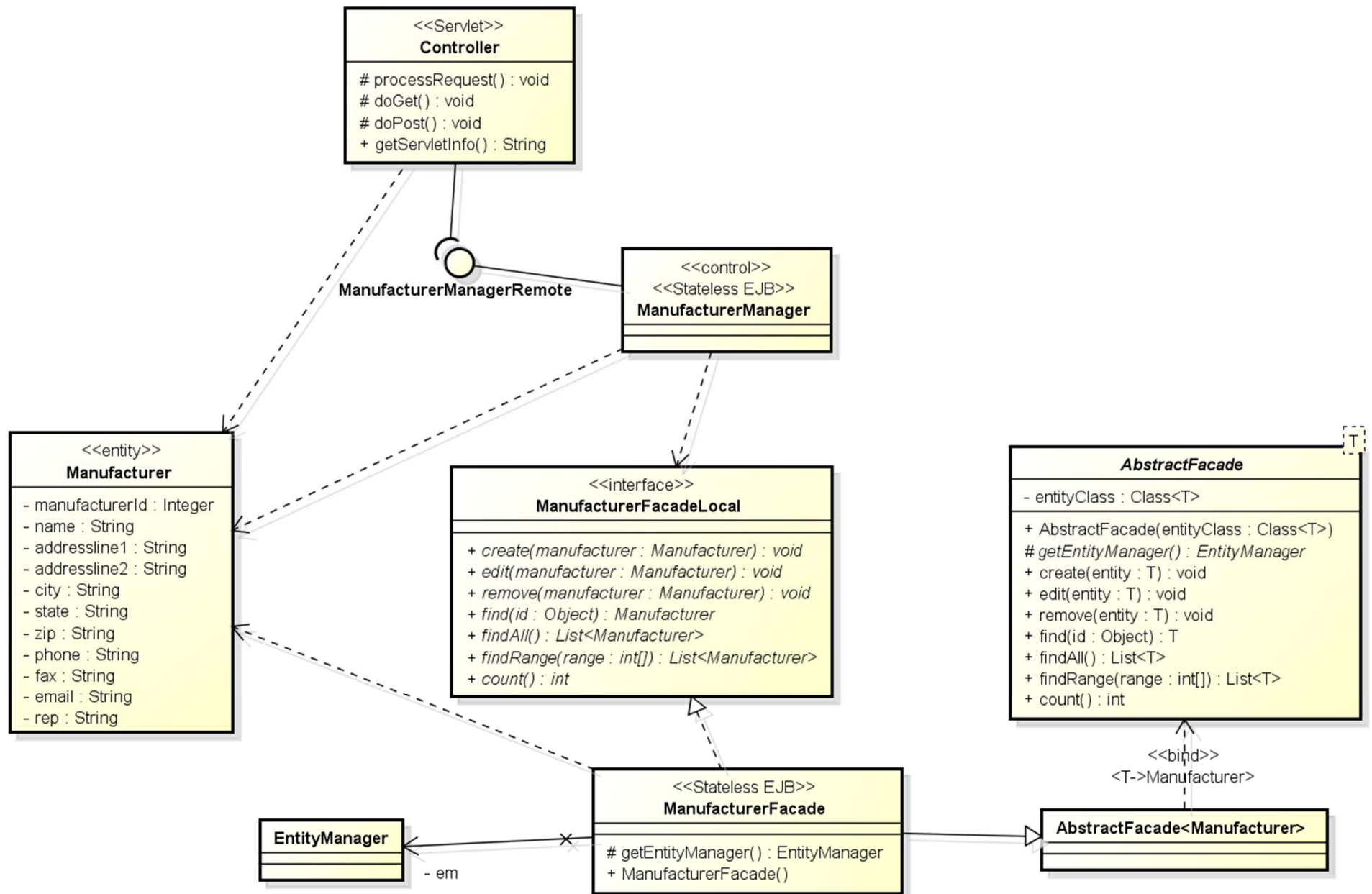
Implementación en el EJB

```
public void create(Manufacturer entity) {  
    getEntityManager().persist(entity);  
}  
  
public void edit(Manufacturer entity) {  
    getEntityManager().merge(entity);  
}  
  
public Manufacturer find(Object id) {  
    return getEntityManager().find(Manufacturer.Class, id);  
}
```

Diseño

- Un avance...capa de persistencia



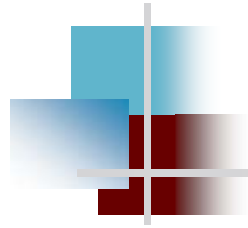


3.4.

Despliegue de Aplicaciones

JEE

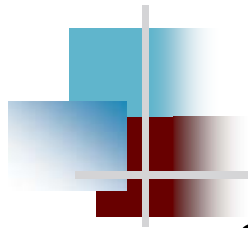
- 
-
- Tecnología de la capa de presentación
 - Despliegue



Dos tipos de aplicaciones Web

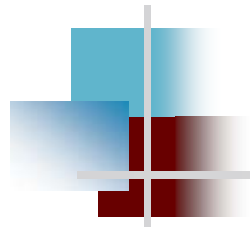
- Aplicaciones Web simples con Java/JSP
- Aplicaciones Web con EJBs

- En ambos casos habrá una colección de recursos comunes tales como
 - JSP
 - Servlets
 - Ficheros Html
 - Imágenes...



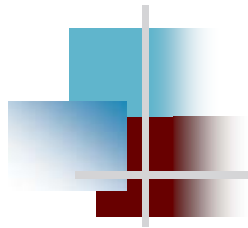
Componentes web

- Servlets
 - Clases escritas en Java que procesan peticiones y construyen respuestas
- JSP
 - Documentos basados en texto que contienen dos tipos de texto: una plantilla de datos estática que puede expresarse en un formato como HTML o XML, y elementos JSP que determinan cómo la página construye el contenido dinámico



Componentes web: Servlets

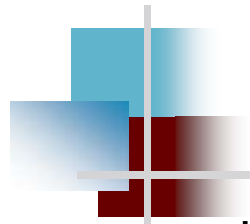
- Clase Java con un ciclo de vida concreto.
 - Init: Ejecutado al cargarse la primera vez.
 - doGet/doPost: Ejecutados al recibirse peticiones de cada tipo concreto.
 - Destroy: Ejecutado al finalizar.
- Importante: Se ejecuta en un hilo.
- Atributos:
 - Request/Response/Session



Componentes web: JSP

- Código Java dentro de HTML.
- Se compila y genera un servlet. Maneja los mismos objetos request, session...
- Custom Tags. Ampliar la sintaxis de HTML.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Ejemplo</title>
  </head>
  <body>
    Hola <%=request.getParameter("nombre") %>.
  </body>
```



Aplicación Web: organización

- Una aplicación web está organizada en una estructura jerárquica de directorios

- Un directorio privado WEB-INF
- Un directorio público

- Ejemplo:

miaplicación\

Index.html

login.jsp

images\ logo.gif

doc\ tutorial.pdf

WEB-INF\

web.xml (Deployment Descriptor)

classes\ ServletCompras.class

lib\ cualquierOtraApi.jar

- Una aplicación web puede ser empaquetada en un fichero WAR.

Web.xml

Descriptor de Despliegue

- WEB-INF/web.xml

- Documento XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>  
<DOCTYPE web-app PUBLIC  
    "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"  
    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

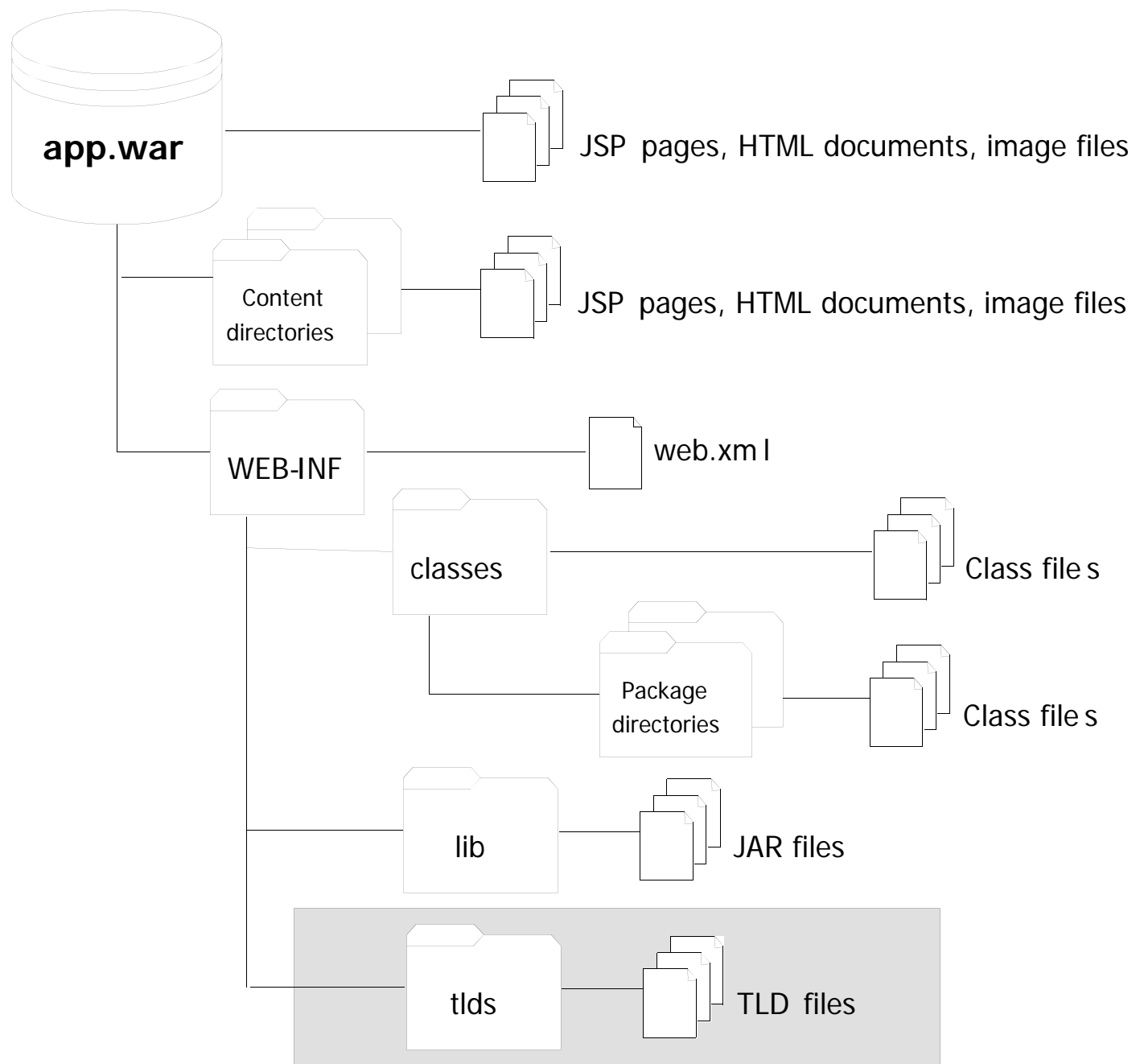
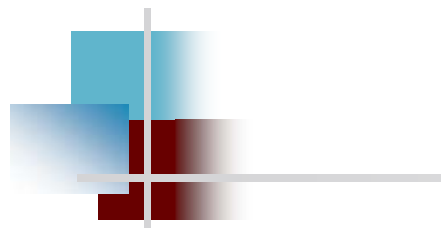
- En él se dan de alta

- Servlets
- Parámetros del contexto
- TLDs (etiquetas)
- Filtros
- Etc.



Archivos WAR

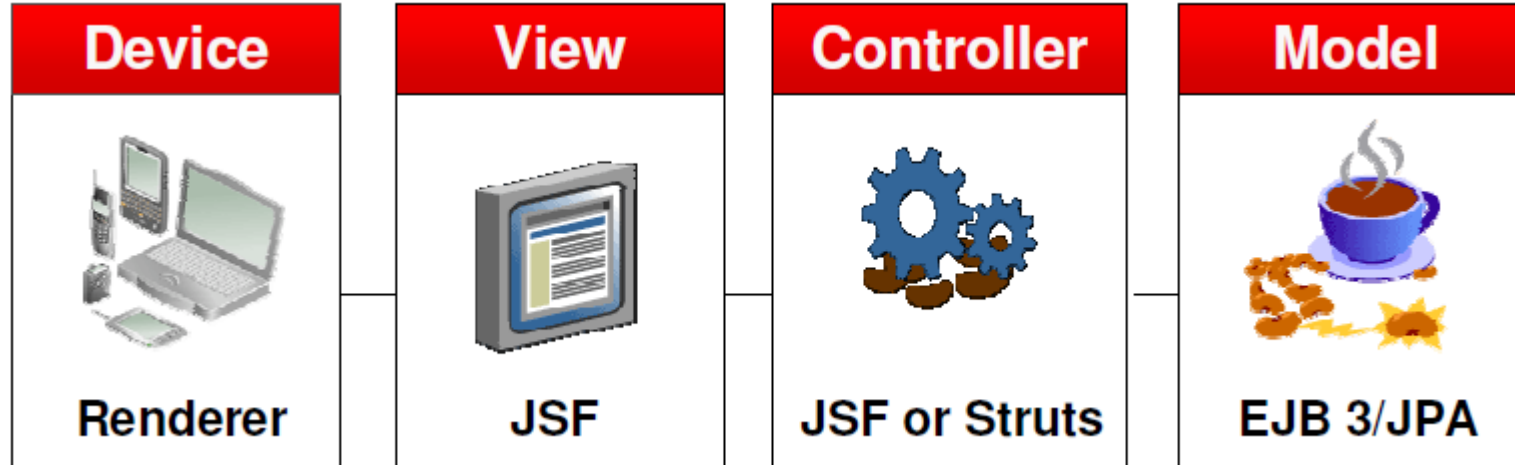
- **Web Application Archive** equivalente al JAR
- Permiten empaquetar en una sola unidad aplicaciones web java completas.
 - Servlets y JSPs
 - Contenido estático
 - Html
 - Imágenes
 - etc.)
 - Otros recursos web



Java EE

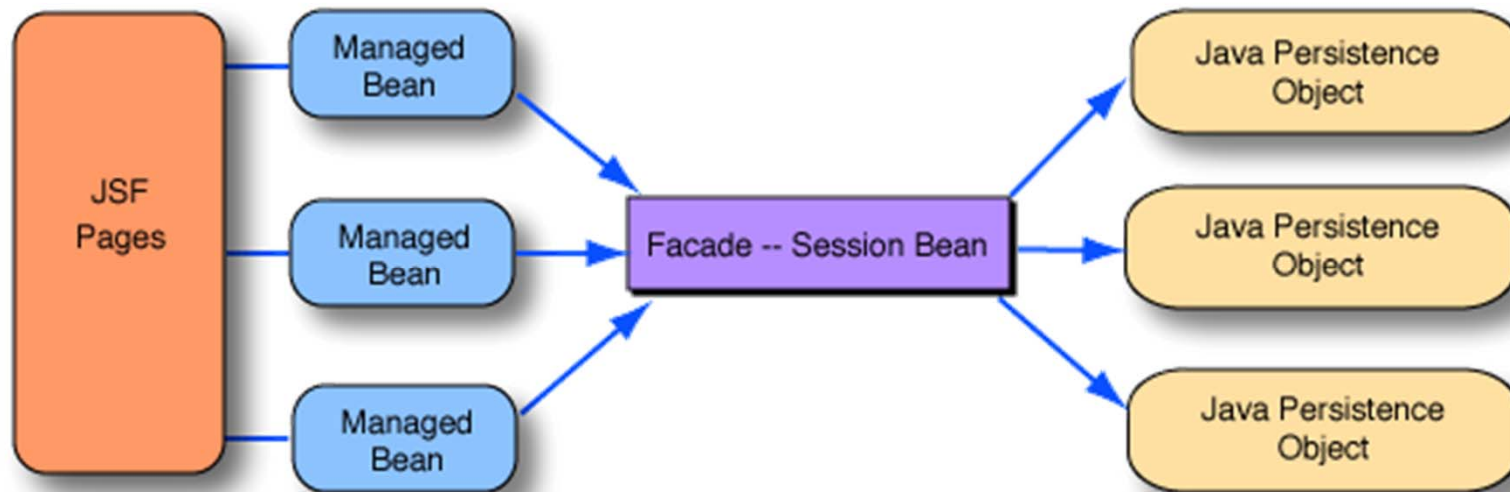
Tecnologías recomendadas

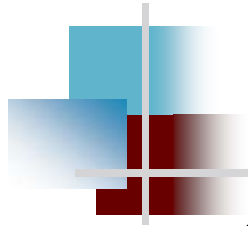
- JSF (Struts)



Tecnologías Java EE

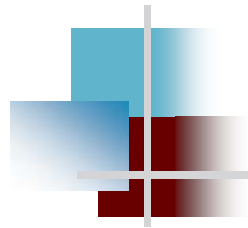
- Aplicación Completa: JSF (Vista) + EJB3 (Controlador) + JPA (Modelo)





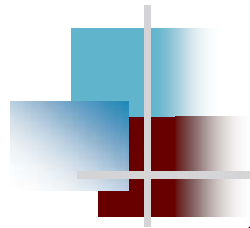
Tecnologías Vista: JSF

- Componentes de JSF:
 - API + Implementación de Referencia.
 - Representan componentes UI y manejan su estado, eventos, validaciones, navegación, etc...
 - Librería de Etiquetas.
 - Etiquetas personalizadas de JSP para dibujar los componentes UI dentro de las páginas JSP.



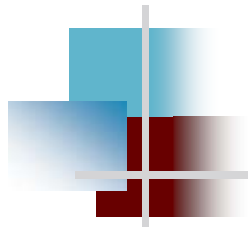
Tecnologías Java EE

- Ciclo de Vida JSF
 - Las peticiones Faces no se limitan a petición-respuesta, disponen de un ciclo de vida.
 - El ciclo de vida depende del tipo de petición.
 - Respuesta No-Faces: Respuesta generada al margen de la fase de renderizar respuesta de faces.
 - Respuesta Faces: Respuesta generada en la fase de renderizar respuesta de faces.
 - Petición No-Faces: Petición enviada a un componente no faces.
 - Petición Faces: Petición enviada desde una respuesta faces previamente generada.
 - El escenario normal Petición faces/Respuesta faces.



Componentes JSF

- Conjunto de clases UIComponent.
 - Representan los componentes.
- Modelo de renderizado.
 - Forma de visualizar el componente.
- Modelo de eventos.
 - Forma de manejar los eventos lanzados.
- Modelo de conversión.
 - Conectar conversores de datos al componente.
- Modelo de validación.
 - Forma de registrar validadores para el componente.
- RichFaces, ICEFaces
 - Librerías de etiquetas.



Facelets

- Complemento ideal para JSF.
- Definir una plantilla para un portal y emplearla en todas sus páginas.

```
<ui:include src="cabecera.xhtml"/>
```

```
<ui:include  
src="menu.xhtml"  
/>
```

```
<ui:insert name="body"/>
```

```
/pagina.xhtml
```

```
<ui:composition template="/plantilla.xhtml">
```

```
    <ui:define name="body">
```

```
        ...
```

```
    </ui:define>
```

```
</ui:composition>
```

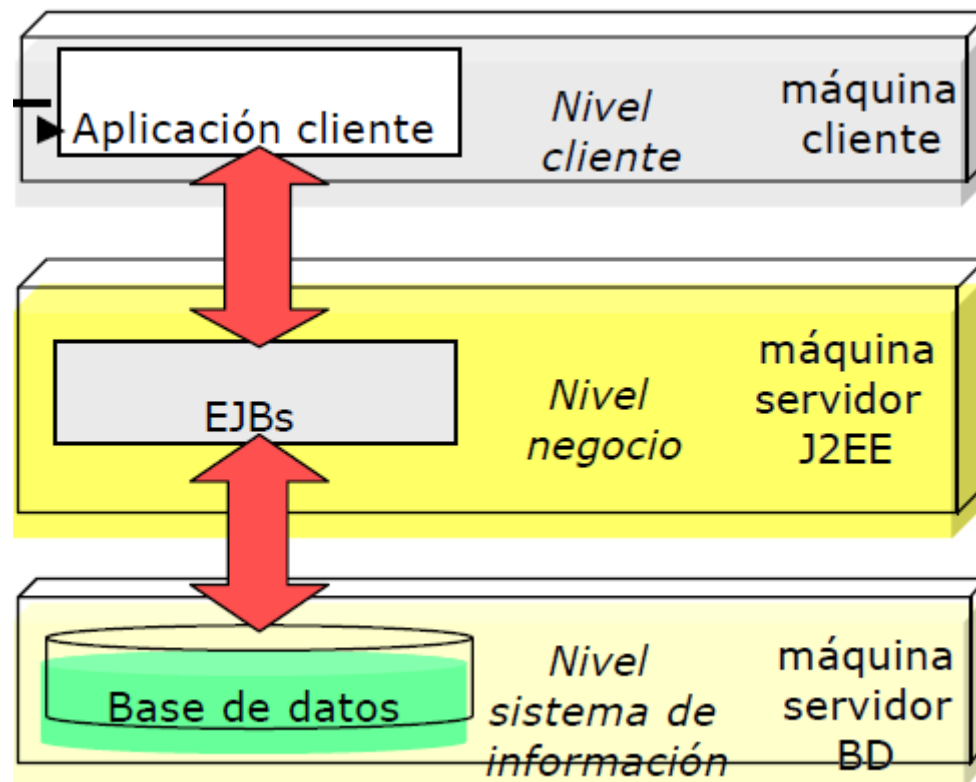
```
<ui:include src="pie.xhtml"/>
```



Aplicación JEE

Aplicación no web que utiliza JEE (Dominio y Persistencia)

- Problema: Es necesario utilizar JDNI para acceder a los EJB

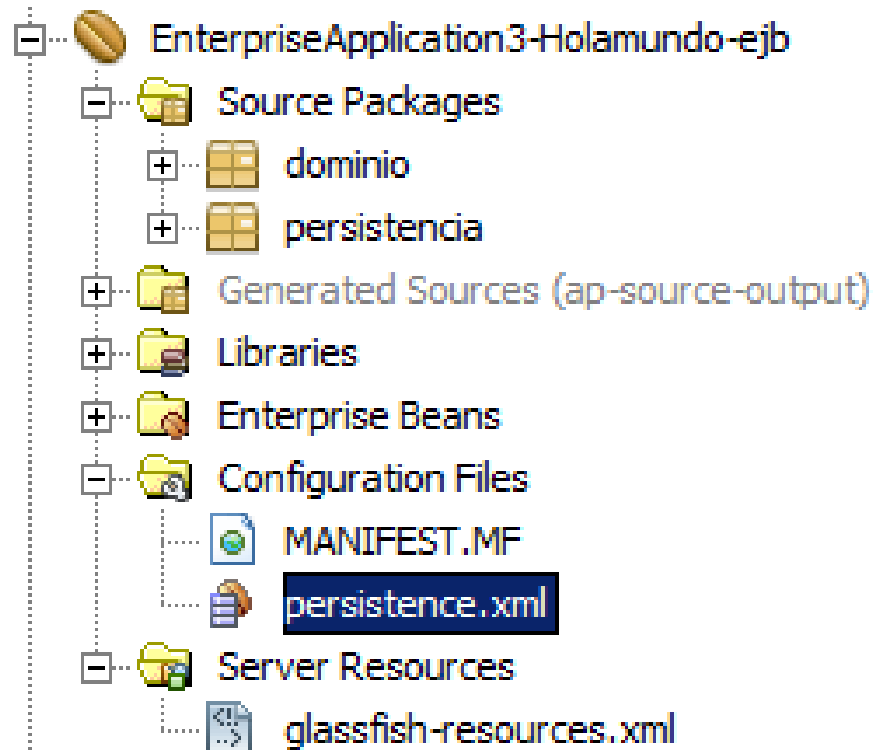




Organización de los EJBs

- Los EJBs se agrupan en paquetes un EJB-JAR para ser distribuidos
 - De la misma forma en que los WARS modularizan el desarrollo de aplicaciones con JSP's/Servlets, un EJB-JAR lo hace para desarrollos con EJB's
- En un solo paquete se pueden incluir EJBs de diferentes tipos
- Facilitan el despliegue de los EJBs en cualquier servidor de aplicaciones compatible JEE

Estructura EJB-JAR



/dominio

/persistencia

Los archivos .class
que conforman los
EJB

/META-INF

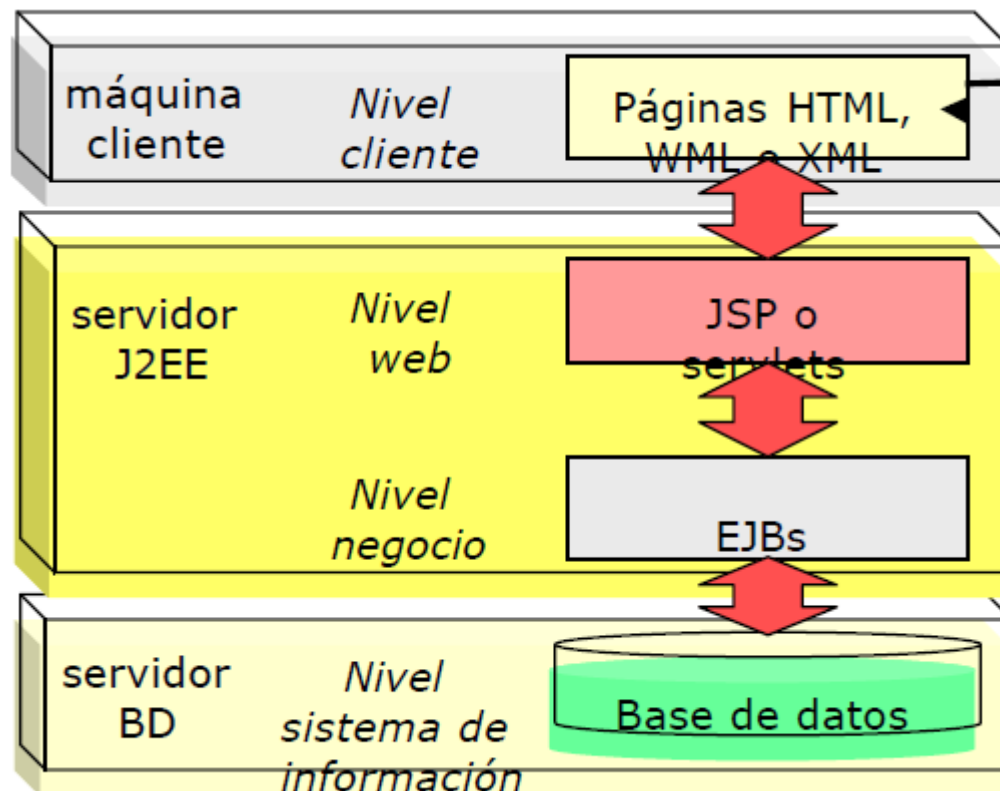
Descriptor de
despliegue,
incluido
persistence.xml

Recomendación: separar la declaración de la interfaz remota del
componente de la implementación (ambos son JAR <- proyecto NetBean)

- Incluso las "entities" se pueden separar en otro paquete

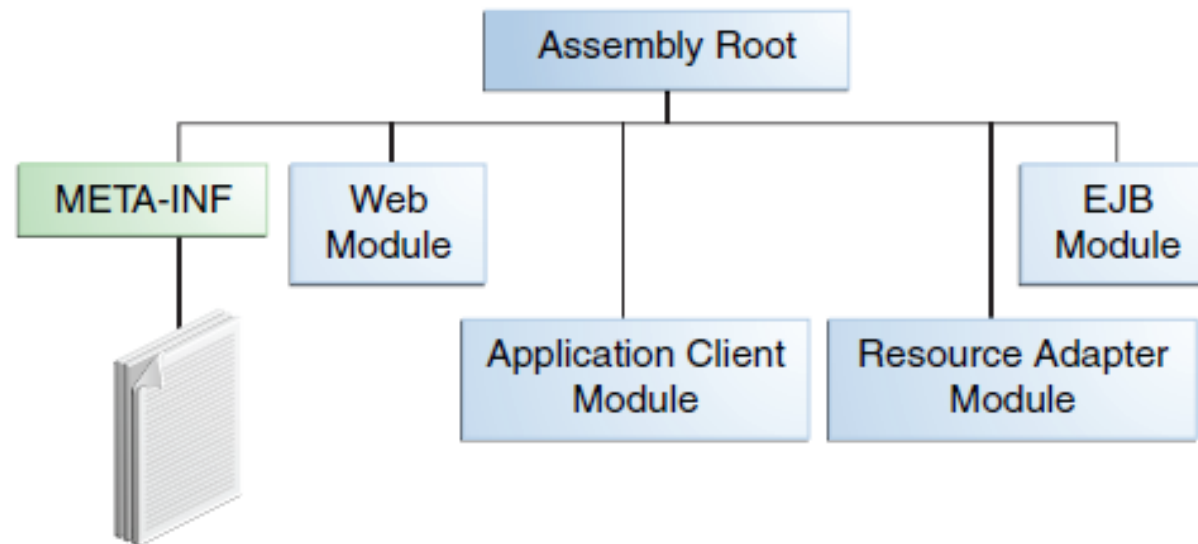
Aplicación Web/JEE

- En los Servlets se puede utilizar inyección de dependencias (pero no directamente en las JSP)

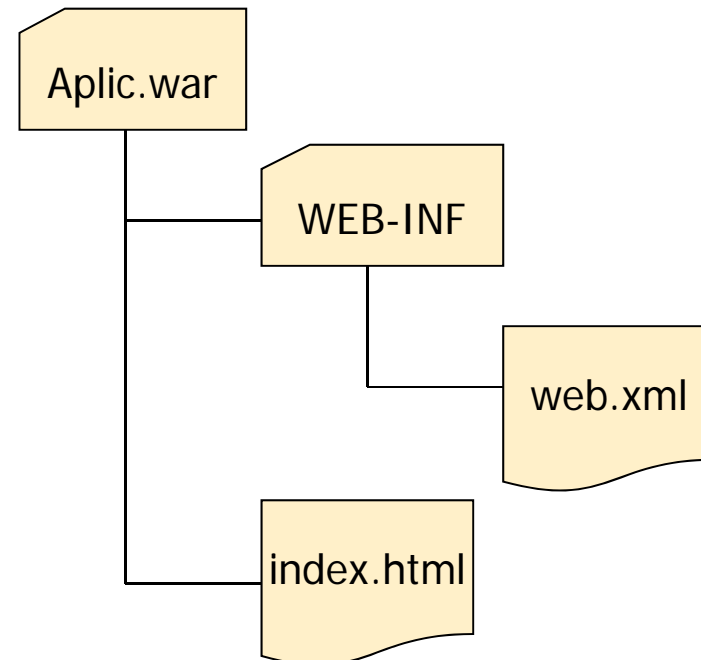
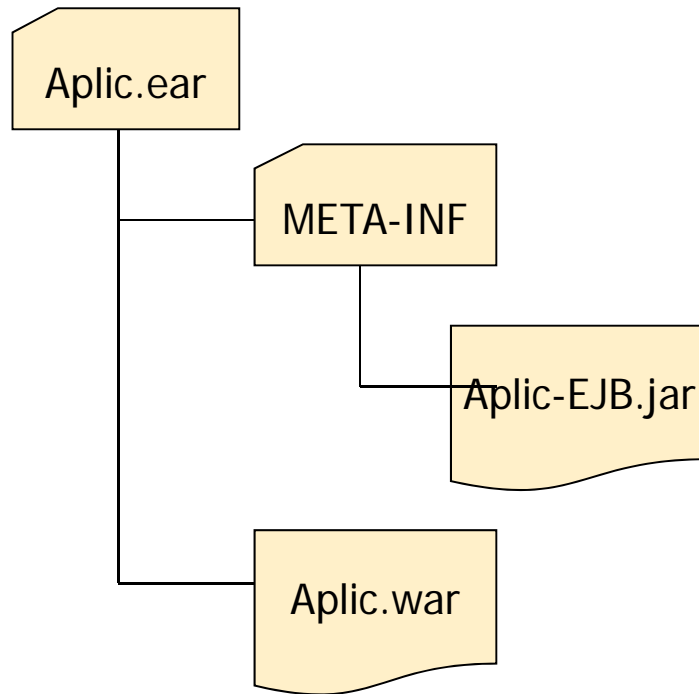


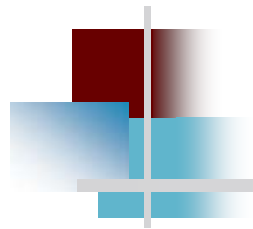
Aplicación Web/JEE

Aplicación empresarial JEE .EAR =
Aplicación web java empaquetadas en WAR +
EJB distribuidos empaquetados en JAR +
información opcional de despliegue

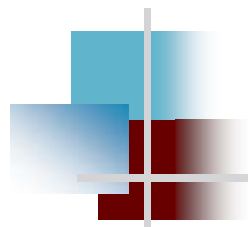


Estructura básica de EAR Y WAR

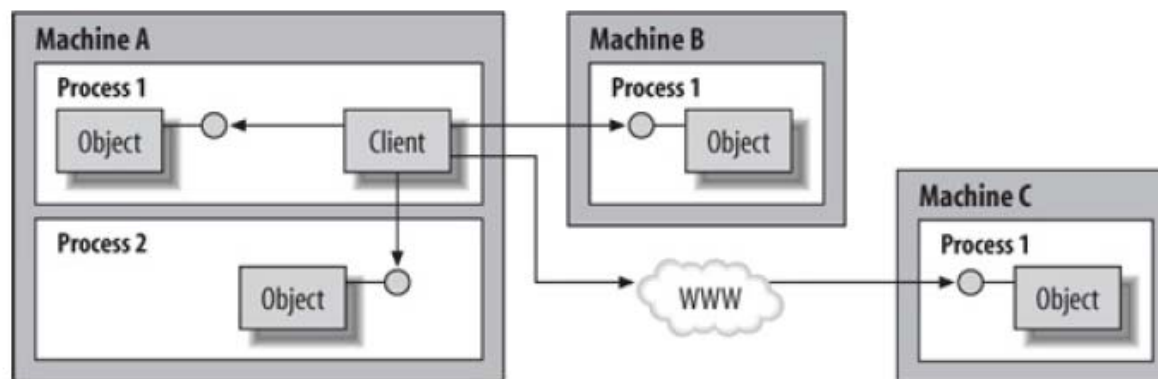




3.5. DCOM/COM+



DCOM



Cliente.exe

Iarchivo *IpArchivo → Proxy COM

RPC

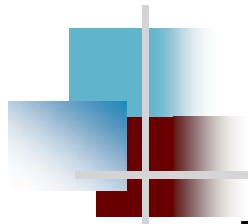
Stub COM

CMiArchivo

IArchivo

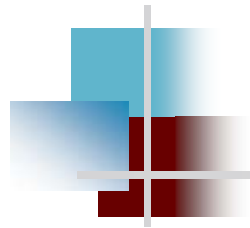
Servidor.exe

- Servidor remoto:



COM+

- DCOM + MTS (Servidor de transacciones)
- COM + se puede ejecutar en "granjas de componentes" (con balanceo de cargas, Component Load Balancing)
- Utiliza MSMQ (mensajería asíncrona entre aplicaciones) con componentes en cola (Queued Components)
- COM+ introdujo un mecanismo de eventos de (COM+ Events)
- COM+ proporciona herramientas que generar proxies del lado del cliente sea más fácil

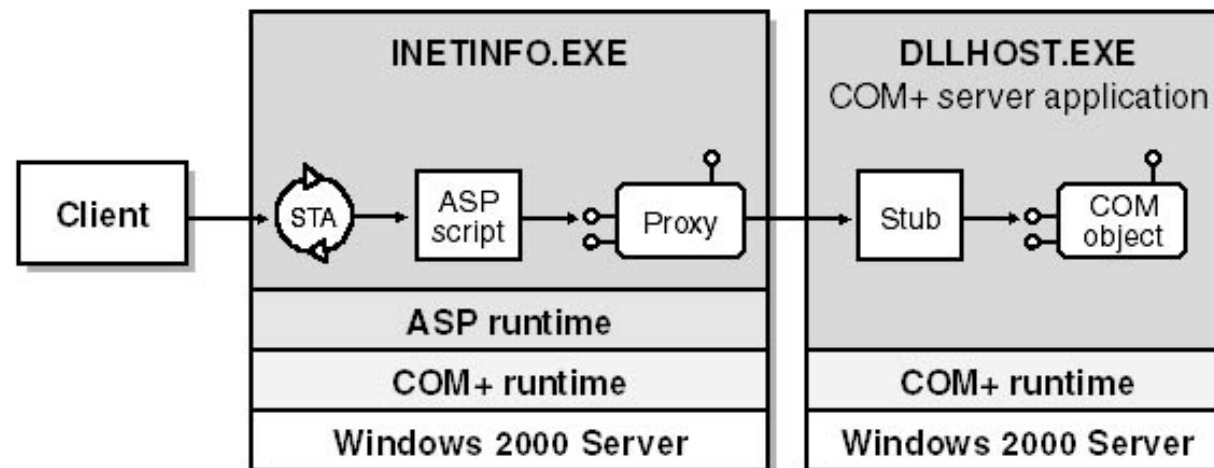
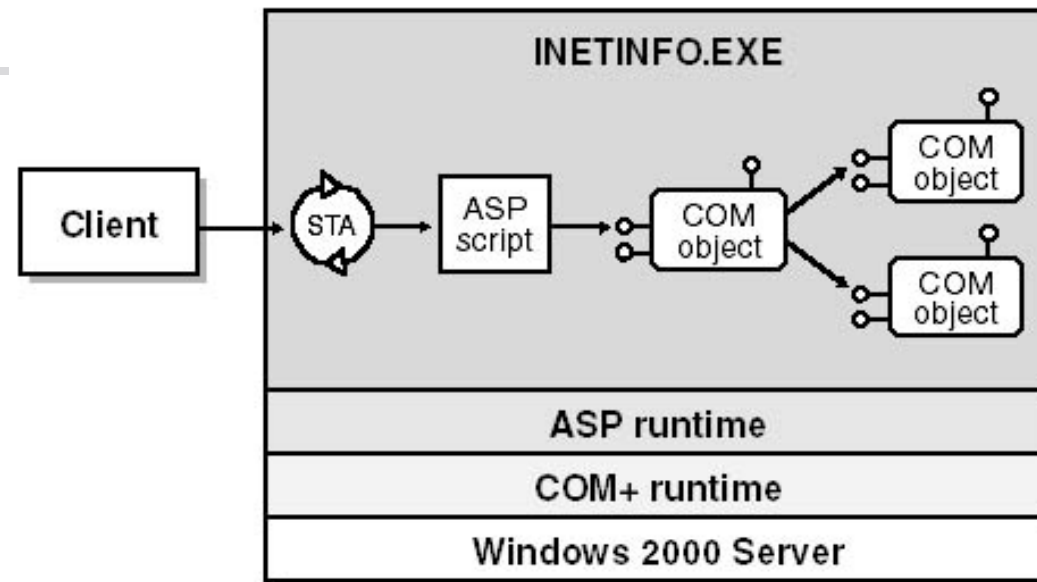


Infraestructura empresarial

- Aplicaciones utilizada por muchos tipos de personas en una organización
- Aplicaciones en la intranet y/o Internet
- Soporte para la seguridad
- Proporciona acceso a datos y comunicación a través de conexiones de red no fiables

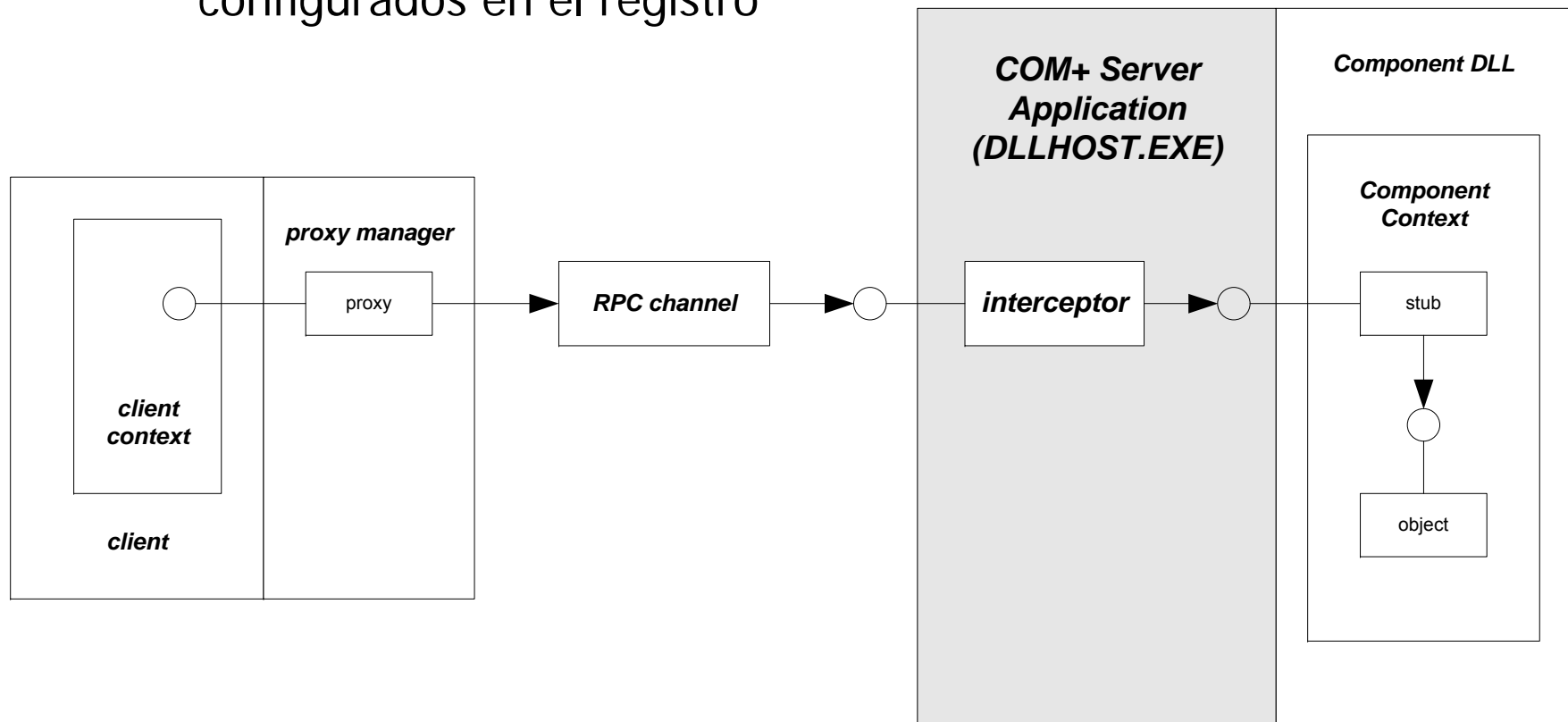
- Promueve el modelo de tres capas
 - Capa de presentación en el escritorio del cliente
 - Capa de lógica de Negocio en el servidor de aplicaciones
 - Capa de acceso a datos en los servidores de bases de datos remotos

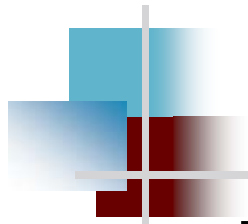
Arquitectura COM+ (Web Server)



Arquitectura COM+

- Gestor de configuración:
 - Intercepta lectura y escritura de atributos de componentes configurados en el registro





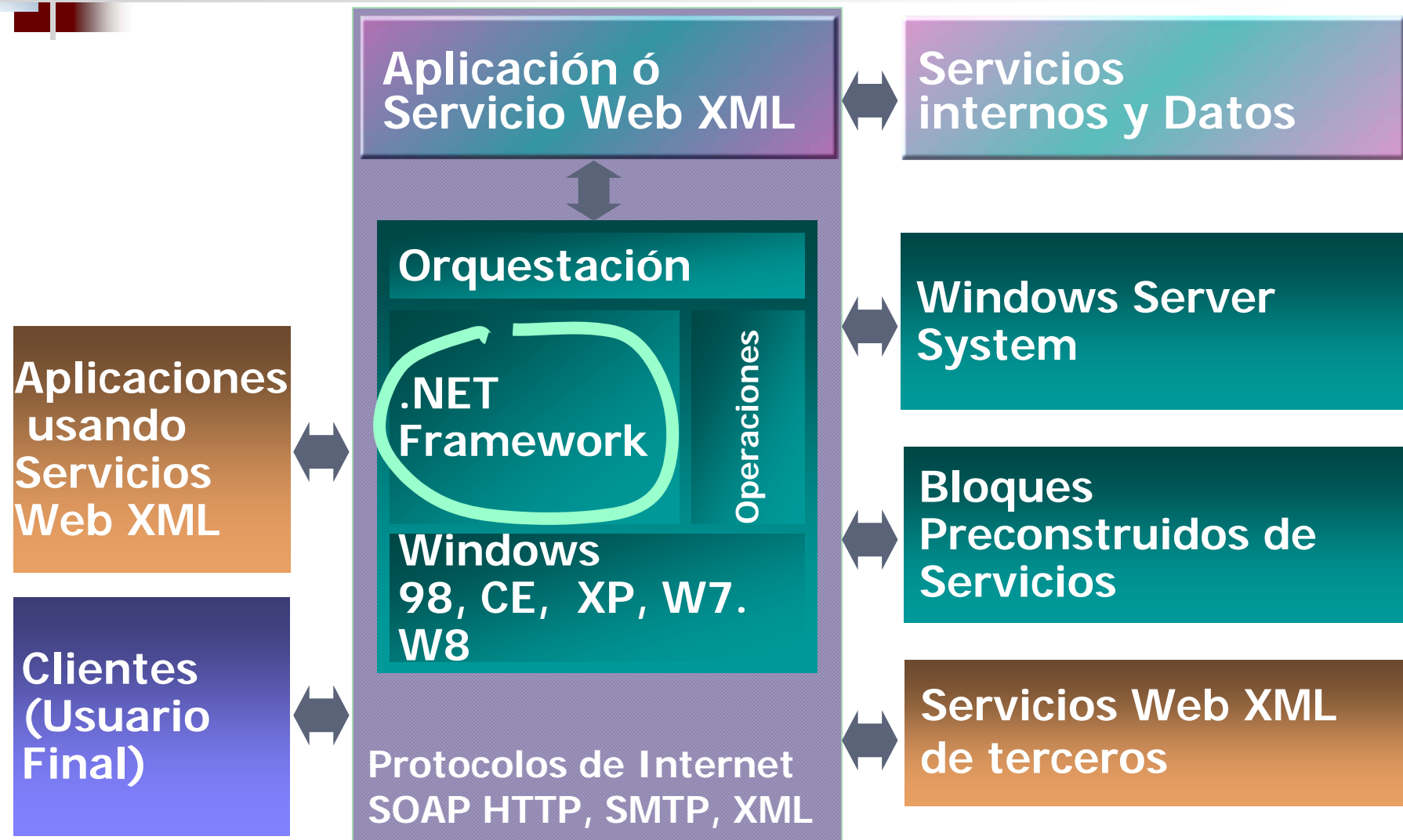
CORBA y COM+

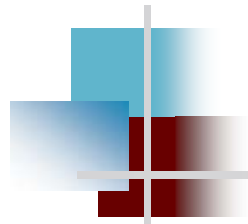
- DCOM/COM+ fue uno de los mayores competidores de CORBA.
- Los defensores de ambas tecnologías sostenían que algún día serían el modelo de servicios sobre Internet.
- Dificultad:
 - Conseguir que las conexiones funcionasen a través de cortafuegos y sobre máquinas inseguras o desconocidas
- Situación actual:
 - las peticiones HTTP combinadas con los navegadores web han ganado la partida para aplicaciones empresariales (ASP y .NET/JSP y JEE)



3.6. NET

.NET en sistemas distribuidos





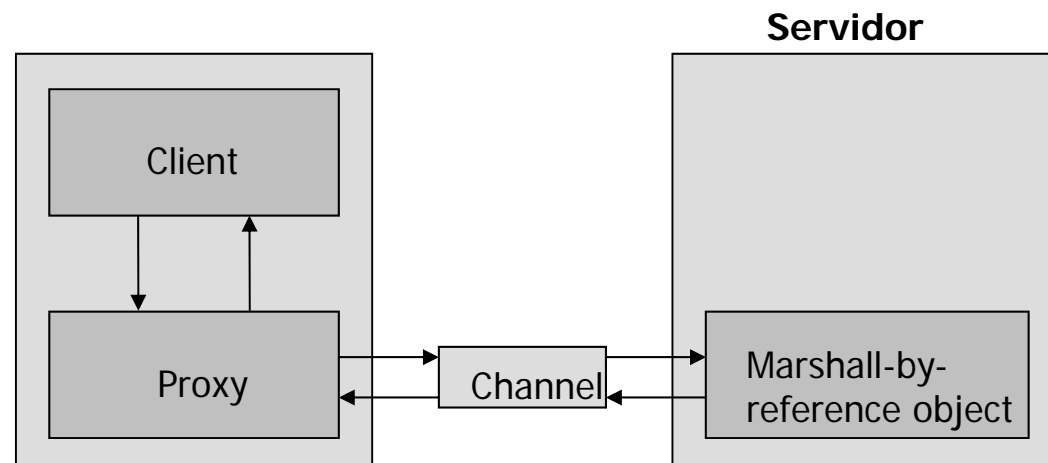
.NET Remoting

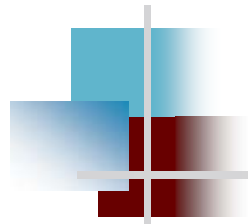
- Remoting sustituye a la tecnología DCOM para crear aplicaciones distribuidas sobre plataformas Windows.
- Remoting proporciona una arquitectura orientada a objetos, que permite desarrollar de forma sencilla aplicaciones distribuidas.
- El espacio de nombres `System. Runtime.Remoting` proporciona la infraestructura para el desarrollo de este tipo de aplicaciones

Clase remota

- Un clase es remota cuando puede ser usada por clientes en otro dominio de aplicación: en el mismo proceso, en otro proceso o en otras máquinas
- Para construir una clase remota en primer lugar hay que hacer que la clase derive de: `System.MarshalByReference`

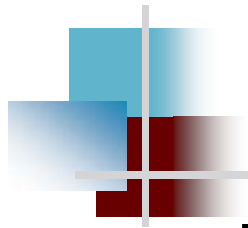
```
public class RemotableClass: MarshalByReference {  
    ...  
}
```





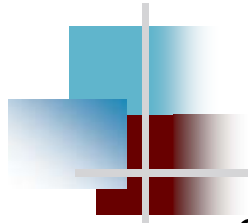
Características

- La arquitectura .NET remoting es flexible, permitiendo una fácil personalización de la aplicación.
- En vez de utilizar mecanismos propietarios Remoting soporta estándares ya existentes como:
 - SOAP (Simple Object Access Protocol)
 - HTTP y TCP como protocolo de comunicación.
- Proporciona servicios y canales de comunicación que transmiten mensajes entre aplicaciones remotas.
- Proporciona formateadores que codifican y decodifican los mensajes que se transmiten por los canales.



Canales

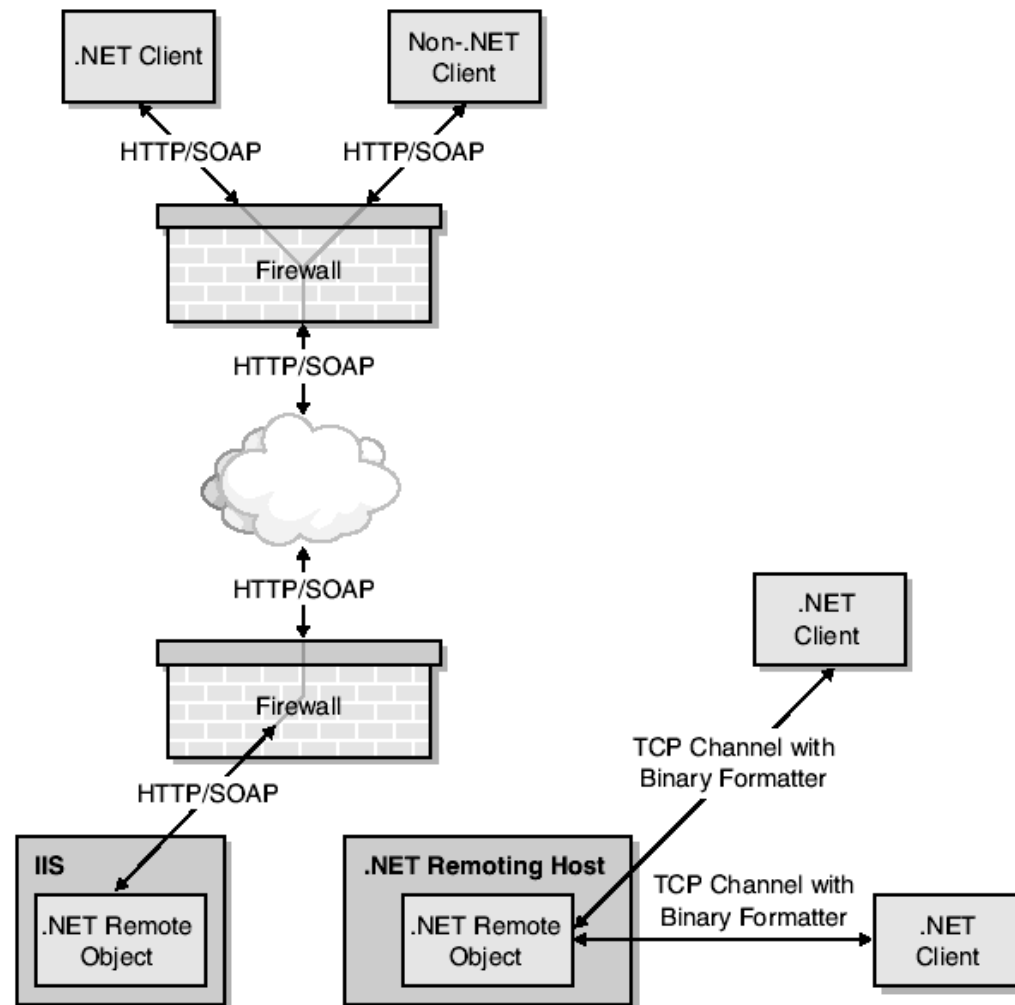
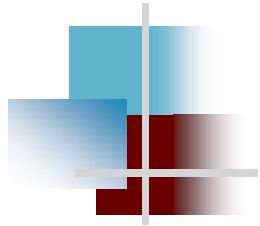
- Dos tipos de canales HTTP ó TCP.
 - Canales TCP: Usan TCP para comunicarse y transmiten datos en formato binario. Son adecuados cuando el rendimiento es lo importante.
 - El canal TCP tiene un rendimiento mayor ya que se conecta directamente a un puerto seleccionado por nosotros.
 - Canales http: Usan HTTP para comunicarse. Lo más normal es que transporten mensajes de tipo SOAP. Son adecuados cuando lo que prima es la interoperabilidad.
 - El canal HTTP es comúnmente utilizado para las comunicaciones en Internet.

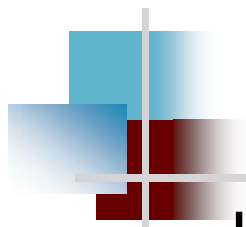


SOAP (estándar servicios Web)

- Simple Object Access Protocol
 - Gran parte de la capacidad de .NET Remoting para operar con diferentes entornos reside en SOAP.
 - Aunque no es el protocolo más eficiente, permite gran flexibilidad.
 - SOAP es un protocolo basado en XML que especifica un mecanismo mediante el cual aplicaciones distribuidas pueden intercambiar información independientemente de la plataforma
 - Aunque SOAP utiliza HTTP para su transporte, SOAP podría utilizar cualquier protocolo de transporte (pe SMTP).
 - <http://www.w3.org/TR/SOAP/>.

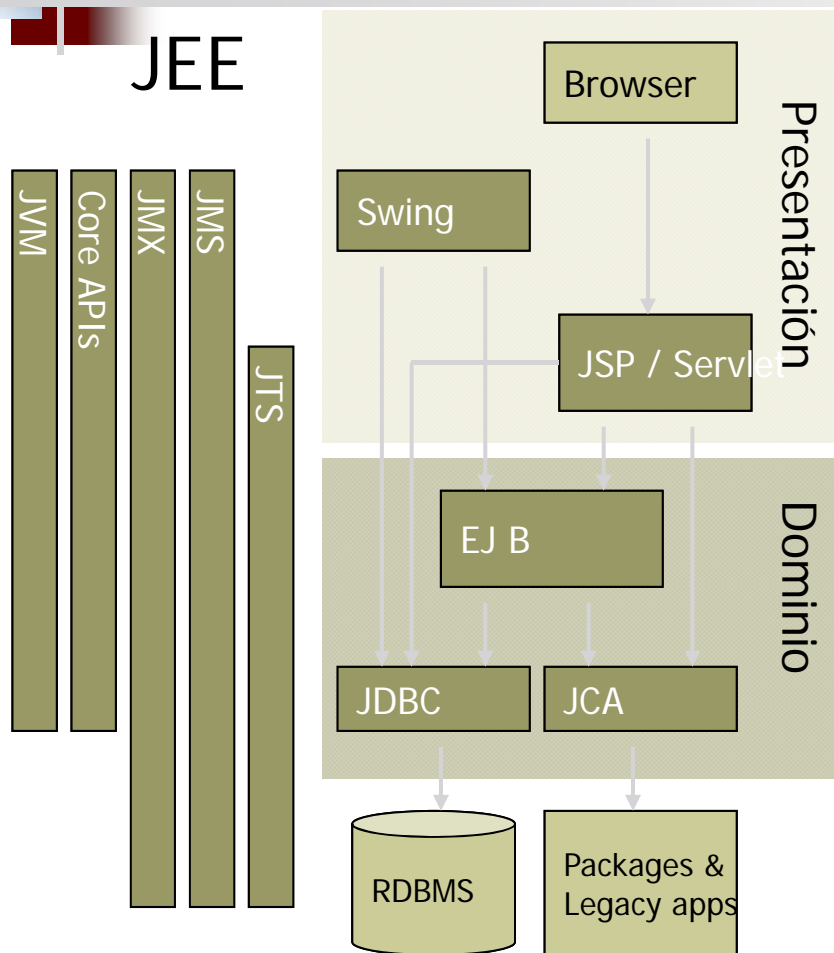
Compatibilidad vs. Rendimiento



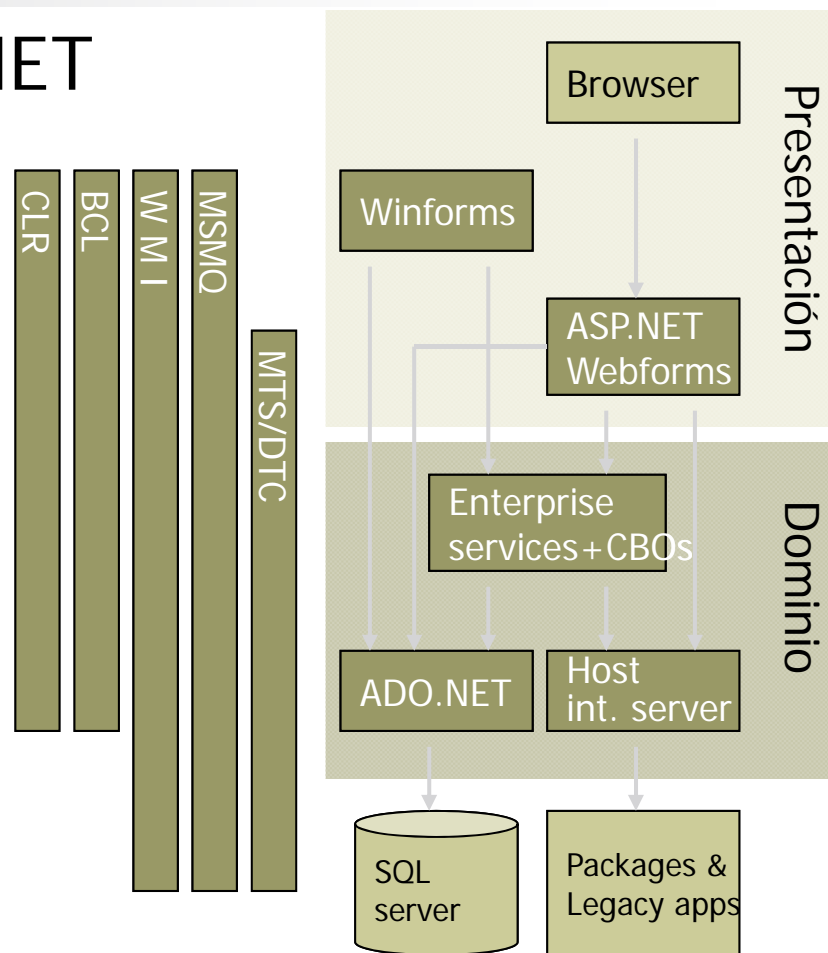


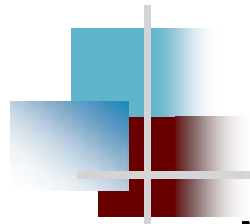
JEE y .NET

JEE



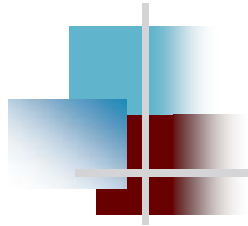
.NET





Ventajas .NET

- Multi-lenguaje
- Idiomas conceptualmente más avanzada
 - Permiten código no seguro (acceso a la memoria) y no administrado (simplificado JNI)
 - atributos (metadatos)
 - delegados (llamadas síncronas / asíncronas, Observables)
- Protocolo multi-transporte (transporte: HTTP / TCP, formato: SOAP / Binary)
- Cambiar entre una capa de presentación web y GUI más fácil
- Interfaz entre la base de datos y la capa de negocio más avanzado y potencialmente con mejor rendimiento
- AppDomains: = Código compartimentos
 - permitir que se ejecute una aplicación de n niveles en 1 proceso: serialización / intercepción automáticas entre compartimentos



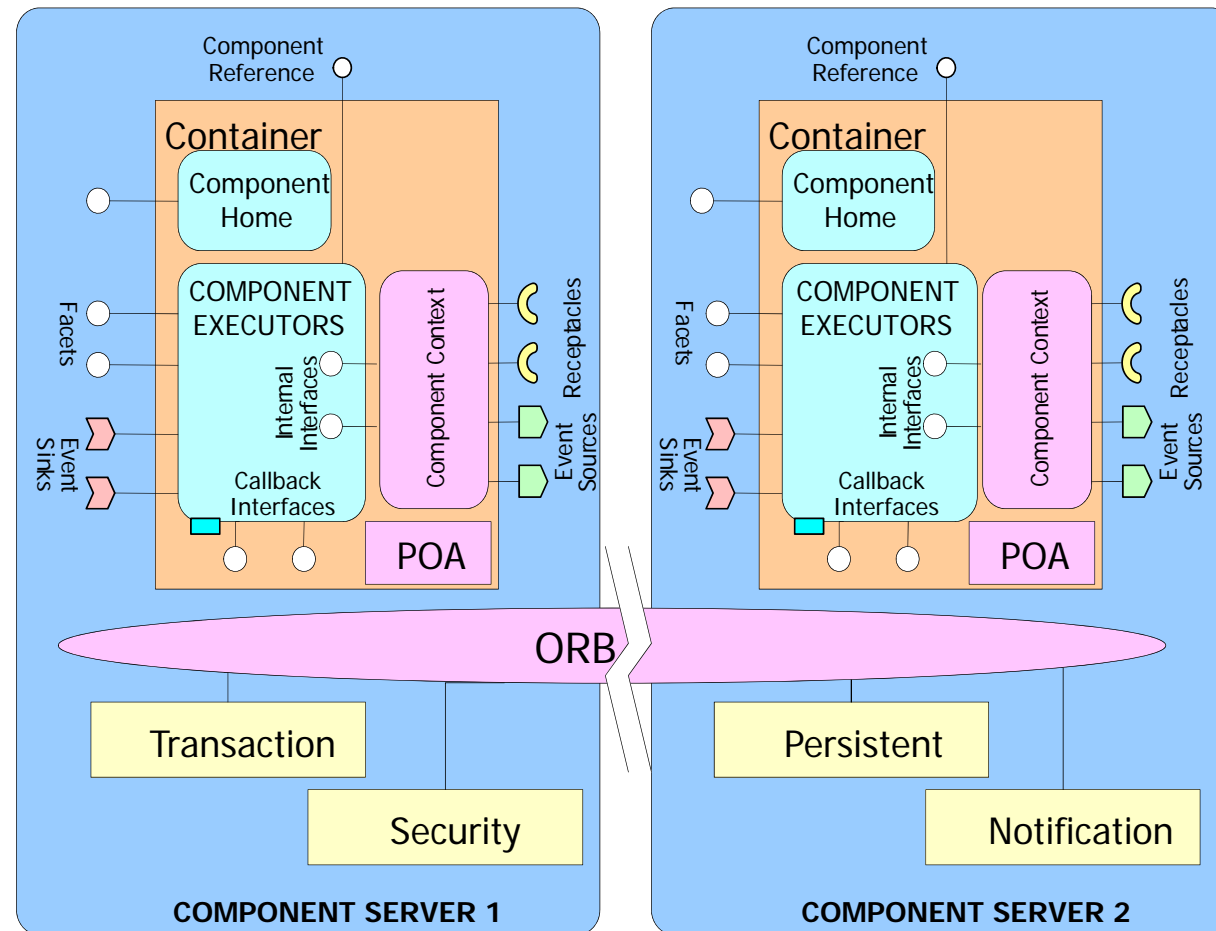
Ventajas JEE

- Madurez
 - 2-3 años ventaja (sobre todo en capa de negocio: WMI, marco de componentes)
- Intrínsecamente multiplataforma
- Implementaciones de múltiples proveedores
- Modelo de componentes distribuidos (EJB) más avanzada
- Java Connector Architecture (JCA)
- “No es de Microsoft”

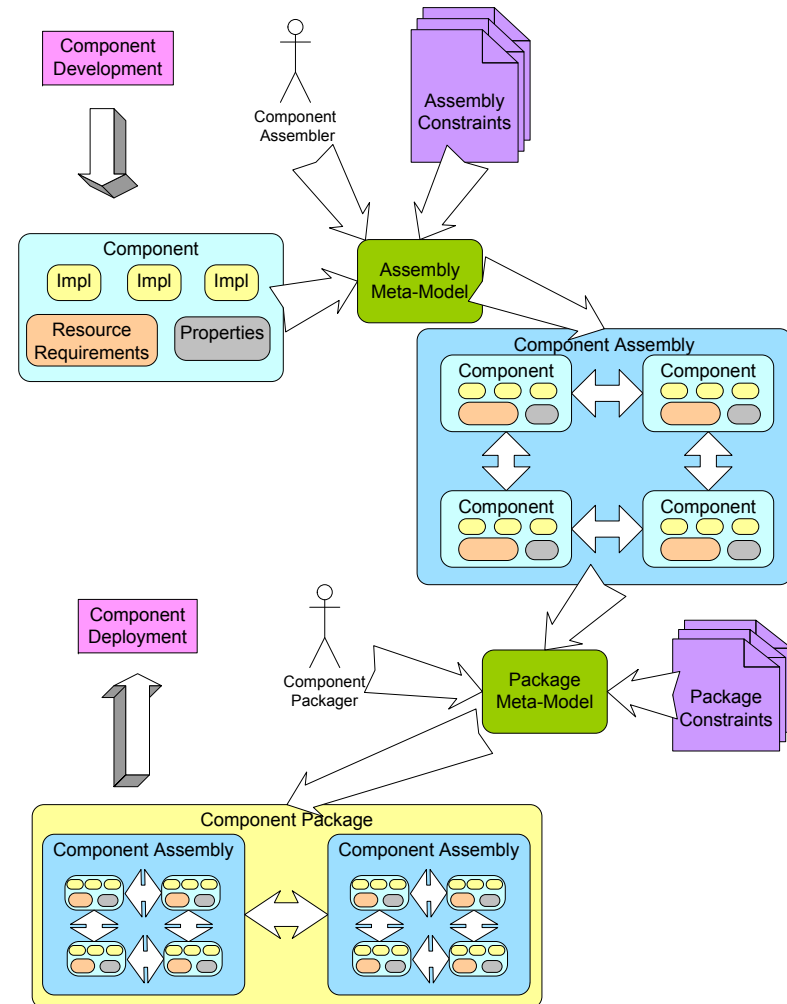
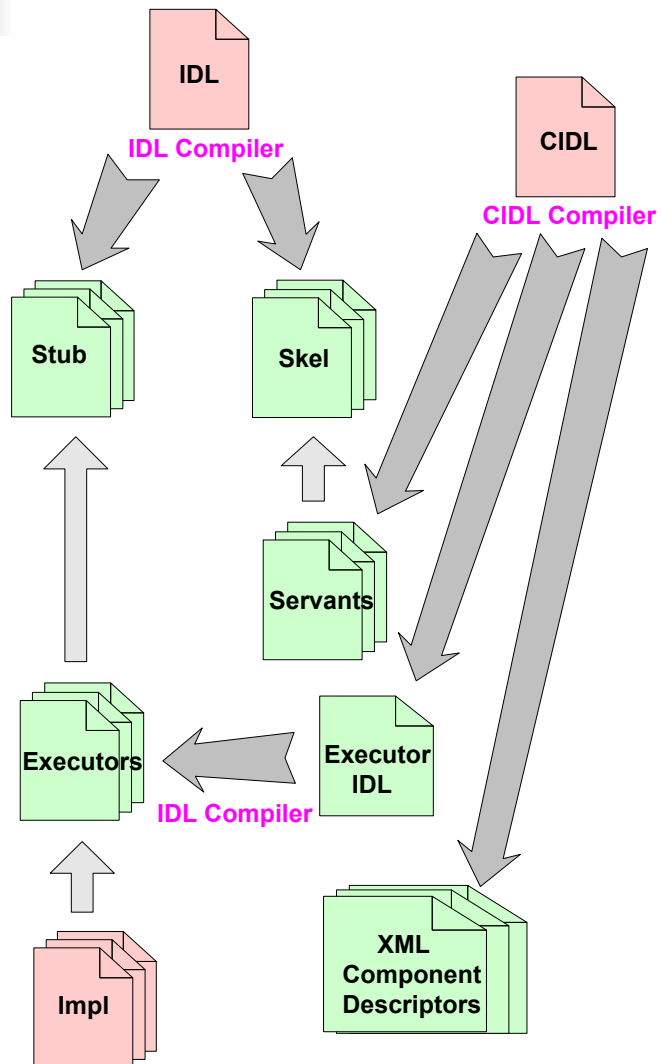


3.7. CCM

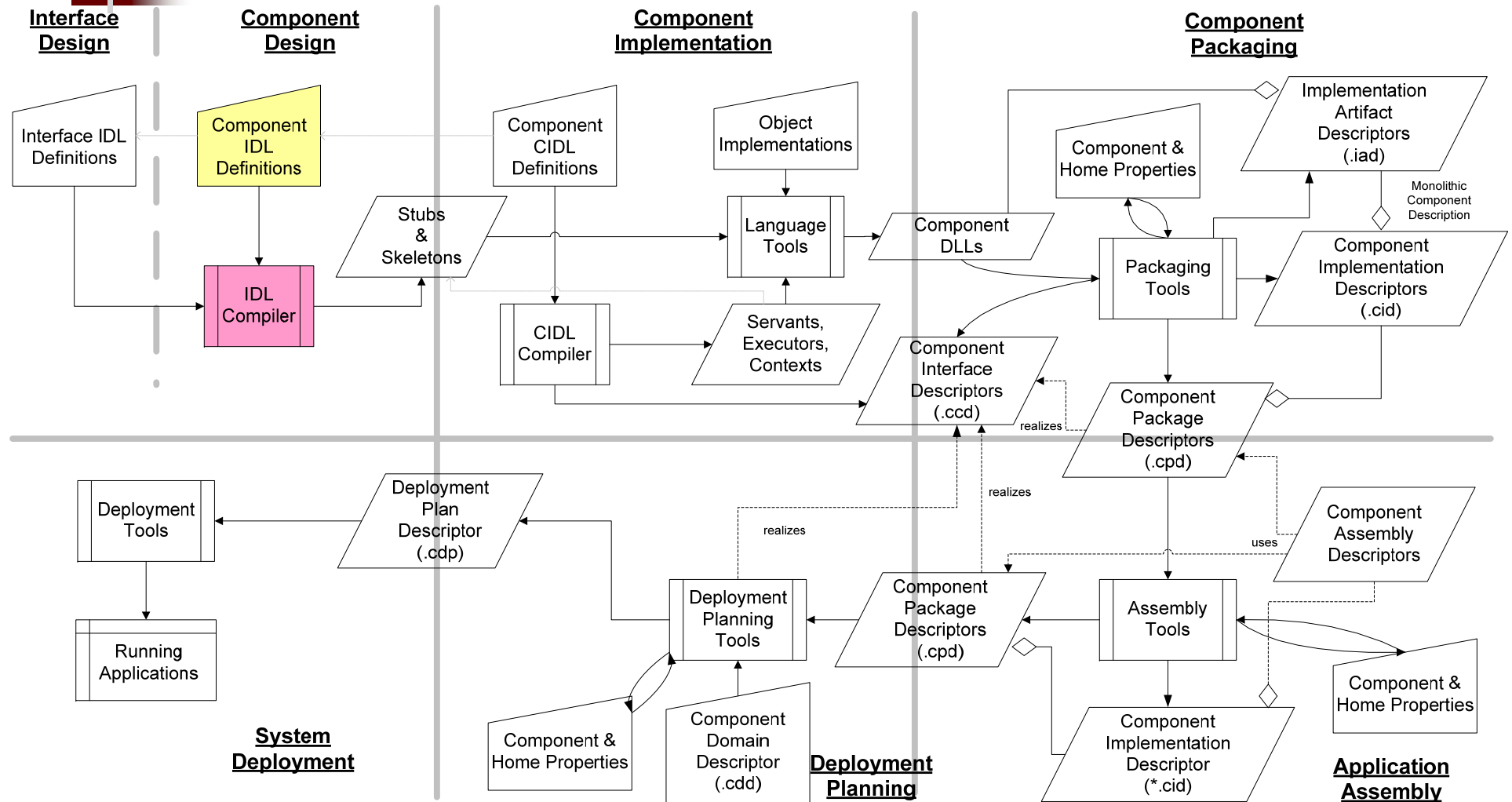
CORBA CCM



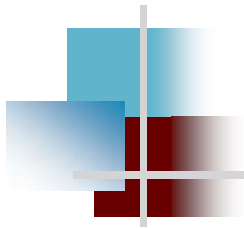
CORBA CCM IDL



Desarrollo con CCM



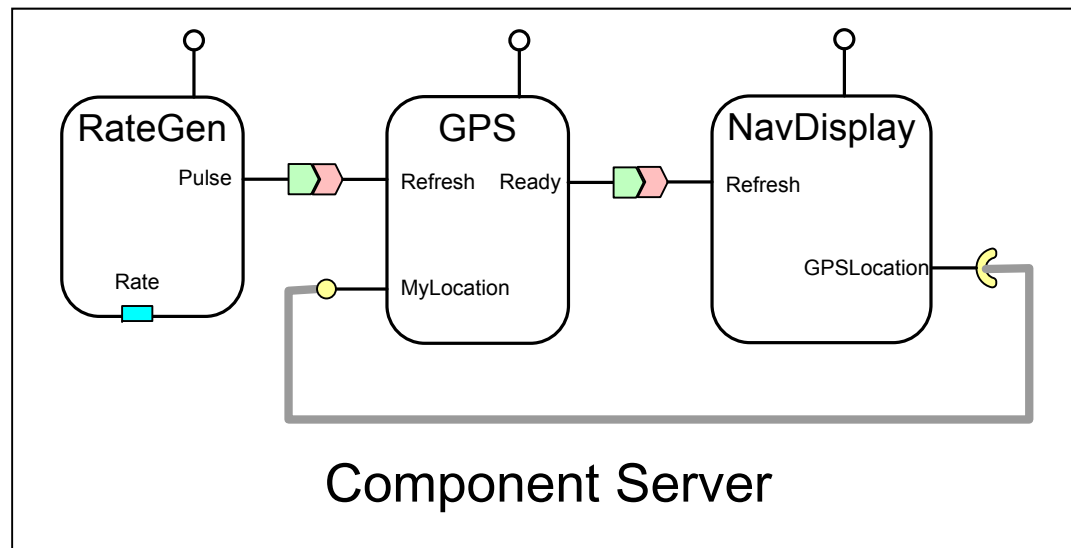
CCM: Aviónica, sistemas empotrados



Rate
Generator

Positioning
Sensor

Display
Device



■ *Rate Generator*

- Sends periodic **Pulse** events to consumers

■ *Positioning Sensor*

- Receives **Refresh** events from suppliers
- Refreshes cached coordinates available thru **MyLocation** facet
- Notifies subscribers via **Ready** events

■ *Display Device*

- Receives **Refresh** events from suppliers
- Reads current coordinates via its **GPSLocation** receptacle
- Updates display



CCM and EJB, COM, & .NET

- Like Sun Microsystems' Enterprise Java Beans (EJB)
 - CORBA components created & managed by homes
 - Run in containers that manage system services transparently
 - Hosted by generic application component servers
 - But can be written in more languages than Java
- Like Microsoft's Component Object Model (COM)
 - Have several input & output interfaces per component
 - Both point-to-point sync/async operations & publish/subscribe events
 - Component navigation & introspection capabilities
 - But has more effective support for distribution & QoS properties
- Like Microsoft's .NET Framework
 - Could be written in different programming languages
 - Could be packaged to be distributed
 - But runs on more platforms than just Microsoft Windows