

Complejidad, Componentes, implementación de framework de componentes (JEE)

Por: Luis Daniel Benavides Navarro Ph.D.

Departamento de Ingeniería de Sistemas y Computación
Especialización en Construcción de Software

Bogotá, COLOMBIA

Agenda

- Complejidad en sistemas
- Diseño por componentes
- Introducción a JEE



Complejidad

Complejidad

- Los tontos ignoran la complejidad
- Los pragmáticos la sufren
- Algunos pueden evitarla
- **Los genios la remueven**

Alan J. Perlis, "Epigrams in Programming" (1982)

Complejidad

- Difícil de definir, fácil de identificar.
- Síntomas:
 - Gran número de componentes
 - Gran numero de interconexiones
 - Muchas irregularidades
 - Una larga descripción
 - Distribución, concurrencia
 - Un equipo de diseño, implementación y mantenimiento.


Atacando complejidad

- Modularización: divida y conquiste
- Abstracción: separación de interfaces e implementación
- División por capas
- Jerarquía: Interconexión en forma de árbol
- Uniendo todo: nombres para hacer conexiones
- * Iteración
- * Manténgalo simple


Mac Pro

Design Features Performance The Environment Tech Specs [Buy Now](#)


Designed to be custom designed.




Expansion




PCI Express



Storage




Memory



Easy I/O

Expansion made easy.

The easy-access interior of the Mac Pro feels like the well-organized workstation it is. No rat's nest of components here. You don't need to turn the system on its side or struggle to reach into awkward spaces to make changes. Just remove the side panel for instant access to everything. Slide out the processor tray to add memory. Slide out drive bays to add storage. Slide a simple bar to change up to four expansion cards at once. And with plenty of I/O ports both front and back, you'll have room for all your external devices.



Diseño basado en componentes

Diseño basado en componentes

- Tres principio básicos [Hei2001]:
 - Los sistemas son ensamblados con partes preconstruidas y no se construye cada componente desde 0.
 - Las partes se pueden reutilizar en otros sistemas.
 - Las partes se pueden mantener y adaptar fácilmente para producir nuevas funcionalidades y características.

Algunas definiciones

- Ver [Hei2001]:
- Un **componente de software** es un elemento de software que adopta un modelo de componente.
- Un **modelo de componentes** define estándares de interacción y composición.
- La **implementación de un modelo de componentes** es la plataforma que permite la ejecución de los componentes.

Pero que es un componente

- Función, procedimiento?
- Clase?
- Paquete?
- Módulo?
- Librería?
- Aplicación?
- Sistema?
- Todos los anteriores?

Ejemplos de componentes

Plataforma de componentes	Componentes	Lenguaje de interfaz	Protocolo de interacción	Composición	Entidad que define
Linux OS	Aplicaciones	API del OS	API del OS	Por medio de APIs	linuxbase.org
CORBA	Componentes CORBA	IDL	IIOP	Remote method invocation	OMG
JEE	EJB, Servlets, Facelets	Java + anotaciones	RMI, IIOP	Remote method invocation	ORACLE /Sun
.Net	.NET components	Varios, e.g., C#	Propietario	Remote method invocation	Microsoft
Internet / Web Services	Web Services	WSDL	SOAP	Remote method invocation	W3C

Ejemplo de un Modelo de componentes: Introducción JEE

Motivación

- Aplicaciones Empresariales
 - **Distribuidas**
 - **Concurrentes**
 - Soporte a múltiples atributos de calidad
 - Seguridad,
 - Disponibilidad,
 - Interoperabilidad,
 - Escalabilidad, ...

JEE

- Orientada a aplicaciones empresariales
- Multicapa
- Presenta una arquitectura de componentes,
- APIs multipropósito: Presentación WEB, Componentes/Servicios, Transacciones, persistencia

Características

- Su objetivo es proveer a los programadores un conjunto de APIs que ofrecen
 - Reducción del tiempo de desarrollo
 - Reducción de la complejidad
 - Aumento de la velocidad
- Introduce un modelo simplificado de programación
- Uso de XML y Anotaciones
- Programación basada en POJOs (Plain Old Java Objects)
- Inyección de dependencias
 - Contenedor JEE automáticamente inyecta referencias a otros componentes requeridos

Historia de una solución I

- 1995 soporte para applets en NetScape
 - Applets: Aplicaciones Java corriendo en Browsers
 - Rápida adopción
 - Rápida caída debido a problemas de comunicación y compatibilidad
- 1997 Adopción de arquitectura Ultra Thin Client (UTC)
- 1997 Introducción del Java Web Server con soporte de servlets
 - Acceso no transaccional a Base de datos (JDBC)

Historia de una solución II

- 1998 Modelo de Componentes EJB
- 1998 Soporte de transacciones (JTA)
- 1998 Soporte a mensajería JMS
- 1998 Soporte a Nombre e interface de directorio (JNDI)

Historia de una Solución

- 1999 J2EE Creación de un estándar con versiones de API definidos
- 2001 Arquitectura de conectores para conectarse con otros EIS
- 2003 Soporte de Web services
- 2006-2009 Rediseño, Adopción de estándares y facilidad de uso

Mecanismos principales de Java EE

- POJOS + Anotaciones
- Convención vs. configuración
- Inyección de dependencias
- Aspect-Oriented Programming / Interceptores
- Java Persistence API

POJOs

- POJO: Plain Old Java Object
 - Campos
 - Metodos de acceso (getXX, setXX)

Anotaciones

- Mecanismo para soportar meta-data en clases
- Permite adicionar metadata en los diferentes modelos de componentes soportados

Convención vs. Configuración

- Configure solo comportamiento que se desvía de la convención
- Clases + anotaciones vs. Grandes archivos descriptores en XML
- Anotaciones permiten chequeo del compilador (type safety)
- Por ejemplo si no se especifica transacciones el contenedor aplica:
 - `@TransactionAttribute(TransactionAttributeType.REQUIRED)`

IoC + Inyección de dependencias

- Código para soportar la infraestructura de componentes ya no se codifica, simplemente se inyecta.

Antes:

```
try{
    Context ctx= new InitialContext();
    Object proxy = ctx.lookup("service");
    ServiceHome home = (ServiceHome)
        PortableRemoteObject.narrow(
            proxy,ServiceHome.class);

    Service service = home.create();
}catch(Exception ex){ ... }
```

Ahora:

```
@EJB
private Service service
```

Aspect-Oriented programming

- Separación de preocupaciones entrecortantes. (crosscutting concerns)
- Por ejemplo el código de logging esta entremezclado con el código de negocio
- Los Aspectos =
 - Pointcut: construcción con lenguaje declarativo que intercepta eventos
 - Advice: Acción a realizar antes, en vez, o después de los eventos
- En JEE utilizamos Interceptores:
 - Pointcut: Declarado con Anotaciones en los beans, o si se quiere en archivos descriptores.
 - Advice: Clases interceptoras que definen acciones específica, e.g., Tracing.

Java Persistence API

- Usa inyección de dependencias intensivamente
- Provee un lenguaje para hacer queries
- Mapeo entre Object Orientation y bases de datos relacionales
- El contenedor maneja automáticamente transacciones, persistencia
- Entidad = Tabla en base de datos
- Instancia de entidad = Fila en tabla de base de datos

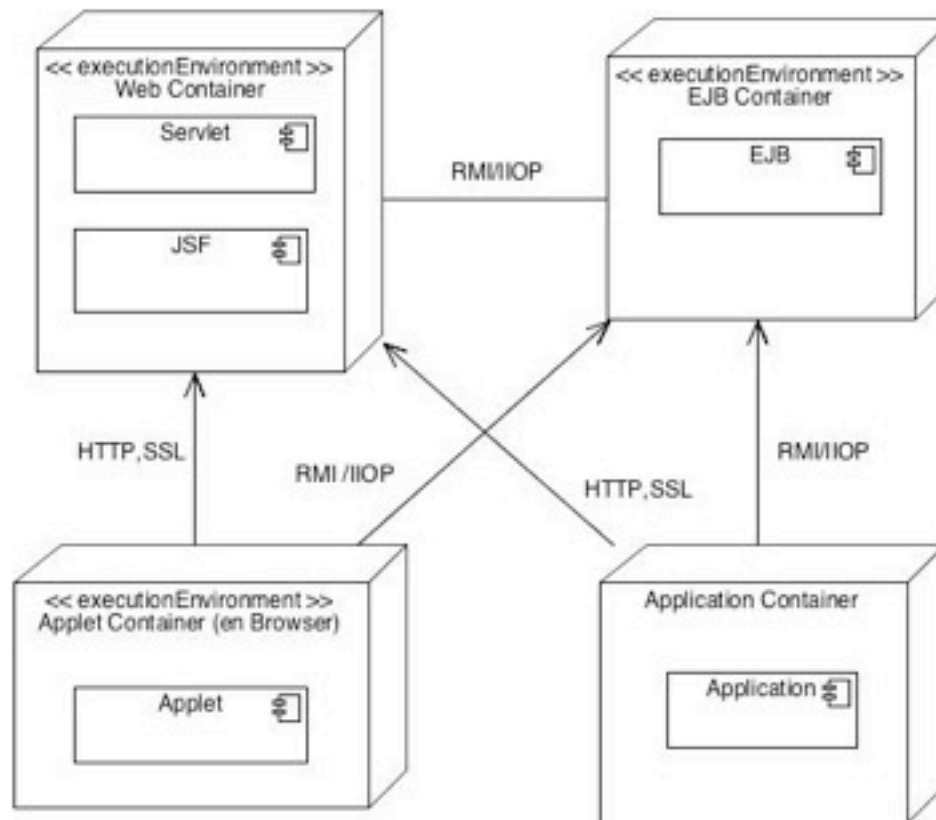
Arquitectura JEE

Aplicaciones distribuidas y multicapa en JEE

Arquitectura básica

- Diferentes modelos de componentes soportados por **contenedores**
- Aplicaciones construidas en **capas**

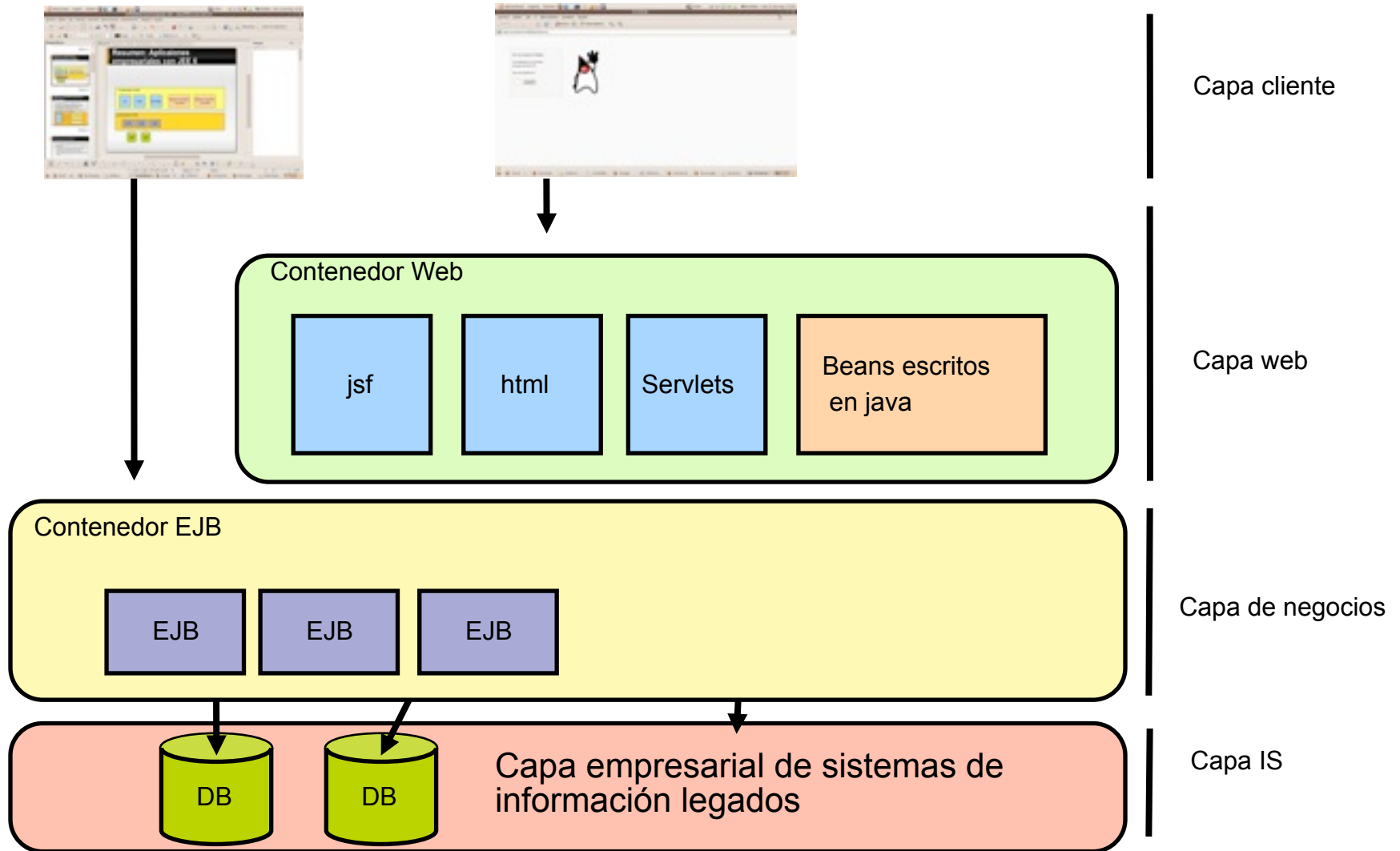
Arquitectura de contenedores



Contenedor provee servicios transversales

- Acceso a Recursos (JNDI)
- Seguridad
- Transacciones/Concurrencia
- Manejo de sesiones
- Persistencia *

Separación en capas



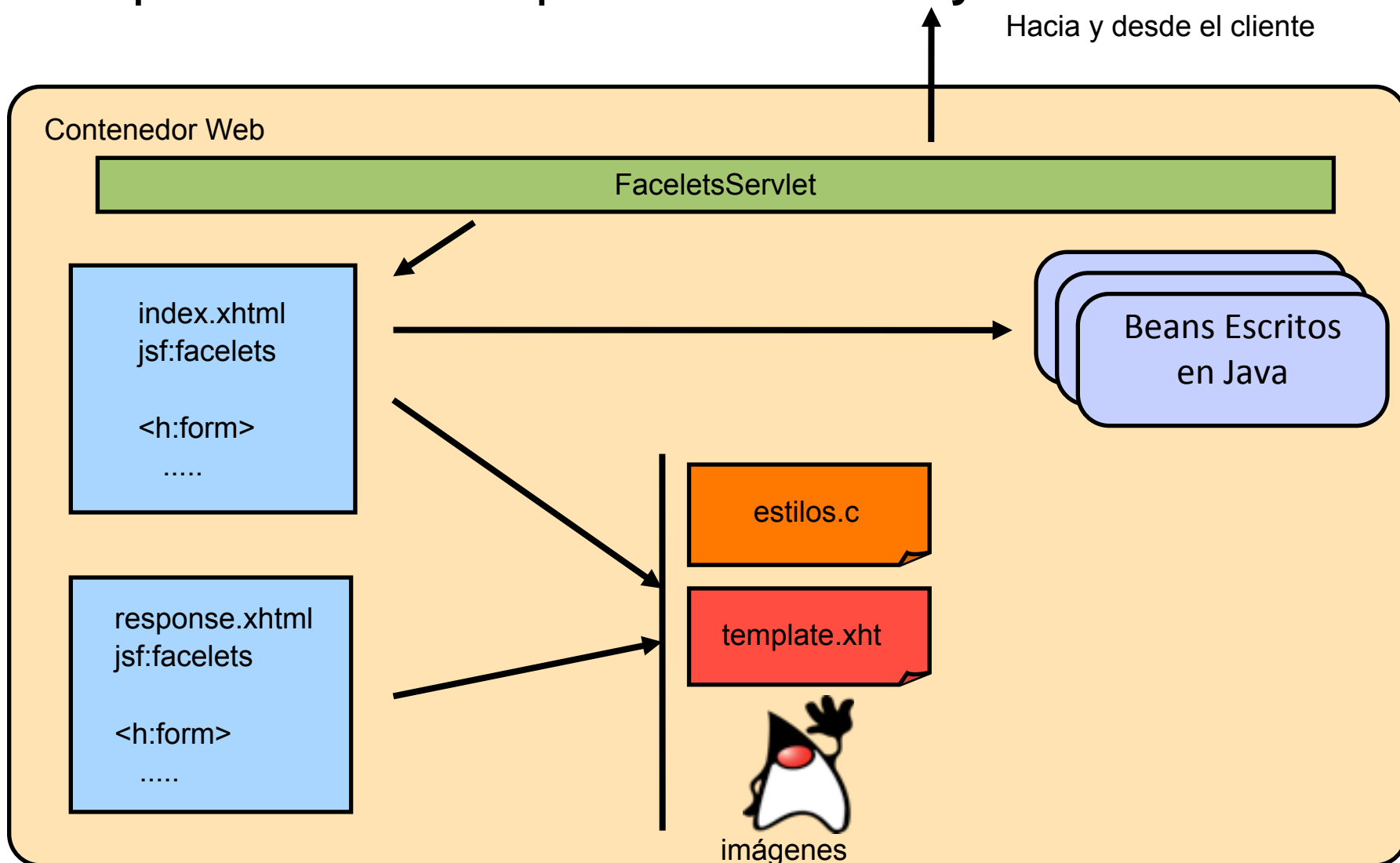
Capa Cliente

- Aplets corriendo en browsers
- Clientes Java que se ejecutan en la máquina del usuario

Capa Web

- Servlets
- Java Server Faces
- Java Server Pages (Deprecated)

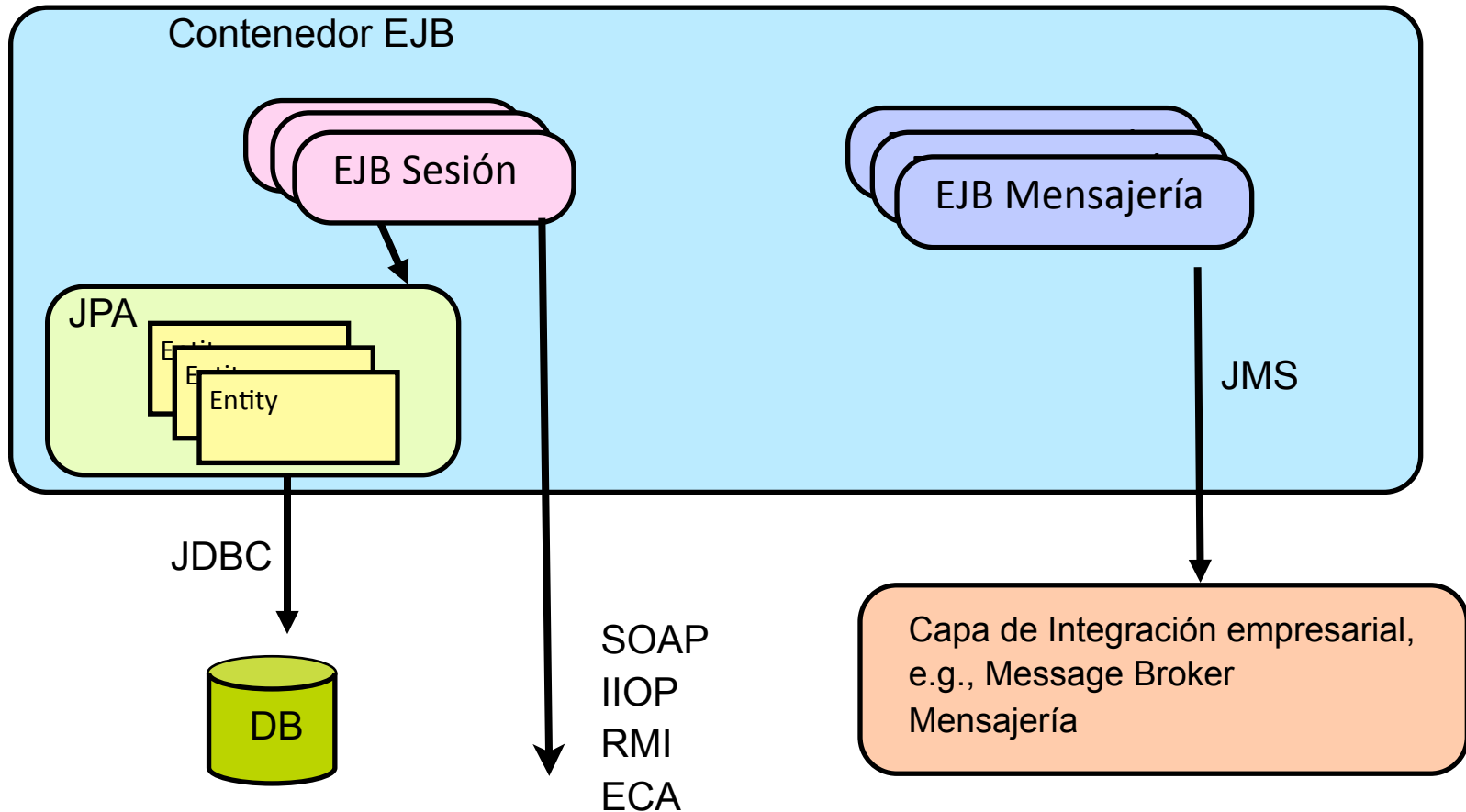
Componentes de la capa web utilizando Java Server Faces



Capa de negocio

- Java Enterprise Bean
 - Un componente de servidor
 - Encapsula la lógica del negocio

Capa de negocio



Beneficios de EJB

- Contenedor provee servicios transversales, e.g., transacciones, seguridad
- Separación de responsabilidades: Negocio (servidor), cliente (presentación)
- Portables y se pueden ensamblar

Cuando usar EJB

- La aplicación tiene que ser escalable y lista para correr en clusters de computadores
- Las transacciones son necesarias para garantizar integridad de los datos
- Muchos tipos de clientes

Capa de Sistemas de Información Empresarial

- Bases de Datos
- Sistemas Legados

Bibliografía

- [Bie2009] Adam Bien. Real World Java EE Patterns – Rethinking Best Practices. Editorial: lulu.com, 2009.
- [Hoh2003] Gregor Hohpe, Bobby Woolf. Enterprise integration patterns: designing, building, and deploying messaging solutions. Addison–Wesley Professional, 2003.
- [Hei2001] Heineman, Council. Component–Based Software Engineering: Putting the Pieces Together. Addison–Wesley Professional, 2001.
- [Szy2002] Szyperski. Component Software: Beyond Object–Oriented Programming. Addison–Wesley Professional; 2 edition, 2002.
- [Sal2009] SALTZER, Jerome., KAASHOEK, Frans. Principles of Computer System Design: An Introduction. Editorial Morgan Kaufmann, 2009.
- [JEE] The Java EE 6 Tutorial: Basic and Advanced Topics. Oracle – Sun Microsystems. Disponible en: <http://download.oracle.com/javaee/6/tutorial/doc/>
- ACM Digital Library (<http://portal.acm.org/dl.cfm>). Acceso a artículos científicos solo con suscripción.

Luis Daniel Benavides Navarro

dnielben@gmail.com

