UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# De los objetos a los componentes

Dr. José Luis Zechinelli Martini
joseluis.zechinelli@udlap.mx
LIS – 3061

Gracias a la Dra. Genoveva Vargas Solar, CNRS-LIG, Francia
y al Prof. Sacha Krakowiak, UJF, Grenoble, Francia

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

## Desarrollo de aplicaciones distribuidas

- Objetivos:
  - Facilidad de desarrollo (bajo costo)
  - Facilidad de evolución, escalabilidad
  - *Openness* (integración, sistemas heterogéneos, etc.)
- Soluciones posibles:
  - Se puede usar *middleware* orientado a objetos…
    - Java RMI, CORBA, DCOM, *Message Bus*
  - … debe extenderse
    - Construcción modular para evolución y *openness*
    - Servicios comunes (evitar « reinventar la rueda » y concentrarse en la aplicación)
    - Herramientas de desarrollo (desarrollo de programas, composición)
    - Herramientas de instalación (generación e instalación de elementos)
    - Herramientas de administración (observación, re-configuración)
  - El *middleware* basado en componentes persigue proveer esas extensiones

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Ejemplo: Museo virtual

- El museo de arte contemporáneo organiza exposiciones temporales:
  - Las exposiciones se abren al público durante un periodo del año (primavera, verano, otoño, invierno) en salas diferentes
  - Cada exposición tiene un tema relacionado con un movimiento artístico particular, *e.g.*, surrealismo, cubismo; por ello, un especialista del tema se ocupa de escoger las obras de arte que se presentarán
- Para cada movimiento, existe una oficina que se ocupa de la colección y de las actividades relacionadas usando aplicaciones que administran:
  - Colecciones (mantenimiento, exposición, préstamo, compra, venta y restauración)
  - Exposiciones temporales y permanentes
  - Comunicación con la aplicación de administración de presupuesto del museo
- **Objetivo**: construir un sistema que integre las aplicaciones de cada movimiento

3

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Limitaciones del paradigma OO

- No hay visión global de la aplicación:
  - Conceptos principales definidos para un solo objeto
  - No hay descripción global de la arquitectura
- Evolución difícil:
  - Consecuencia de no tener una vista global
- Faltan servicios:
  - Los servicios requeridos deben implementarse a mano (persistencia, seguridad, tolerancia a fallas, etc.)
- Faltan herramientas (de composición, de instalación)
- Conclusión:
  - Carga importante para el programador
  - Incidencia en la calidad de la aplicación
  - Una parte del ciclo de vida no está cubierto

4

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Plan

- Componentes:
  - Definición
  - Modelo genérico
- *Enterprise Java Beans* (EJB):
  - Esquema de ejecución
  - Roles y contratos
  - Ciclo de vida administrado
- Desarrollo de una aplicación con EJB
- Conclusión sobre EJB

5

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Definición de componente

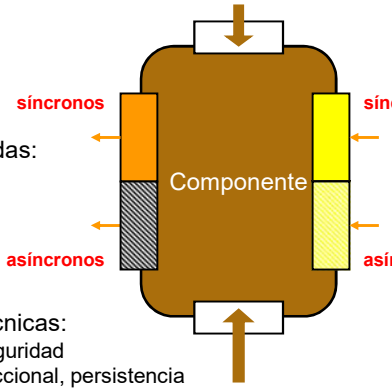
- Módulo de software autónomo:
  - Unidad de instalación (instalación en varias plataformas)
  - Unidad de composición (combinación con otros componentes)
- Propiedades:
  - Especifica explícitamente las interfaces ofrecidas (atributos, métodos)
  - Especifica explícitamente las interfaces requeridas
  - Puede ser configurado
  - Se auto-describe
- Beneficios:
  - Permite la construcción de aplicaciones por composición de piezas elementales configurables
  - Separación de las funciones de un proveedor de componentes y de un ensamblador (condiciones de desarrollo de una industria de desarrollo de componentes)

6

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Modelo genérico

Propiedades configurables
(Interfaces específicas con acceso restringido)




Interfaces ofrecidas:
- Atributos
- Métodos

Interfaces requeridas
(Ofrecidas por otros componentes)

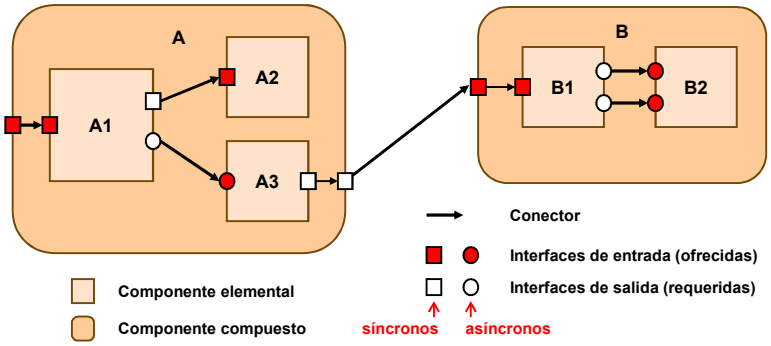
Restricciones técnicas:
- Instalación, seguridad
- Acceso transaccional, persistencia
- Interfaces sistema (librerías, etc.)

7

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Usando componentes

- Composición y encapsulación jerárquica:
  - Componentes compuestos
  - Sub-componentes
- Interconexión de componentes:
  - Conectores
  - *Binding objects*




8

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

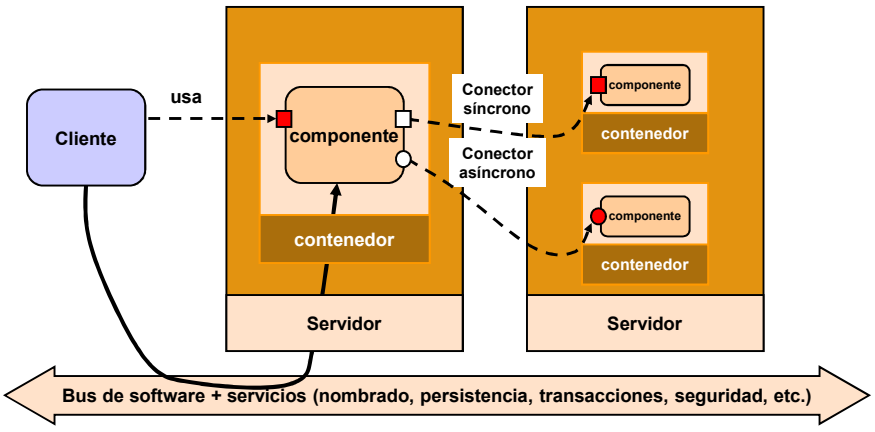
# Soporte de software

- Contenedor:
  - Encapsula uno o más componentes
  - Se ocupa de los servicios sistema
    - Nombrado, seguridad, transacciones, persistencia, etc.
  - Se ocupa (parcialmente) de las relaciones
    - Invocaciones, eventos
  - Técnicas involucradas: interposición, delegación
- Servidor:
  - Espacio de ejecución para contenedores y componentes
  - Mediador entre contendores y servicios sistema

9

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Componentes en acción




10





# Plan

- ✓ Componentes:
  - Definición
  - Modelo genérico
- *Enterprise Java Beans* (EJB):
  - Esquema de ejecución
  - Roles y contratos
  - Ciclo de vida administrado
- Desarrollo de una aplicación con EJB
- Conclusión sobre EJB

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# *Enterprise Java Beans* (EJB)

- Objetivos
  - Facilitar la construcción de programas:
    - Para servidores empresariales → el tercio de en medio de una arquitectura a 3 tercios
    - Ensamblando componentes reutilizables
  - Proveer un ambiente:
    - Para servicios comunes (persistencia, transacciones, etc.)
    - Que permite al desarrollador concentrarse en problemas específicos de la aplicación
  - Favorecer el desarrollo de una industria de componentes:
    - Separando las funciones de producción de componentes del ensamblado de aplicaciones y de la provisión de servicios

13

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# *Enterprise Java Beans* (EJB)

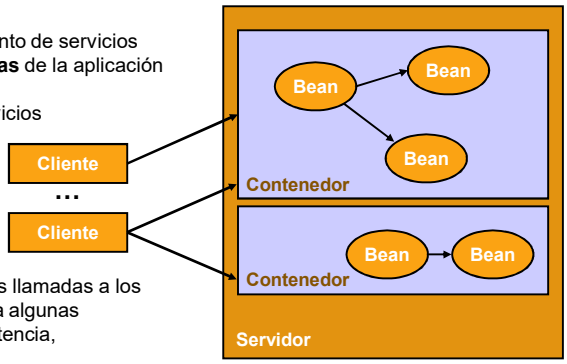
- Visión general:
  - Dos tipos de *Enterprise Beans*:
    - *Entity Bean* que representa un objeto, usualmente persistente
    - *Session Bean* que representa una secuencia de acciones para un cliente
  - Ambiente = servidor + contenedor

14

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Esquema de ejecución

- Los ***beans*** proveen un conjunto de servicios ejecutando **tareas específicas** de la aplicación
- Los **clientes** usan estos servicios

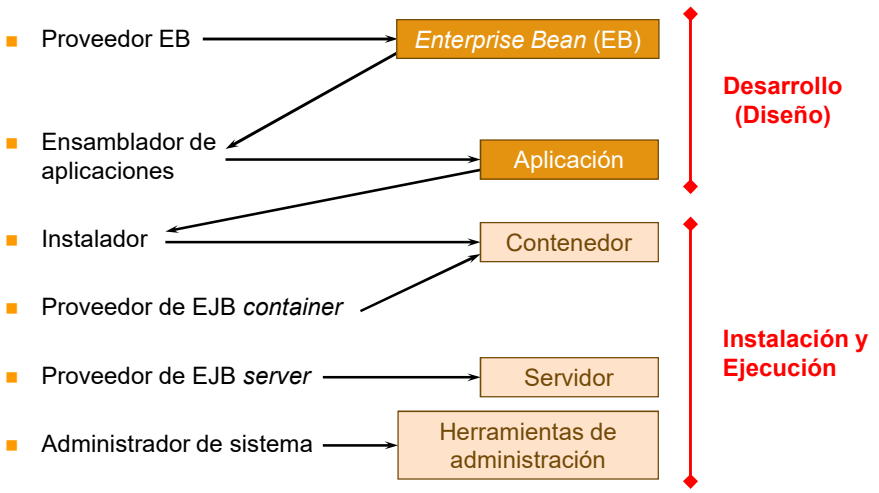



- Un **contenedor** intercepta las llamadas a los *beans* que contiene y ejecuta algunas **funciones comunes** (persistencia, transacciones, seguridad)
- Los contenedores aíslan a los *beans* a) de los clientes y b) de una implementación específica del servidor
- El **servidor** es un **mediador** entre el sistema y el contenedor (permite al contenedor ejecutar sus funciones llamando primitivas del sistema)

15

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Roles de EJB*



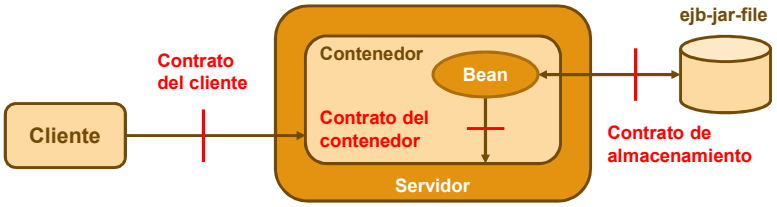

* Documentos JOnAS: http://www.objectweb.org/jonas/

16

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Contratos EJB

- Se asocian “contratos” a cada *bean*:
  - Deben ser respetados por el cliente, el contenedor y el *bean*
  - Contrato del lado del cliente
    - Provee una vista uniforme del *bean* al cliente (esta vista es independiente de la plataforma de instalación)
  - Contrato del contenedor
    - Permite a los *beans* ser portados en diferentes servidores
  - Contrato de almacenamiento (*packaging*)
    - Define un formato estándar de archivo (**ejb-jar-file**) para almacenar los *beans*. Este formato debe ser respetado por todas las herramientas




17

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

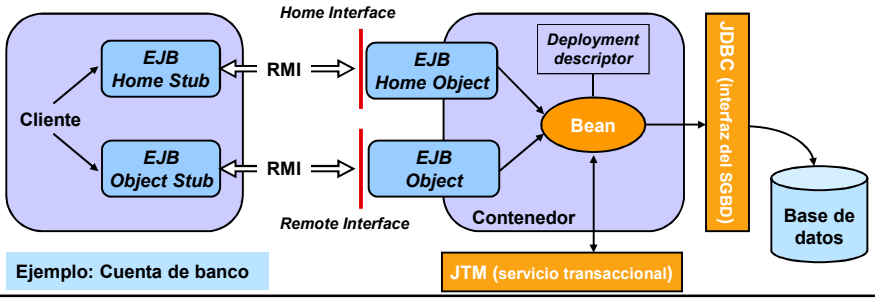
# Contrato del lado del cliente

- Encontrando el *bean*:
  - usar JNDI Java (interfaz al servicio de directorio)
- Usando el *bean*:
  - Vía interfaces estándares ofrecidas por el desarrollador de *beans*
    - ***Home interface*** (métodos para la administración del *bean*: *create*, *remove*, *find*, etc.)
    - ***Remote interface*** (métodos específicos de la aplicación)
- El contenedor implementa un mecanismo de delegación:
  - El cliente no llama directamente al *bean*, sino al contenedor
  - El contenedor reenvía las llamadas

18

# Ejecución: Vista detallada

- Un *Object* EJB y un *Home Object* EJB se asocian a cada *bean*:
  - El *Home Object* EJB se encarga del ciclo de vida (creación, identificación, búsqueda, destrucción), se localiza a través de un directorio (interfaz JNDI)
  - El *Object* EJB implementa la interfaz hacia los servicios del *bean*, intercepta las llamadas dirigidas al *bean* y realiza tareas (transacciones, persistencia, administración del estado, seguridad) de acuerdo a lo que especifica el *deployment descriptor*




Ejemplo: Cuenta de banco

# Contrato del contenedor

- Un contenedor implementa las funciones asociadas a sus *beans*:
  - Gestión del ciclo de vida, gestión del estado, seguridad, transacciones, concurrencia, etc.
  - Los servicios llaman a métodos provistos por el *bean* (*callback methods*)
    - Ejemplo: ejbCreate, ejbPostCreate, ejbLoad, ejbStore, etc.
- Los contenedores administran dos tipos de *beans*:
  - *Entity Beans*: implementan los objetos de la aplicación
  - *Session Beans*: implementan secuencias de operaciones para un cliente
  - Se definen contratos específicos para cada uno de estos tipos (con algunas variaciones según el grado de intervención del contenedor)

20

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Entity Beans

- Propiedades:
  - Representan entidades persistentes (almacenadas en una base de datos) como objetos:
    - La persistencia puede ser administrada por el mismo *bean* (*bean managed persistence*) o delegada al contenedor (*container managed persistence*)
  - Compartidos por varios clientes:
    - Gestión de la concurrencia
  - Están involucrados en varias transacciones
  - Sobreviven a *shutdowns* o fallas de los servidores EJB
  - ¿Cómo se crean?
    - Creación explícita de una instancia
    - Insertando una entidad en una base de datos

21

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Session Beans

- Propiedades:
  - Representan una secuencia de operaciones para un cliente específico:
    - Proceso en memoria
    - Acceso a un DBMS
  - Creados y destruidos por un cliente
  - No persistentes
  - No sobreviven a *shutdowns* o a fallas
  - Dos maneras de administrar el estado:
    - **stateless**: no hay datos internos; puede ser compartido por varios clientes; no hay "pasivación" (cf. después)
    - **stateful**: preserva su estado a lo largo de una secuencia de llamadas a métodos (para un cliente); debe incluir primitivas de "pasivación" y activación (cf. después)

22

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Ciclo de vida

- Administrado por el contenedor (vía *Home Interface*):
  - Permite al cliente crear, destruir y localizar un *bean*
  - El *bean* debe implementar los métodos correspondientes (*callback*): ejbCreate, ejbPostCreate, etc.
- Gestión del estado de un *bean* por el contenedor:
  - Pasivación:
    - Guarda el estado del *bean* en un soporte de persistencia
    - Desactiva el *bean* (ya no puede ser llamado)
  - Activación:
    - Recupera el estado del *bean* de un soporte de persistencia
    - Reactiva al *bean* (puede volver a ser llamado)
  - El *bean* debe implementar los (*callback*) métodos ejbPassivate y ejbActivate

23

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Administración de persistencia

- Persistencia administrada por el contenedor (*container-managed persistence*):
  - El contenedor es responsable de guardar el estado del *bean*
  - Deben ser especificados los campos a ser guardados y el soporte de persistencia, de manera separada en un *deployment descriptor*
- Persistencia administrada por el *bean* (*bean-managed persistence*):
  - El *bean* es responsable de guardar el estado
  - Debe insertar explícitamente las operaciones que ejecutan la administración de la persistencia en funciones apropiadas *callback*
  - Menos adaptable que la administrada por el contenedor porque la administración de la persistencia está « cableada » en el código

24

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Deployment descriptor

- Función:
  - Especificación declarativa de algunas propiedades de un *bean*
    - Identificación, atributos transaccionales, campos persistentes, ambiente, administración por el contenedor o por el *bean*, roles para la seguridad
  - Usado por el contenedor para ejecutar una operación
- Forma:
  - Disponible a partir de EJB 1.1 y escrito en XML
  - Para cada propiedad, un *tag* específico es definido por la DTD
  - Ejemplo:

```
...
<persistence-type>Container</persistence-type>
<container-transaction>
        <method>
            <ejb-name>MyBean</ejb-name>
            <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
</container-transaction>
...
```

Persistencia administrada por el contenedor

Transacciones administradas por el contenedor

Ejecución transaccional para todos los métodos

25

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

26

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Plan

- ✓ Componentes:
  - Definición
  - Modelo genérico
- ✓ *Enterprise Java Beans* (EJB):
  - Esquema de ejecución
  - Roles y contratos
  - Ciclo de vida administrado
- Desarrollo de una aplicación con EJB
- Conclusión sobre EJB

27

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Desarrollo de una aplicación (1)

- Desarrollando un *Session Bean*:
  - Crear la interfaz *Home* (extiende ejb.EJBHome)
    - Métodos create, remove
  - Crear la interfaz *Remote* (extiende ejb.EJBObject)
    - Métodos propios de la aplicación
  - Implementación de los métodos de creación (*Create*, *PostCreate*)
  - Escribir la implementación de la interfaz:
    - Métodos específicos
    - Métodos *callback*:
      - setSessionContext, ejbActivate, ejBPassivate, etc.
      - Si el estado del *bean* es administrado por el contenedor (*container-managed state*): afterBegin, afterCompletion, beforeCompletion, etc.

28

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Desarrollo de una aplicación (2)

- Desarrollando un *Entity Bean*:
  - Crear la interfaz *Home* (extiende ejb.EJBHome)
    - Métodos create, remove
  - Crear la interfaz *Remote* (extiende ejb.EJBObject)
    - Métodos propios de la aplicación
  - Escribir un método "*primary key*" que será usada para localizar y acceder al *bean*
  - Escribir la implementación de los métodos de creación (Create, PostCreate)
  - Escribir la implementación de la interfaz:
    - Métodos específicos
    - Métodos *callback*:
      - setEntityContext, ejbActivate, ejBPassivate, ejbLoad, ejbStore, etc.
      - Para la persistencia administración por el contenedor todos estos métodos pueden estar vacíos

29

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Pasos

- Escribir el *deployment descriptor* :
  - Uno por *bean*
  - Define el comportamiento para las transacciones, la persistencia, la seguridad, comunicación con las bases de datos, ambiente (colocación de los servidores), etc.
- Configurar el servidor:
  - Compilar los programas de los *beans*
  - Generar las clases del contenedor (implementación de las interfaces *Home* y *Remote*) con la herramienta adecuada usando las clases de los *beans* y el *deployment descriptor*
- Desarrollar e inicializar el cliente:
  - El cliente obtiene la referencia de los *beans* a través del servicio de nombres
  - El cliente puede crear e iniciar sesiones o directamente llamar a los *Entity beans*

30





# Plan

- ✓ Componentes:
  - Definición
  - Modelo genérico
- ✓ *Enterprise Java Beans* (EJB):
  - Esquema de ejecución
  - Roles y contratos
  - Ciclo de vida administrado
- ✓ Desarrollo de una aplicación con EJB
- ■ Conclusión sobre EJB

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

# Conclusión sobre EJB

- Un modelo a componentes para la programación del servidor:
  - Ampliamente usado
  - Influye sobre el proceso de normalización (OMG)
- Beneficios:
  - Simplifica el desarrollo de aplicaciones complejas liberando al desarrollador de los aspectos que no son directamente relevantes para una aplicación:
    - Administración declarativa de transacciones
    - Administración de la persistencia
    - Administración de la seguridad
    - Administración de la distribución
  - Incrementa la independencia entre la plataforma y las aplicaciones:
    - Separación de los roles de los proveedores
    - *Openness*, competencia, mejora la calidad
  - Modelo extensible

33

UDLAP
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

34