Patrones de diseño

Sesión 3: Patrones Java EE para aplicaciones distribuídas

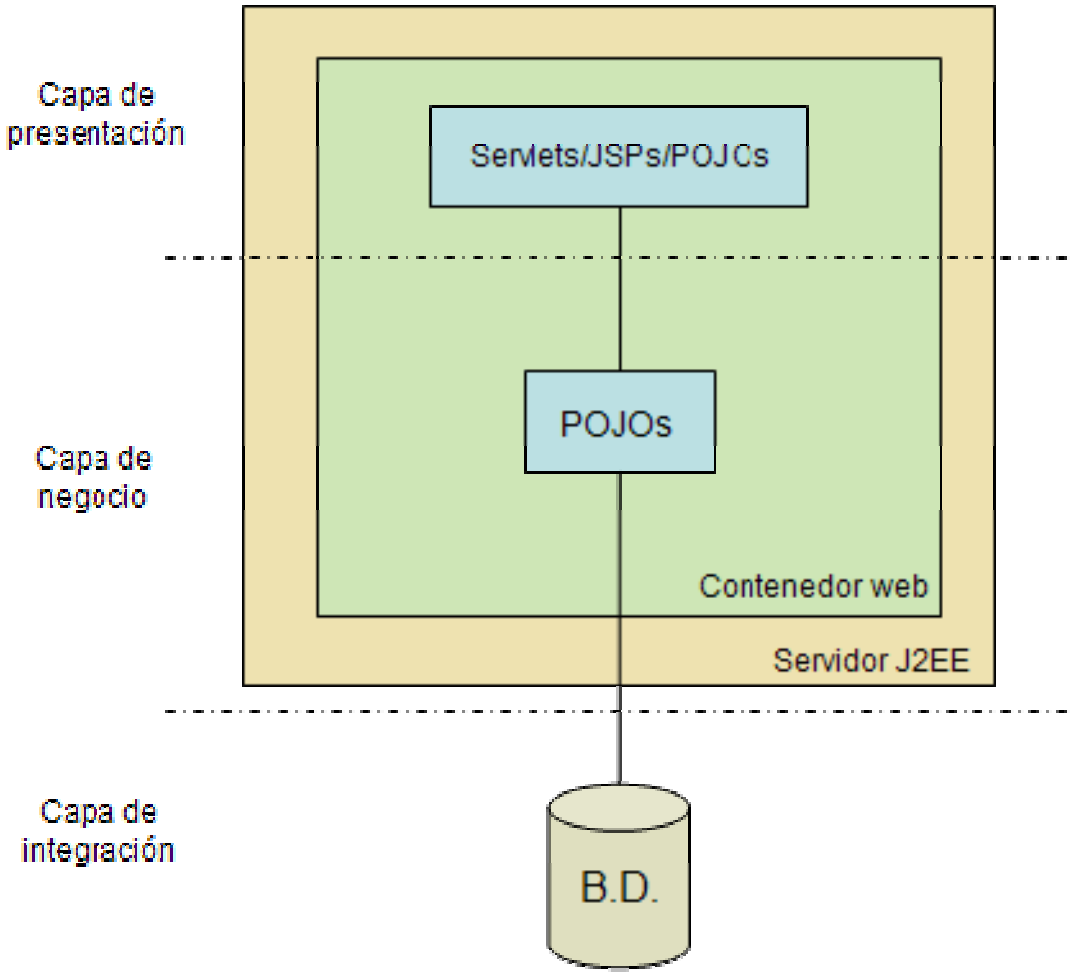
Indice

- Arquitecturas para aplicaciones JavaEE
  - Aplicaciones locales
  - Aplicaciones distribuídas: ¿por qué EJBs?
- Session Façade
- Service Locator
- Transfer Object
- Business delegate

© 2006-2007/ Depto. Ciencia de la Computación e IA

Aplicaciones locales

- El modelo que hemos seguido hasta ahora
- Sencillo, poco coste
- Limitado a clientes web
- Escalabilidad limitada



© 2006-2007/ Depto. Ciencia de la Computación e IA

Aplicaciones *enterprise*

- Requieren una serie de **servicios**:
  - Seguridad
  - Transaccionalidad
- Si son complejas, puede ser necesario hacerlas **distribuidas**
  - Cada capa en una máquina física (y/o lógica) diferente

© 2006-2007/ Depto. Ciencia de la Computación e IA

Introducción (muy) acelerada a los EJBs

- Objetos “similares” a los beans de Spring
- El contenedor EJB gestiona automáticamente el **ciclo de vida**
  - Sea *singleton* o pool de instancias
- Permiten **acceso remoto** de modo (casi) transparente
- Permiten gestionar de modo **declarativo**
  - La seguridad
  - La transaccionalidad

© 2006-2007/ Depto. Ciencia de la Computación e IA

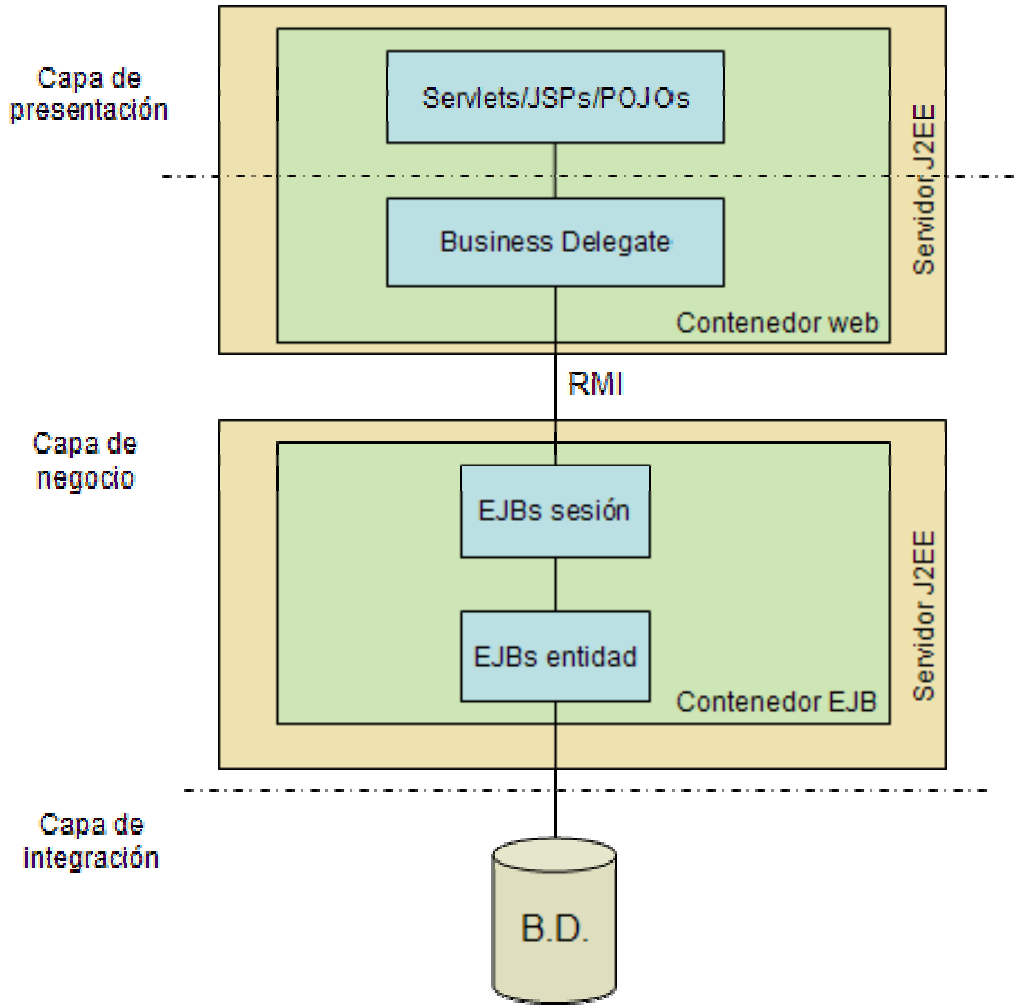
Tipos principales de EJBs

- **De sesión**: idóneos para la capa de negocio (xxxBO), por (recordemos)
  - Acceso remoto (casi) transparente
  - Transaccionalidad y seguridad declarativas
- **De entidad**: objetos del dominio, persistentes. Mismo papel que Hibernate, pero mucho peor implementados
- Existe un consenso más o menos general de que **los EJBs de sesión son muy útiles**, no tanto los de entidad, que han quedado muy desfasados (en la versión 2.x)

© 2006-2007/ Depto. Ciencia de la Computación e IA

Arquitectura distribuída con EJBs

- Ventajas:
  - Escalabilidad
  - Acceso remoto
  - Servicios declarativos
- Problemas
  - Coste de comunicaciones
  - Complejidad de pruebas, despliegues,…



© 2006-2007/ Depto. Ciencia de la Computación e IA

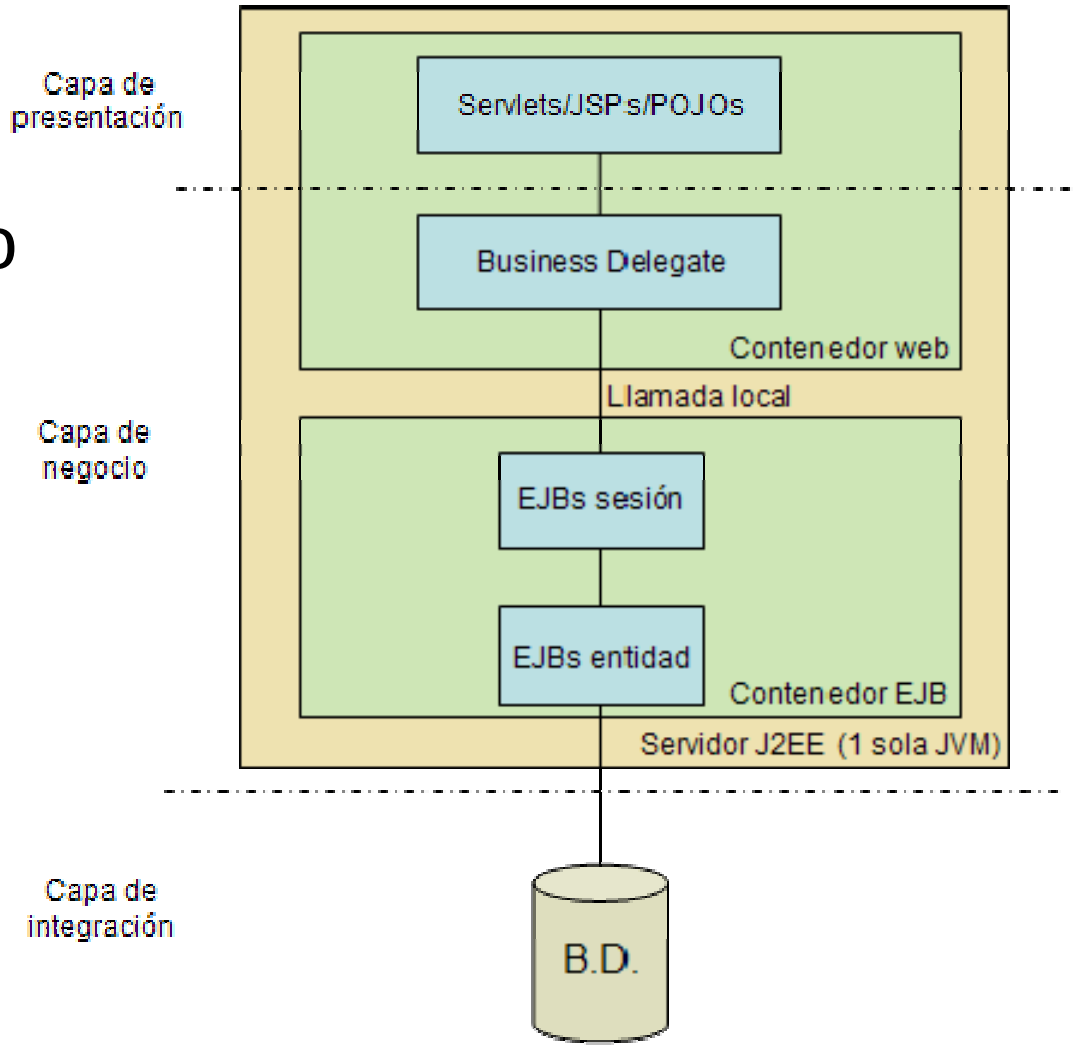
Problemas de las aplicaciones J2EE distribuídas

- Hace unos años se tomaba el modelo anterior como “prototipo”
  - Promovido desde Sun en sus “buenas prácticas” (el caso *Pet Store*)
  - Parecía “obligatorio” (o como mínimo *cool*) que todas las aplicaciones usaran EJBs y fueran distribuídas
- Problemas:
  - Inmadurez de la tecnología EJB
  - Por muy sencillo que lo pusieran las herramientas, las aplicaciones distribuídas son **complejas**

© 2006-2007/ Depto. Ciencia de la Computación e IA

Arquitectura “mixta”

- Aprovechar las ventajas de los EJBs intentando minimizar los costes (comunicación remota)
- Usan **EJBs locales**



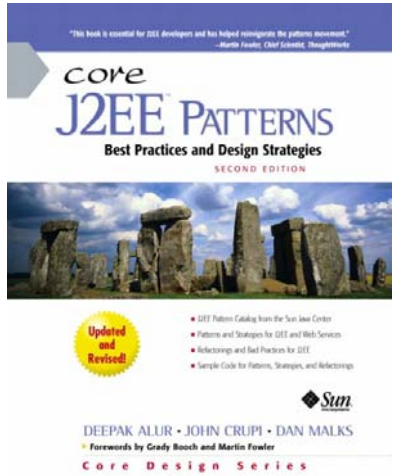
© 2006-2007/ Depto. Ciencia de la Computación e IA

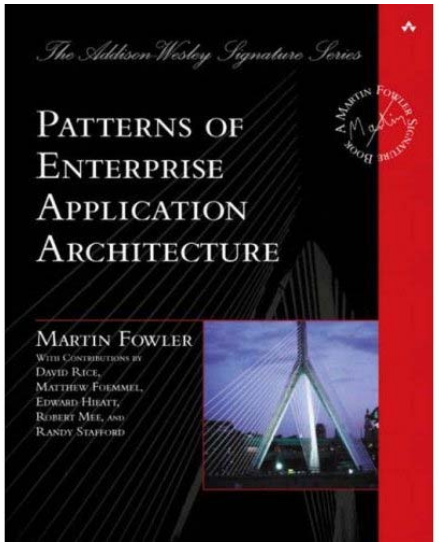
Patrones para aplicaciones distribuídas

- Buscan sobre todo reducir el coste de las **comunicaciones remotas**
- Algunos son aplicables a cualquier plataforma, otros son específicos de algún API de JavaEE
- Fuentes:
  - *Core J2EE Patterns*

    Alur, Crupi & Malks (www.corej2eepatterns.com)
  - *Patterns of Enterprise Application Architecture*

    Martin Fowler (www.martinfowler.com)

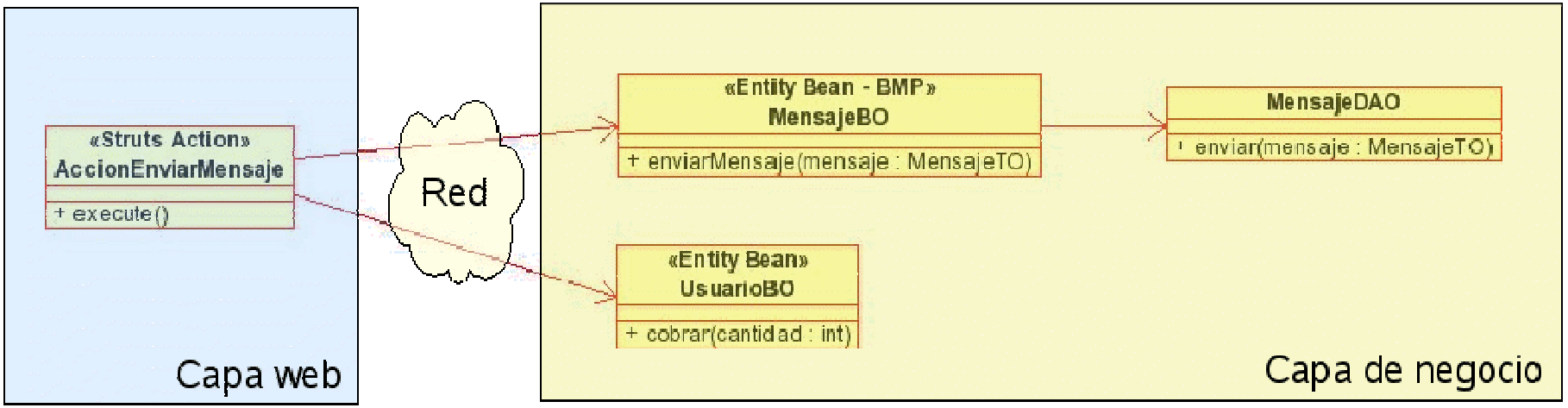




© 2006-2007/ Depto. Ciencia de la Computación e IA

Comunicación presentación-negocio

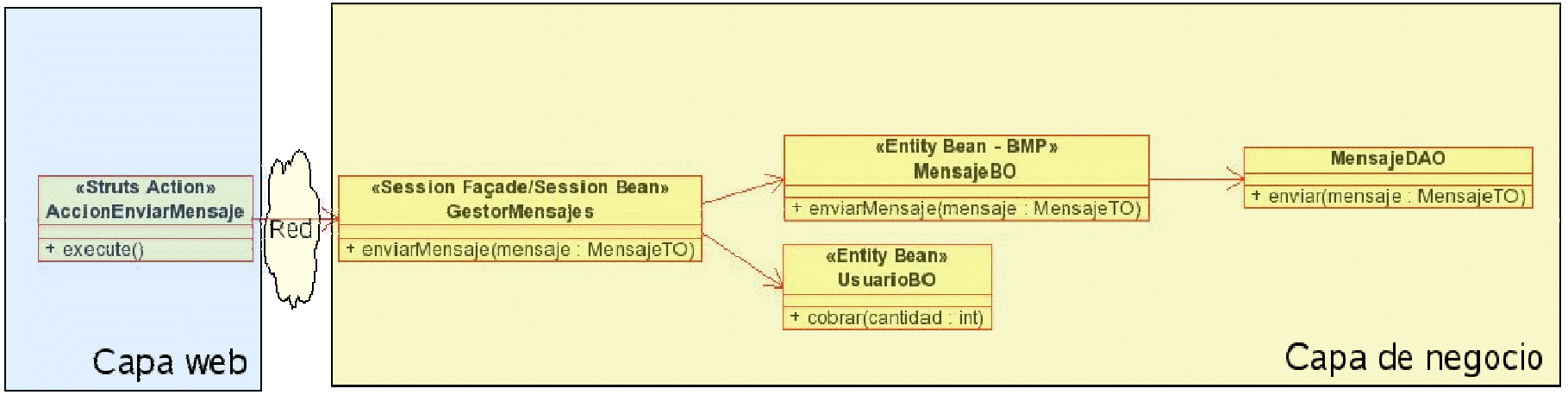
- Ejemplo: enviar un mensaje (envío + cobro coste)
- Desde el punto de vista:
  - **Lógico**: la capa de presentación no debe implementar lógica de negocio
  - **Coste**: requiere 2 llamadas remotas
  - **Transaccionalidad** declarativa: independiente por cada EJB



© 2006-2007/ Depto. Ciencia de la Computación e IA

Session Façade

- EJB de sesión local a la capa de negocio
  - EJB porque así podemos aprovechar servicios declarativos
  - En realidad es un caso especial del Façade del GoF
- Soluciona:
  - Lógica: queda en negocio
  - Coste: 1 sola llamada remota para cualquier operación
  - Transaccionalidad: 1 solo EJB: 1 sola transacción



© 2006-2007/ Depto. Ciencia de la Computación e IA

Transfer Object

- Igual al de las aplicaciones locales, pero sin "culpabilidad" para el programador
  - Por muy elegante que sea, interactuar a "grano fino" con un objeto de negocio remoto es muy costoso

    getNombre(), getApellidos()…vs. getUsuarioTO()
  - Pero muchos recomiendan limitar los TOs a la comunicación de capas en pro de la "ortodoxia OO"

- Por tanto, los TOs deben ser **serializables**
  - Ya lo hacíamos aunque hasta ahora en realidad no era necesario. Esto nos protege ante los cambios

© 2006-2007/ Depto. Ciencia de la Computación e IA

Service Locator

- Instanciar EJBs en 2.X requiere JNDI
  - Nuestro código se llenará por todos lados de bloques como este

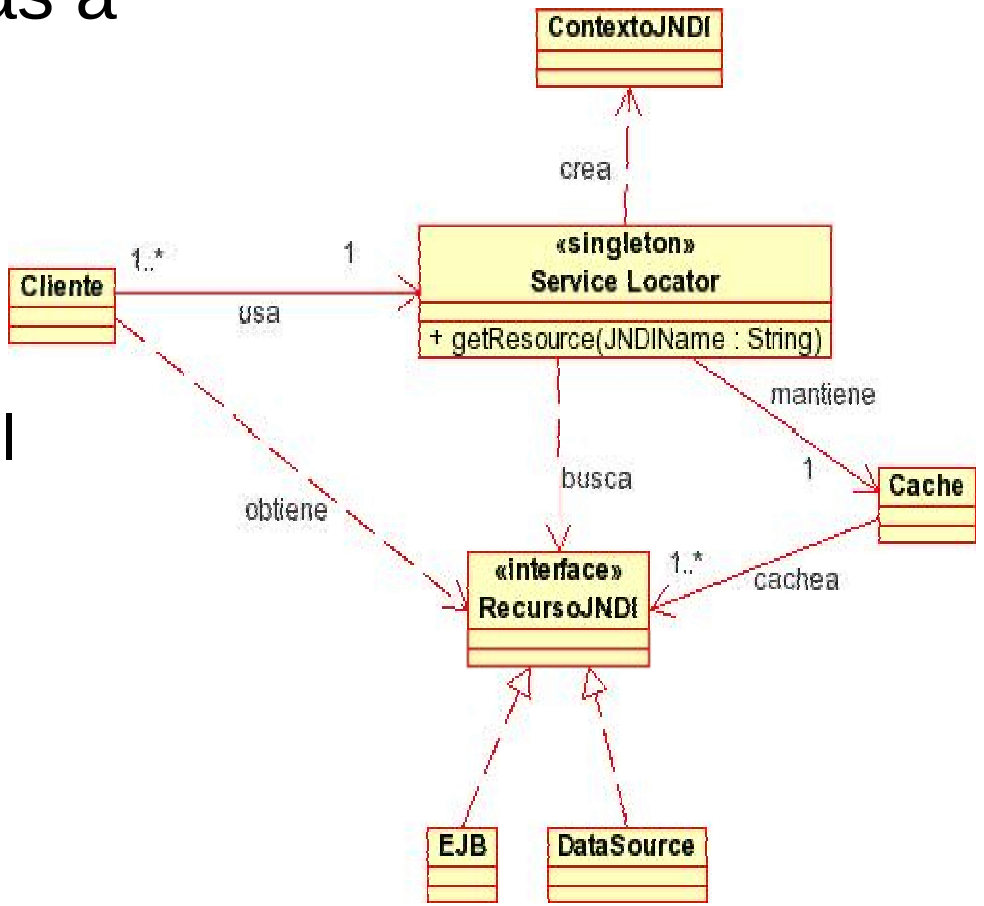
```
Context ctx = new InitialContext();
Object o = ctx.lookup("java:comp/env/UserBO");
UserBOHome home = (UserBOHome) PortableRemoteObject.narrow(o, UserBOHome.class);
UserBO ubo = home.create();
```

- ***Aclaración****: todo esto es complicado porque no se hace como new(), en realidad el que instancia es el contenedor, como en Spring*

© 2006-2007/ Depto. Ciencia de la Computación e IA

Service Locator

- Centraliza las llamadas a JNDI (una especie de Factory)
- Puede evitar trabajo superfluo
  - Manteniendo creado el contexto
  - Guardando una cache



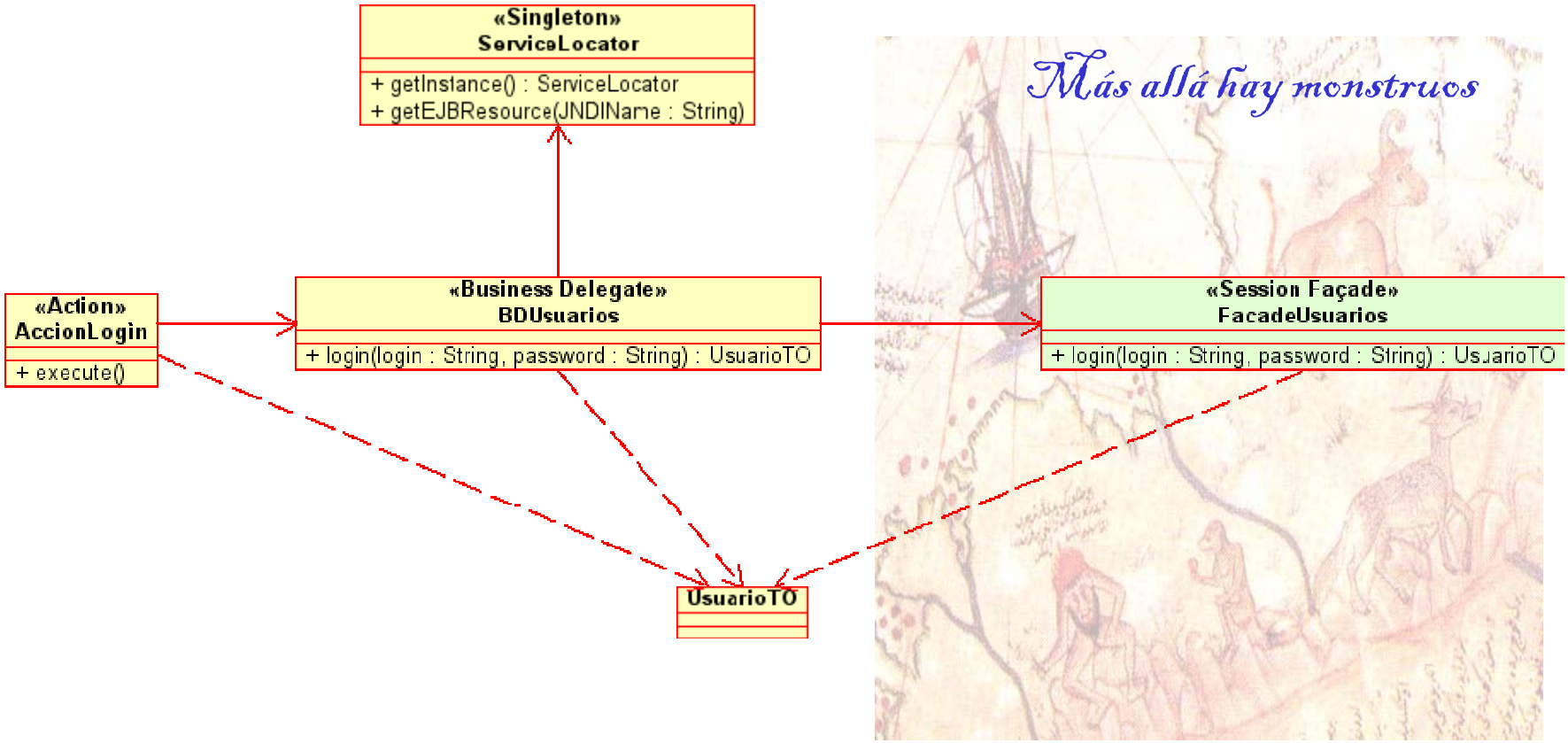
© 2006-2007/ Depto. Ciencia de la Computación e IA

Business Delegate

- Una capa intermedia entre presentación y negocio, aislando las dos
- Características
  - Físicamente estará en la misma máquina que la presentación
  - Aquí se coloca el código dependiente del API de EJB
- Con ello conseguimos
  - La capa de presentación hace llamadas locales, esté o no distribuída la aplicación
  - La capa de presentación no sabe si hay o no EJBs

© 2006-2007/ Depto. Ciencia de la Computación e IA

Ejemplo: BD + Service Locator + Façade



© 2006-2007/ Depto. Ciencia de la Computación e IA

Código simplificado: acción + BD

```
public class AccionLogin extends Action {
   public ActionForward execute(ActionMapping mapping, ActionForm form,
      HttpServletRequest req, HttpServletResponse resp){
            //Aquí habría que poner un factory, mejor
            BDUsuarios bdu = new BDUsuarios();
            LoginForm lf = (LoginForm) form;
            //no sabe que detrás hay EJBs remotos ( =monstruos) :)
            UsuarioTO u = bdu.login(lf.getLogin(), lf.getPassword());
            …
      }
}
```

```
public class BDUsuarios  {
   public void UsuarioTO login(String login, String password) {
      ServiceLocator sl = sl.getInstance();
      //conoce la implementación de la capa de negocio
      FacadeUsuariosHome h = sl.getEJBResource("java:comp/env/UserFacade");
      FacadeUsuarios fu = h.create();
      return fu.login(login, password);
   }
}
```


© 2006-2007/ Depto. Ciencia de la Computación e IA

Código simplificado: Service Locator

```
public class ServiceLocator  {
   private static ServiceLocator unico = null;
   private Context ctx;

   private ServiceLocator() {
      ctx = new InitialContext();
   }

   public ServiceLocator getInstance() {
      if (unico==null)
            unico = new ServiceLocator();
      return unico;
   }
   public EJBHome getEJBResource(String JNDIName) {
            return (EJBHome) contexto.lookup(JNDIName);
   }
}
```


© 2006-2007/ Depto. Ciencia de la Computación e IA

¿Preguntas...?