

# ¿“To EJB” or not “To EJB”?

Marzo de 2005

@author: Jorge Rodríguez

EJB es considerado por la mayoría el núcleo del J2EE, y por mucho tiempo la solución a todos los requerimientos de aplicaciones empresariales. Fue centro de atención de muchas empresas vendedoras de servidores de aplicación por algunos años, pero hoy el mercado está consolidado por lo que solo quedan algunas empresas líderes.

En el año 1998 aparece la especificación EJB 1.0 en medio de la moda del dotcom, así las empresas del sector no limitaban los gastos en infraestructura, sin importar muchas veces si la inversión significaba beneficios. Era una época de un clima económico benigno, que ayudó al surgimiento de tecnologías como el J2EE, costos como el hardware, las licencias, e incluso el entrenamiento del personal se contaban dentro de los presupuestos.

Por otra parte, CORBA era la única alternativa abierta frente a COM/DCOM de M\$<sup>1</sup>, pero CORBA es una tecnología compleja y que además no abstrae al desarrollador de tener que lidiar con APIs de comunicación remota, así, EJB aparece como una solución ideal, simple y abierta.

## Mito vs Realidad

EJB fue creado entonces para facilitar nuestra vida, el comité encargado del EJB 1.0 nos promete entre otras cosas que:

- “Sería muy fácil escribir aplicaciones usando los Enterprise Java Beans, pues los contenedores de EJB nos proveen de servicios que nos permitirían concentrarnos en el negocio de la aplicación”.
- “Nuestras aplicaciones se escribirían un sola una vez, y luego se instalarían en cualquier plataforma sin tener que modificar el código fuente ni recompilar nuestros EJBS, o sea aplicaciones 100% portables”.

Pero todas estas promesas relacionadas al uso de EJB no se cumplen muy bien, la realidad práctica ha mostrado otros resultados con respecto al uso de estos componentes en nuestras aplicaciones J2EE. Desarrollar una aplicación usando EJB es costoso tanto del punto de vista económico como desde el tecnológico.

No abundan desarrolladores con buen grado de conocimientos en el área EJB, y de encontrarlos es evidente que se tenga que pagar un precio caro por contratarlos en un proyecto. Por otro lado el esquema para el equipo de desarrollo que propone la especificación donde hay roles tales como Proveedor de Beans (Bean Provider), Ensamblador de Aplicación (Application Assembler), Instalador de EJB (EJB Deployers) y Administrador de Sistemas (System Administrator) solo existe en la mente del comité encargado de mantener la especificación.

Del lado tecnológico, lidiar con componentes EJB no es una tarea trivial, a pesar que existen herramientas que ayudan en el trabajo de desarrollo, no es transparente el ciclo de crear, deployar y probar un EJB. Mesmo en ambientes de desarrollo sofisticados como el Workshop de bea o el WSAD de IBM, casi nunca escapamos al infierno que significa entender y muchas veces modificar descriptores de instalación estándares como los `ejb-jar.xml`, o propietarios como `weblogic-ejb-jar.xml`.

En fin, el uso de EJBS significa altos costos en los proyectos, y bajas de la productividad en el desarrollo, y como consecuencia demora y falla en el cumplimiento de plazos de entrega establecidos en los proyectos, pero hay más.

---

<sup>1</sup> Micro\$oft.

## **Sobre los Stateless Session Beans**

Los Stateless Session Beans son muy populares entre arquitectos e ingenieros J2EE, y su fama no está mal infundada, ya que en un final proveen a la aplicación servicios necesarios como el manejo de transacciones, concurrencia, acceso remoto, soporte a clusters, pool de objetos de negocio, seguridad declarativa y el gerenciamiento de los objetos de negocio, abstrayéndonos de su implementación.

No obstante, como veremos más adelante, es posible contar con estos servicios sin depender de un contenedor EJB.

## **Modelo de Componentes y Modelo de Objetos**

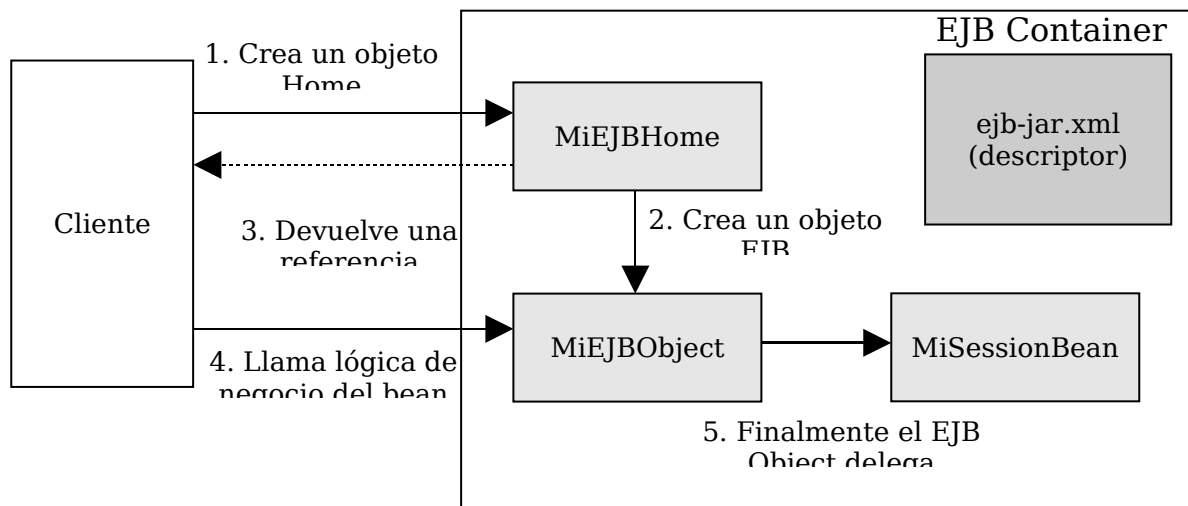
EJB introduce un modelo de componentes complejo, que incumple la promesa de escribir aplicaciones fácilmente, pues además de que tenemos que lidiar con todas las implementaciones de clases e interfaces de la API (ver figura 1), a nivel de código debemos hacer "lookup" de todos los EJB, y a pesar de hacer uso de patrones de diseño como el ServicesLocator, o el BusinessDelegate para minimizar el dolor, estos introducen dependencia de JNDI y las APIs EJB en nuestro código que finalmente queda amarrado al contenedor.

Por otra parte cuando diseñamos teniendo en cuenta que usaremos el modelo EJB, particularmente entity beans estamos limitando el diseño de nuestro modelos de objetos, personalmente considero a EJB como una tecnología invasiva en la etapa de diseño OO y que va en contra de los conceptos de ADOO, imaginen a la clase CuentaCorriente, que es evidentemente una clase de negocios teniendo que implementar la interface EntityBean, con esto estamos marcando la clase como persistente, ¿porque habría de saber CuentaCorriente que va a ser persistente?. El uso de entity beans introduce lo que Martin Fowler<sup>2</sup> describe como modelos anémicos, donde un objeto no posee comportamiento, solo estado.

---

<sup>2</sup> <http://www.martinfowler.com>

Fig1. Fragmento del modelo EJB, los objetos de color gris son implementaciones que debemos proveer al container.



## Donde están los vendedores de EJB?

Otra especulación de la que alardea el modelo EJB, es la de poder comprar componentes EJB hechos por terceros y ensamblarlos para construir una aplicación de forma ágil (recuerden la definición de roles descrita en párrafos anteriores). Todo parece indicar que la mala portabilidad de los EJB ligados muchas veces con características propietarias de los servidores de aplicación son causa de que este mercado nunca tuviera éxito en comparación con el mercado de controles ActiveX o Java Beans<sup>3</sup>(en sus versiones de tag libs o incluso librerías .jar para diversos fines).

## Java ya no es el mismo

La especificación EJB 1.0 aparece meses antes que el J2SE saliera en su versión 1.2 con grandes mejoras como el framework Collections, o la incorporación de Swing al JRE<sup>4</sup>, así mismo continua el desarrollo del J2SE llegando las versiones 1.3, 1.4 y hace muy poco la 5.0, sin embargo EJB sufre pocos cambios, casi siempre, soluciones a problemas de performance o escalabilidad como la inclusión de las interfaces locales en la especificación 2.0 o los Web Services en la 2.1.

<sup>3</sup> Objeto Java con un constructor vacío (default) e implementaciones de métodos get/set para cada uno de sus atributos private, o protected.

<sup>4</sup> En versiones anteriores Swing eran librerías aparte.

La versión 1.3 del J2SE introdujo características nuevas como los Proxis Dinámicos, que permiten implementar cualquier interface en runtime, lo cual fue incluso usado por las implementaciones EJB 2.0 que permiten la generación de las clases (stubs) ahora en la fase de deployment de la aplicación. Por otra parte esta nueva funcionalidad acompañada de otras como la mejora en la performance del paquete reflection del nuevo JDK dio lugar al nacimiento de una nueva era en la que aparecieron librerías como las CGLIB (Code Generation Library), librerías usadas por Hibernate, iBatis, o Spring y que permiten generar código en runtime que intercepte llamadas a métodos de cualquier clase java.

Por su parte el J2SE 5.0 incluye el uso de meta-datos a nivel de código fuente (en los .java), conocido como “annotations”<sup>5</sup> que permiten eliminar el uso de archivos descriptores como los ejb-jar.xml.

Otro modelo de programación ha surgido con esta ola de nuevas tecnologías y es la AOP<sup>6</sup> (Aspect Oriented Programming), que basado en la intercepción, permite ejecutar código antes y después de la llamada de cualquier método de cualquier clase, sin tener esta que extender de clase o interface alguna. A diferencia del modelo EJB, AOP es un complemento a la programación orientada a objetos, lo cual define en muchos aspectos su éxito y uso actual. No obstante el uso de AOP incluye una curva de aprendizaje en los desarrolladores, pero hoy es posible el uso de frameworks que nos abstraen de su uso directo, como es el caso de Spring (que hace extensivo uso de AOP, por ejemplo para el manejo de transacciones en métodos de negocios).

## Antipatrones

La aparición en primera instancia de la especificación EJB y luego de las implementaciones reales, obligaron a los vendedores de containers EJB a cumplir los estándares, olvidando la experiencia práctica, lo cual provocó la aparición de ciertos “Patrones de Diseño”, creados en su mayoría para resolver problemas del modelo EJB como la alta dependencia de llamadas JNDI (ServiceLocator), la baja performance provocada por el exceso de llamadas remotas (BusinessDelegate), o el transporte de datos entre capas remotas (DTO<sup>7</sup>), estos patrones además introducen nuevas curvas de aprendizaje, no basta con leerse las casi 400 páginas de la especificación, debemos aprender también como usar los patrones. Sin embargo muchos de estos forman parte de un conjunto de malas prácticas de programación que se ponen de moda entre desarrolladores, un ejemplo clásico lo constituye el patrón DTO que introduce duplicación de código, y como consecuencia de esfuerzo a la hora de la manutención, imaginen de nuevo a CuentaCorriente, pero además ahora CuentaCorrienteDTO y que ahora además implementa la interface Serializable.

## Escalabilidad

EJB por su naturaleza remota permite la distribución de componentes dentro de las aplicaciones (separación física usando interfaces remotas) y por mucho tiempo se pensó que escalaban mejor que las aplicaciones web co-locadas (separación lógica), pero la práctica ha demostrado lo contrario, tanto Entity Beans como los Statefull Session Beans no escalan bien en ambientes clusterizados, sin embargo los Stateless Session Beans escalan bien, pero es evidentemente por su naturaleza sin estado. De cualquier forma es preferible clusterizar aplicaciones web usando balanceadores de carga por hardware como el Cisco Load Balancer o por software como servidores web. Por otra parte es mejor manipular el estado de una aplicación web a nivel de objetos HttpSession y no Statefull Session Beans, pues los primeros escalan mejor.

## Metodologías Ágiles

---

<sup>5</sup> XDoclet es un producto open source que permite hacer esto hoy en día y fue fuente de inspiración para la aparición de los annotations en el J2SE 5.0

<sup>6</sup> Se encarga de los aspectos ortogonales de la aplicación como el manejo de transacciones, la seguridad, el login, etc. Algunas implementaciones pueden ser encontradas en:

<http://aspectwerkz.codehaus.org/>

<http://eclipse.org/aspectj/>

<http://aopalliance.sourceforge.net/>

<sup>7</sup> Data Transfer Object

En el dinamismo económico que vive el mundo hoy, incluso los procesos de desarrollo de software han tenido que adaptarse, naciendo nuevas disciplinas como la eXtreme Programming o la TDD (Test Driven Development, donde la filosofía es incluso escribir las pruebas antes de la implementación), basada esta última en el uso pruebas unitarias<sup>8</sup>. En estas nuevas disciplinas donde el ciclo de desarrollo de una aplicación se hace en iteraciones cortas, juega un papel muy importante el poder realizar pruebas rápidas del código implementado. Así, y debido a la dependencia de los EJBs con el servidor, se hace muy difícil la práctica de estos procesos de desarrollo ágiles, pues el ciclo de implementar, deployar y probar puede llegar a durar mucho tiempo.

## Motivados por EJB 3.0

La publicación de la especificación EJB 3.0 (es notable el salto del número de 2.1 a 3.0) a mediados del 2003 y que aún está en revisión (puede ser bajada desde <http://java.sun.com/products/ejb/docs.html>), hace pensar que la Sun sintió la necesidad de dar aires frescos a su tecnología (ahora con el lema “facilitar el uso de EJBs, pero desde el punto de vista de los desarrolladores”), dando soporte al uso de meta-datos en los EJB<sup>9</sup> mediante las nuevas capacidades del J2SE 5.0 (recuerden los “annotations”). A modo de ejemplo, cuando este lista dicha especificación seremos capaces de escribir código como el de la figura 2

Figura 2 – Uso de meta-datos según especificación EJB 3.0

```
@Remote
@Stateless
public class MiNuevoBean implements InterfaceDeNegocios {

    public void metodoDeNegocio() {
        // implementación ...
    }
    ...
}
```

Y de esta forma definir un EJB Stateless Session Bean como un objeto java plano que implementa su interface de negocios sin tener que implementar la antigua interface SessionBean de la API EJB.

No obstante y a pesar que la especificación J2EE 1.4<sup>10</sup> salió en Noviembre del 2003 , existen pocos servidores J2EE 1.4 Certified<sup>11</sup>, por tanto es de esperar que pasen algunos años antes de aparecer implementaciones de EJB 3.0.

## Cuando realmente usarlos

EJB tiene aun su lugar en el mundo java, y es evidente que con la especificación 3.0 va a imponerse nuevamente como la tecnología *de facto* en la construcción de aplicaciones J2EE. Decidir hoy de si debemos usar o no EJB no es trivial, pues depende de los requerimientos tanto funcionales como no funcionales de la aplicación, sin embargo hay escenarios donde es apropiado usarlos, por ejemplo cuando:

- La aplicación será 100% middleware y distribuida. No posee una interfaz web definida, sino que su función será la de brindar servicios a otras aplicaciones. Aquí es evidente que el uso de RMI es una tecnología fácil de usar para implementar esto.
- Estemos implementando servicios dentro de un ambiente SOA. Nuevamente la mejor solución puede ser el uso de RMI o Web Services a través de Stateless Session Beans<sup>12</sup>.
- Es un hecho que la aplicación dará servicios a múltiples clientes (como web y aplicación desktop). En la práctica esto es algo raro.

<sup>8</sup> En java a través de frameworks como JUnit, HttpUnit, etc

<sup>9</sup> No obstante continuará el soporte de EJB 2.1 y 2.0

<sup>10</sup> Incluye entre otras la especificación EJB 2.1

<sup>11</sup> ver <http://java.sun.com/j2ee/compatibility.html>

<sup>12</sup> Según especificación 2.1 de EJB podemos convertir nuestros Stateless Session Beans en WebServices.

## Soluciones alternativas hoy

¿Pero si **no** es conveniente el uso de EJBs, entonces nuestra aplicación deja de ser J2EE, y además no podremos gozar de los servicios que el servidor de aplicaciones nos ofrece, ¿cierto?, NOOO, errado. J2EE != EJB, “J2EE es mucho más que EJB”<sup>13</sup> y más aún, “EJB es mucho menos que Java”. A continuación algunos productos alternativos que un arquitecto pragmático puede usar para sustituir el uso de EJB (session beans y entity beans) en aplicaciones J2EE.

- Spring Framework: es un contenedor de aplicaciones que usa Inversión de Control (IoC), o también conocido como Inyección de dependencias<sup>14</sup> como concepto principal, y que junto a su implementación AOP ofrecen soporte a servicios mediante el uso de objetos planos<sup>15</sup>. Algunos de los servicios son:
  - Manejo de transacciones tanto declarativos como programados, o incluso ambos dentro de una misma clase (lo cual es imposible mediante el uso EJB, ya que o usas CMT o usas BMT).
  - Gerenciamiento de los objetos de negocio. Mediante estilo JavaBean set/get.
  - Seguridad a través de interceptores. Nuevamente gracias a AOP.
  - Acceso remoto. Podemos exponer servicios en objetos planos a través del soporte a varias tecnologías como RMI, WebServices, JMS, o protocolos propietarios como Hessian o Burlap, todo esto a través de proxies dinámicos.
- JDO: Estándar JSR 12<sup>16</sup> de Sun, cuyas implementaciones como KodoJDO ofrecen buena alternativa como sustitución a Entity Beans.
- Hibernate: Framework ORM<sup>17</sup> open source y muy popular, que provee una forma sencilla, y fácil de usar como solución de persistencia. Usa internamente CGLIB, lo cual permite que cualquier objeto de nuestro modelo pueda persistirse transparentemente sin extender ninguna clase o interface, favoreciendo así el ADOO.
- iBatis: Producto open source que ofrece una capa de abstracción entre los objetos de negocio y el uso de JDBC para acceder a modelos relacionales.
- PicoContainer y HiveMind: Contenedores que al igual que Spring usan como núcleo el patrón IoC y pueden combinarse con implementaciones AOP.

Estos productos pueden formar parte de una aplicación J2EE y de esta forma proveer servicios que EJBs ofrecen de forma más sencilla y que nos permiten continuar pensando e implementando orientado a objetos en nuestros modelos. No obstante los Stateless session beans, pueden ser usados como fachada a los objetos de negocio para dar soporte de acceso remoto a los mismos (este es el mayor valor que tienen los sessions beans).

En el caso de la persistencia de nuestras aplicaciones, una buena práctica es colocar todo el acceso a los datos detrás de objetos DAO<sup>18</sup>, permitiendo independizar nuestro modelo de negocio de la tecnología usada para persistir los objetos, y pudiendo así “cambiar”<sup>19</sup> dicha tecnología sin alterar la lógica de negocio.

En próximos artículos veremos como integrar estos productos para construir aplicaciones J2EE, que sean escalables y no dependan de un contenedor EJB.

## Concluyendo

EJB provee servicios de mucho valor. Los Stateless session beans nos proveen de excelente solución a la hora de distribuir servicios en nuestras aplicaciones<sup>20</sup>, además, escalan bien en ambientes clusterizables, y son una forma fácil de separar interface de implementación en objetos de negocio. Por otra parte, a través del uso de los

---

<sup>13</sup> J2EE es un conjunto de especificaciones bien definidas entre las que se encuentran Servlets, JSP, JDBC, JMS, RMI, JTA, JCA, entre otras.

<sup>14</sup> Dependency Injection (<http://www.martinfowler.com/articles/injection.html>)

<sup>15</sup> Conocidos en la literatura como Plain Old Java Object (POJO)

<sup>16</sup> Abandonado en un principio por la aparición de los EJB y retomado hace algunos años.

<sup>17</sup> Object Relational Mapping

<sup>18</sup> Data Access Object que evidentemente implementan interfaces DAO.

<sup>19</sup> Casi siempre el diseño de una aplicación es influenciado por la tecnología a usar.

<sup>20</sup> Ya sea a través de RMI o incluso Web Services para el caso de EJB 2.1

Message-Driven Beans, podemos dar soporte a mensajería asíncrona a nuestras aplicaciones.

Sin embargo, EJB no cumplió muy bien algunas de las promesas con las que la especificación alardeo en sus días de gloria, entre las que recitamos:

“Hacer fácil el escribir aplicaciones”. Como ya citamos EJB elimina cierta complejidad, introduce un modelo de programación complejo e invasivo.

“Desarrollar una vez, e instalar en cualquier plataforma, sin cambios en el código ni recompilación”. Mismo cumpliendo con la especificación, una aplicación EJB no hace el código 100% portable como puede hacerse con Servlets y/o JSP, en la vida práctica migrar una aplicación de un servidor de aplicaciones a otro incluye siempre algún tipo de re-escritura y recompilación del código fuente.

Por otra parte estamos en una era de soluciones pragmáticas que van más allá del EJB, y que si bien no cumplen estándares de mercado están determinadas al éxito por su filosofía “basada en la experiencia”, como es el caso de los productos: Hibernate, JDO, implementaciones AOP , o el uso de contenedores de Inversión de Control como PicoContainer o Spring. Todos estos productos pueden ser usados para construir de forma ágil aplicaciones J2EE sin que dejen de ser escalables y fáciles de dar manutención.

Por tanto y sin concuerdan conmigo en todo lo antes expuesto, les pido que el día 0 del próximo proyecto J2EE, arquitectos e ingenieros y con el documento de requerimientos en mano, se reúnan como buenos samaritanos, compartan un café y coloquen en el pizarrón la pregunta “¿Es factible el uso de EJB en este proyecto...?”.

## Recursos

J2EE – <http://java.sun.com/j2ee>

WebLogic Workshop 8.1 - <http://dev2dev.bea.com/products/wlworkshop81/index.jsp>

WebSphere Studio Application Developer -

<http://www-306.ibm.com/software/awdtools/studioappdev/>

CGLIB - <http://cglib.sourceforge.net/>

Hibernate – <http://hibernate.org/>

JDO - <http://java.sun.com/products/jdo/>

iBatis – <http://ibatis.com/>

Spring – <http://www.springframework.org/>

PicoContainer – <http://www.picocontainer.org/>

HiveMind – <http://jakarta.apache.org/hivemind/>

J2SE 5.0 – <http://java.sun.com/j2se/1.5.0/>

eXtreme Programming - <http://www.extremeprogramming.org/>

Test Driven Development – <http://www.testdriven.com/>

JUnit – <http://www.junit.org/>

Especificación de EJB 3.0 - <http://java.sun.com/products/ejb/docs.html>

Acerca del autor:

Sun Certified Programmer for the Java 2 Platform 1.4 y Sun Certified Web Component Developer for the Java 2 Platform Enterprise Edition, además cuenta con una certificación en WebLogic Workshop 8.1, trabaja como Arquitecto J2EE para Interplanet y Xperience Consulting Services y puede ser contactado en [jrodriguez@interplanet.cl](mailto:jrodriguez@interplanet.cl).