

# Mi Arquitectura preferida

Diciembre 2005

@autor: Jorge Rodriguez <jorge.rodriguez@interplanet.cl>

Este artículo se enfoca en mostrar cual, de las diversas variantes de arquitecturas J2EE que se pueden usar, es la que prefiero a la hora de desarrollar aplicaciones web, así como el conjunto de tecnologías que, pueden combinarse para dar vida a una implementación de dicha arquitectura.

## Capas:

En arquitecturas web es clásico hablar de 3 capas principales:

- **Capa de negocio o servicios.** Por mucho la capa más importante.
- **Capa de presentación o web.** Donde se centra la lógica de recopilación y despliegue de datos del modelo de negocio.
- **Capa de acceso a datos.** Donde se persiste el estado de la aplicación.

Adicionalmente, aparecen casi siempre otras capas como consecuencia del diseño de la solución, estas son:

- **Capa de integración con sistemas legados.** Capa de abstracción para acceder a lógica legada.
- **Capa remota de acceso a los servicios.** Soporte de acceso a la lógica de negocios para clientes remotos.

## La capa de Negocios o Servicios:

Esta es la principal de las capas, pues significa la razón de ser de la aplicación, y se apoya en las demás capas para exponer el negocio a clientes de naturaleza web (para el caso de browsers) o incluso remotos (para el caso de otras aplicaciones escritas en java u otro lenguaje de proposito general). Normalmente esta capa consiste en múltiples interfaces que representan contratos bien definidos.

Por tanto, una capa de negocios bien diseñada debe:

- **Ser simple y centrada en el negocio<sup>1</sup>.** Contener solo las operaciones que el negocio exige, sin preocuparnos por aspectos ortogonales como el soporte remoto a la lógica o incluso el manejo de transacciones<sup>2</sup>.
- **Contener la lógica definida en interfaces<sup>3</sup>, no en clases.** Las operaciones deben ser definidas en interfaces, desacopando el uso del servicio de su implementación, y permitiendo modelos basados en prototipos.
- **Ser totalmente orientada a objetos.** Usar los paradigmas de la OO, pero básicamente no exigir mucho de esta, por ejemplo no obligar a extender de clases propietarias o de terceros. Mientras menos exijamos de nuestros modelos de objetos, menos dependiente de cualquier plataforma serán, y por tanto mas portables y extensible se harán.
- **Facil de escribir y probar.** Su implementación debe ser facil de escribir y de probar, no debe depender de productos que obliguen a la instalación y prueba de ambientes complejos,

---

1 Aquí voto por el uso de Interfaces Humanas, ver <http://www.martinfowler.com/bliki/HumaneInterface.html>

2 Aquí se da pie a toda una discusión programísticamente filosófica de donde, y como manejar las transacciones, lo dejamos para tema de otras tecnografías.

3 por ejemplo, de la palabra reservada de java **interface**

debe ser posible escribir pruebas unitarias que solo dependan de un ambiente básico<sup>4</sup> para ejecutarlas.

- **Ser Stateless.** En la medida de lo posible no manejar estado, así por definición esta capa queda escalable vertical y horizontalmente<sup>5</sup>.

## Exponer objetos de negocio en la red

Habiendo conseguido un buen diseño de la capa de negocio, debe ser fácil poder enviar por la red objetos del modelo de negocio y permitir a cualquier aplicación escalar hacia una arquitectura basada en Servicios<sup>6</sup>.

En ocasiones, o casi siempre, es necesario transportar entre capas distintos objetos de negocio que se obtienen desde un ambiente persistente, y aquí hay que tomar decisiones en el uso de tecnologías, evaluando aquellas con limitaciones como es el caso de los Entities Beans, dado que estos no pueden vivir fuera de una transacción. En estos casos deben tomarse soluciones tales como:

**Uso de una capa de transformación de DTO<sup>7</sup> o Value Object Pattern.** Pero este patrón tiene más debilidades que fortalezas<sup>8</sup>, pues obliga al mal-uso de la programación orientada a objetos estimulando además la duplicidad de código y limitando el mantenimiento del mismo.

**Evitar el uso de Entities Beans.** Usando en su lugar alguna API de persistencia como *JDBC*, *Hibernate*, *iBatis*, *JDO 2.0*, que permitan la conexión-desconexión de los objetos persistentes, sin dudas una mejor solución a la anterior.

## Separación de las capas web y de negocio.

Comunmente el protocolo más usado para exponer la lógica de negocio en una aplicación a través de la red es el **HTTP**, usando una capa web para permitir a usuarios ejecutar los servicios de la aplicación. Esto es hecho a través de un browser<sup>9</sup>, y por tanto ya existe una **capa remota** entre el usuario y el servicio, pero arquitecturas clásicas u ortodoxas además colocan otra más entre el servidor web y los servicios de negocio.

Que exista una separación conceptual entre la capa web y de negocio no significa que tenga que existir una separación física que involucre el uso de tecnologías remotas entre estas. Por tanto no se justifica que en muchos proyectos web la lógica de negocios sea escrita usando artefactos como los *Session Beans* o *Web Services*, práctica común entre desarrolladores J2EE.

Es más, aunque exista la separación física, la capa de servicios, y como se dijo antes, debe ser una interfaz bien diseñada, donde siempre sea posible en caso de ser necesario, colocar sobre esta una capa de acceso remoto, lo cual incluso ahora nos permitiría escoger que servicios exponer en la red.

Esto último, a pesar que significa un trabajo extra al tener que agregar una capa más, no significa trabajo pesado, e incluso puede ser destinado a personas con más expertise en el tema.

---

4 JSDK (Java Standard Development Kit)

5 De ser necesario el estado puede manejarse en la capa web.

6 SOA

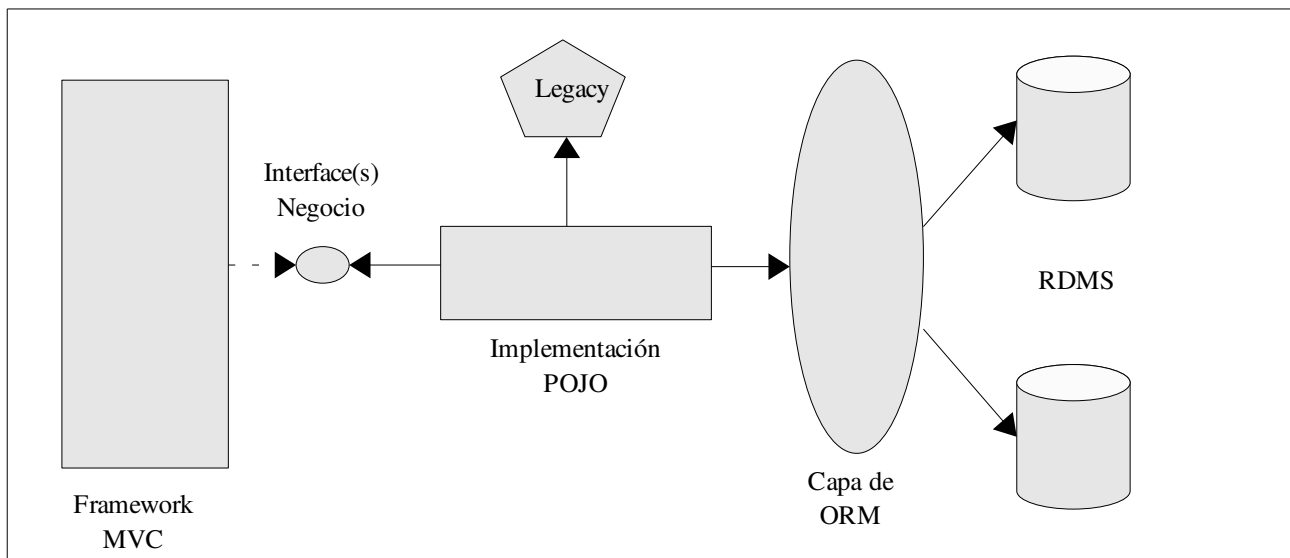
7 Data Transfer Object.

8 To EJB or not To EJB? Revista Informática, edición Enero-Febrero 2005.

9 Cliente Universal.

## El modelo

A continuación el esquema que muestra los elementos de la arquitectura:



La idea de esta arquitectura es evitar cualquier modelo complejo, la única complejidad que debe alcanzar el desarrollo de una aplicación debe estar en la implementación de su lógica de negocio y no en la definición de su arquitectura.

## Ventajas

Las ventajas de este modelo:

- **Simple.** Es una arquitectura basada más en java que en cualquier tecnología.
- **Independiente de un contenedor EJB.** Al no contener por default ningún tipo de EJB y estar basada en interfaces, no es necesario tener un contenedor de EJBs para montar la arquitectura.
- **Portable.** Se deriva del item anterior que la arquitectura sea portable por definición.
- **Testeable.** Dado que la implementación de la lógica de negocio son puras clases Java (POJO) es muy facil escribir y probar la solución.
- **Extensible.** Al estar basada en interfaces, es posible cambiar, o extender la lógica con solo agregar implementaciones nuevas a la solución.
- **Escalable.** Esta arquitectura escala bien tanto vertical como horizontalmente pues posee pocas limitaciones<sup>10</sup>.

## Debilidades

Las debilidades de este modelo.

- **No posee soporte a clientes remotos.** Sin embargo es muy fácil colocar una capa de acceso remoto a los POJOs que implementan la lógica, permitiendo así publicar cualquier servicio en cualquier momento, esto permite el uso de una soluciones SOA.
- **No cumple 100% con los estándares J2EE de Sun.** No obstante, esto permite que la arquitectura **no** sea dependiente de un contenedor EJB, y por tanto la hace mas portable.
- **Poco familiar para desarrolladores EJB.** A muchos desarrolladores de EJBs les cuesta

<sup>10</sup> La mayor limitación siempre dice del manejo de estado en una aplicación, en este caso se le deja a la capa web mediante implementaciones de HttpSession, que por mucho funcionan mejor que los Stateful Session Bean.

trabajo acostumbrarse a pensar que no trabajarán de entrada con componentes de la especificación.

## Tecnologías en cada Capa:

Esta arquitectura debe apoyarse en el uso de frameworks que configurados realicen el trabajo de manipulación de datos y archivos, permitiendo de esta forma concentrar los esfuerzos en la implementación de la lógica de negocios y servicios.

A continuación un conjunto de tecnologías que pueden usarse disyuntivamente en cada capa, para formar arquitecturas basadas en el modelo anterior:

- **Web (Presentación):** JavaServer Faces, Struts, Spring-MVC, Tapestry, WebWork.
- **Lógica de Negocios:** Spring-Core, PicoContainer, Avalon.
- **Datos:** JDBC, Hibernate, Spring-DAO, iBatis, JDO.
- **Remota:** Session Beans Stateless, Web Services (Axis), Hessian, Burlap.

Las consideraciones a tomar en cuenta para seleccionar una u otra tecnología depende de los requerimientos de cada aplicación y escapan del alcance de este artículo.

## Resumen

Existen diversas variantes de modelos de arquitecturas web J2EE, aquí he querido presentarles el modelo que más me gusta, y el set de tecnologías con que puede combinarse para formar **Mi arquitectura preferida**.

Vimos que la arquitectura base de cualquier aplicación J2EE, basada principalmente en:

- **Una bien definida capa de negocios o servicios.**
- **Una capa de presentación.**
- **Una capa de acceso a datos.**

Vimos además las fortalezas y debilidades que presenta dicha arquitectura.

## 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 certificaciones en WebLogic Workshop 8.1 de BEA y WebServices Programmer Certified de IBM, trabaja como arquitecto J2EE para Interplanet y Xperience Consulting Services y puede ser contactado en [jrodriguez@interplanet.cl](mailto:jrodriguez@interplanet.cl).