Unidad 8 – Arquitectura - Persistencia

Industria y mercado

En el mercado actual las bases de datos orientadas a objetos aún no son una opción masiva, completamente madura, probada, difundida o accesible del mismo modo que las base de datos relacionales.

Los fabricantes de bases de datos todavía invierten y promueven en tecnologías relacionales, y se esmeran en producir bases de datos relacionales cada vez más rendimiento con infinitas herramientas dedicadas a su uso (Reporting, Modelado, Administración, etc). Esto es un gran freno para las tecnologías de OODB.

El Problema

Los escenarios actuales de desarrollo suelen trabajar con lenguajes orientados a objetos utilizando bases de datos relacionales, la combinación de estas tecnologías es un tema de discusión permanente que producen una serie de soluciones arquitectónicas diferentes.

Esta mezcla o combinación de tecnologías basadas en diferentes paradigmas plantea varios problemas importantes:

- ¿Se debe diseñar un sistema orientado a objetos y la base de datos debe ser creada en base a este modelo?
- ¿Se debe crear un modelo de datos y diseñar un modelo de objetos dependiente de la estructura de datos?

De momento no existe teoría firme, sino más bien un conjunto de buenas prácticas y recomendaciones que pueden guiarnos a buen puerto si son utilizadas con inteligencia.

Analizando las preguntas anteriores, podríamos rápidamente descartar la segunda por ir en contra de la filosofía misma del paradigma de objetos. [1]

La primera pregunta es un poco más tramposa, por las siguientes razones:

- A simple vista podríamos decir que diseñar un modelo de objetos y que el modelo de datos sea acorde a este parecería ser lo más indicado pero si bien hay una ventaja clara que es realizar un solo diseño (lo que todos nosotros agradeceríamos) tenemos grandes desventajas como la imposibilidad de representar en un modelo relacional (de forma consistente) relaciones de herencia o la imposibilidad de generar de forma simple estructuras de relaciones normalizadas que son el alimento de prácticamente todas las herramientas de reporting del mercado.
- Es posible armar mapeos de clase – tabla y relación – tabla pero estos generarían una estructura de muy baja performance en su navegabilidad.

Otra alternativa, la más usada hoy día, es la combinación de lo “mejor de los dos mundos”:

Diseño orientado a objetos **< + >** Diseño de datos relacional

¿Cómo hacemos esto?

- Con diferentes métodos de mapeo que garanticen independencia de ambos modelos.

[1] Si hacemos la base de datos primero, generaríamos un diseño anémico, compuesto por clases que no tendrían sentido semántico, o responsabilidades concretas, para toda clase que provenga de tablas de entidades débiles que no son más que proyecciones de relaciones entre entidades fuertes.

Estrategias Generales

1. Patrones de Persistencia de Implementación Manual

   Aptos para proyectos pequeños o medianos, proyectos de start-up o proyectos en los que se puedan tomar decisiones de diseño o arquitectura con mayor libertad.

Ventajas	Desventajas
Total control sobre el código generado	Gran es fuerzo de Codificación
Modificaciones con impacto relativamente razonable	Modificaciones con impacto relativamente razonable
Debug Sencillo	

2. Generación de código (SQL modificable)

   Usualmente, basado en herramientas de ingeniería reversa de la base de datos y plantillas de templates.

Ventajas	Desventajas
Control de código parcial	Control de código parcial
Bajo nivel de errores	Requieren generar el código ante un cambio en el modelo de datos
Permiten el tunning de SQL	No solucionan los “joints”

3. O/R Mappers (Generación Dinámica de SQL)

   Usualmente basada en frameworks y archivos de mapeo. Cada tecnología suele aportar su propia

Ventajas	Desventajas
Control Poco control de código	Control Poco control de código
Solucionan “joints” (pero con costo de performance a considerar)	Solucionan “joints” (pero con costo de performance a considerar)
Utilizan lenguajes de consulta propietarios con semántica OO (que dan portabilidad del fabricante de la base de datos)	Utilizan lenguajes de consulta propietarios con semántica OO (que generan dependencia con el fabricante del framework)
Son relativamente fáciles y rápidos de usar.	Difícil de debugear
	SQL Generado dinámicamente con poco o nulo tunning.

4. Modelos híbridos (Capa CRUD)

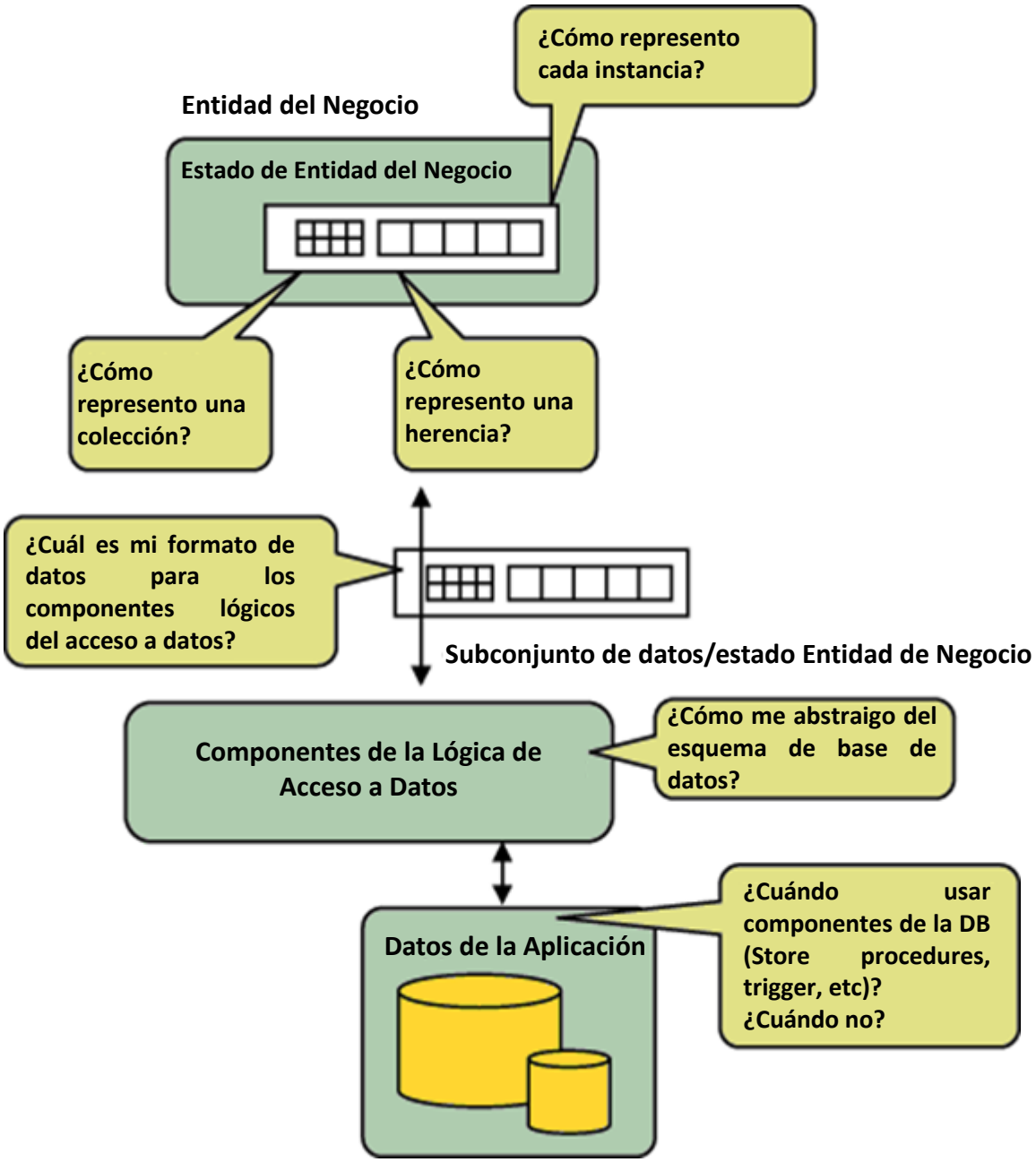
   Cuando ninguna de las estrategias anteriores resuelve completamente el problema, cuando se busca compensar las ventajas y desventajas de cada estrategia anterior, se suele recurrir a esta solución.

   Se trata simplemente de implementar una capa propia CRUD (Create – Remove – Update – Delete), que busque alojar y combinar alguna o varias de las estrategias anteriores. Esta solución será posible de acuerdo al nivel de libertad de decisión o administración del esfuerzo con la que contemos.

Desafíos de Capa de Persistencia

El siguiente gráfico, se muestran las usuales preguntas, por lo general abiertas, que surgen en el abordaje del problema de la persistencia en una base orientada a objetos.

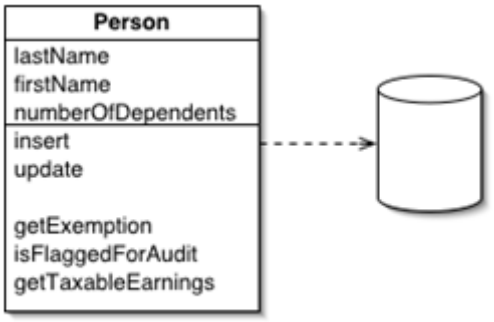
Más allá de las soluciones que se presentan en este apunte, es importante tener en cuenta estos desafíos y registrar las decisiones de diseño que se tomen en cada caso.



Patrones de Persistencia de Implementación Manual

1. Active record

Se trata de que el mismo objeto de negocio, contenga el registro de una tabla (o vista). Esto implica que el mismo objeto se ocupe de encapsular el acceso a la base de datos y manejar la lógica del dominio a los datos.

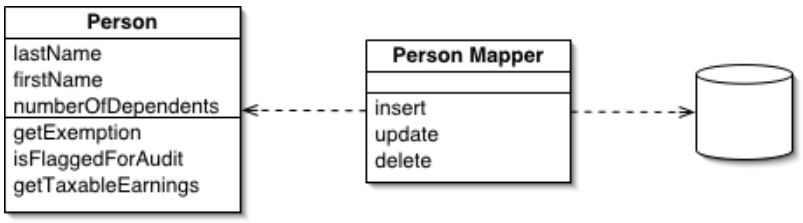


Esta solución implica un máximo acoplamiento (y por ende dependencia) entre los modelos del dominio y la base de datos. Es una solución de baja complejidad. Es recomendable para proyectos muy pequeños.

Sin embargo viola los principios de asignación de responsabilidad sobre los objetos, dado que le estamos pidiendo al objeto que haga algo más de lo que le corresponde: persistir sus propios datos.

2. Data Mapper

Se trata de agregar un objeto específico que realice el mapeo de la estructura del objeto con la que corresponde a la de la base de datos.



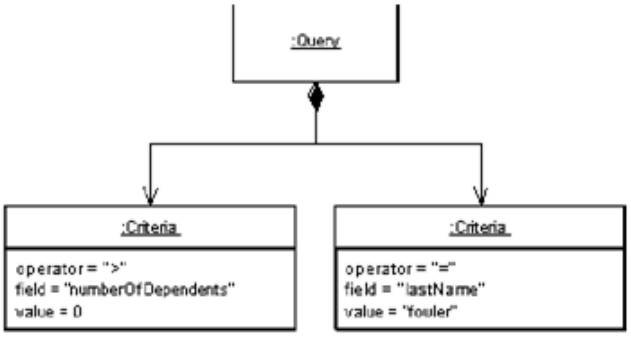
Por cada objeto que necesite persistencia terminaremos teniendo un objeto (o artefacto de persistencia) adicional. Es terminará definiéndonos una capa de objetos que mueven datos entre los objetos de negocio y la base de datos manteniendo a cada uno independiente del otro.

Este patrón, presenta una opción que mantiene las responsabilidades donde deben estar. Es muy utilizado, tiene gran cantidad de variantes de implementación. Como desventaja se puede ver que en un modelo grande la cantidad de clases de mapeo también será grande. Es muy utilizado por herramientas de generación de código y por la estrategia de O/R Mappers.

En ocasiones, dependiendo del lenguaje, o la forma de implementación, el mapeo se suele realizar con archivos de configuración externos (archivos XML por ejemplo) externos a la clase componente del patrón. Dependiendo del punto de vista esto resulta de relativa comodidad dado que no hay que recompilar el sistema si los cambios son de bajo impacto.

3. Query Object

Se trata de que un objeto represente y contenga un Query de la base de datos. Esto implica que el objeto modele al query con todos sus componentes reglas y restricciones. En la imagen de abajo se muestra una posible representación del query.

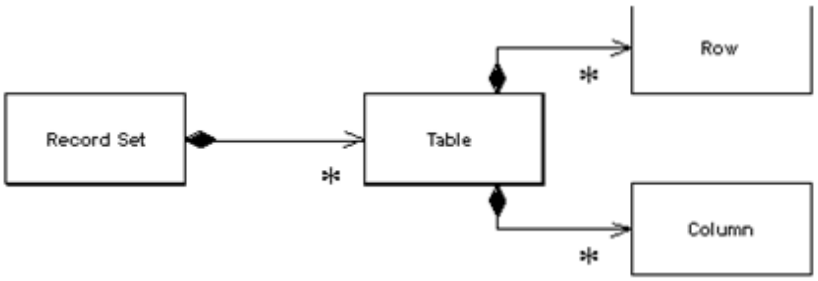


Se utiliza en sistemas de O/R Mappers. El Query Object puede implementarse con la ayuda o asistencia de otros patrón de diseño de distinto nivel el: **Interpreter** [ver libro del GoF página.243], y del patrón Builder debido a la complejidad, secuencia y variantes de sus componentes.

4. Recordset

Es una estructura de datos, representada por el diagrama de clase de abajo, cuya utilidad es la de almacenar información desde una tabla de una base de datos.

Se utiliza mayoritariamente en lenguajes de programación propietarios de Microsoft como Visual Basic ó C++.



En la actualidad el proveedor ha propuesto otros patrones más modernos, prácticos y eficientes, pero aún se sigue utilizando.

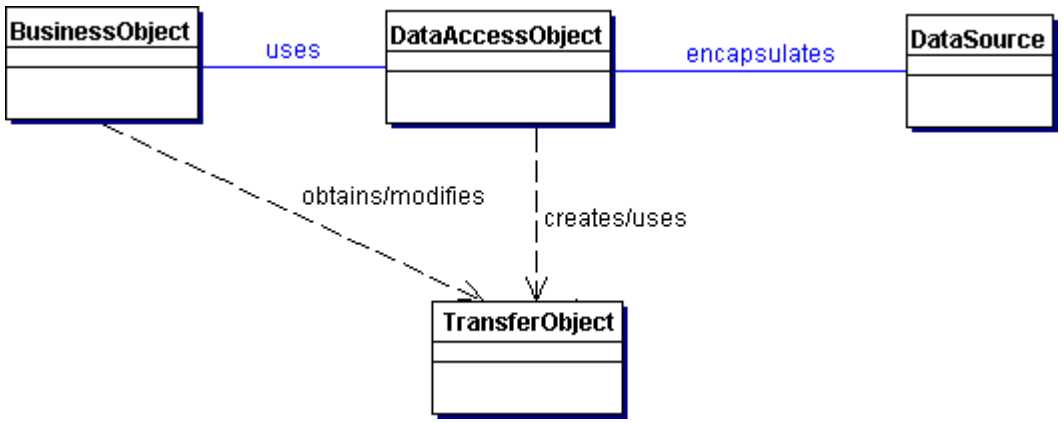
5. DAO (Data Access Object)

Suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos (Base de datos, archivos, etc) . Se trata de que el objeto de negocio le pida a su objeto de persistencia DAO lo necesario, y que éste sea el que implemente la lógica de acceso a los datos, junto con el armado de las consultas necesarias a la base en SQL.

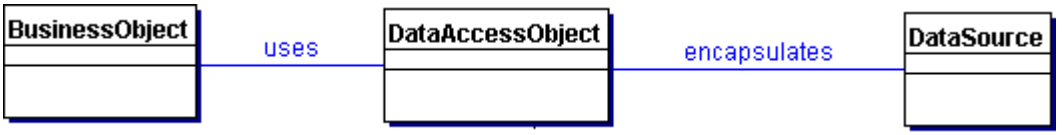
La idea central es que los objetos de negocio no tengan que saber nada de cómo se guardan o recuperan los datos. En la imagen de abajo, se muestra la forma más completa y/o formal del patrón, en la que se considera un objeto de transferencia de datos (DTO) como elemento de intercambio o de transporte. Muchas veces en la realidad el patrón se suele implementar sin objeto de transporte o trasferencia de datos.

En alguna medida, este patrón provee parte de lo provisto por el patrón Data Mapper, solo que sin la mecánica del mapeo. También comparte sus ventajas y desventajas respecto del aumento de los artefactos involucrados y la necesidad de establecer una capa de persistencia adicional para alojar estas clases adicionales.

Con objeto de transferencia



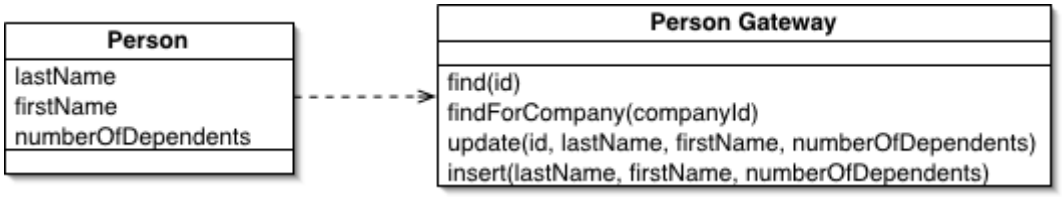
Sin objeto de transferencia



6. Table Data Gateway

Se trata de un objeto que actúa como Gateway a una tabla de una base de datos. Consta de una instancia maneja todos los registros de una tabla. Suele ser una solución propietaria de un lenguaje o de un proveedor de tecnología (framework en este caso).

En ocasiones esta solución puede condicionar el diseño (aumentando el acoplamiento) debido a la manera en la que hay que relacionarse con las clases Gateway, conociendo la manera en la que trabajan y en alguna medida su implementación.

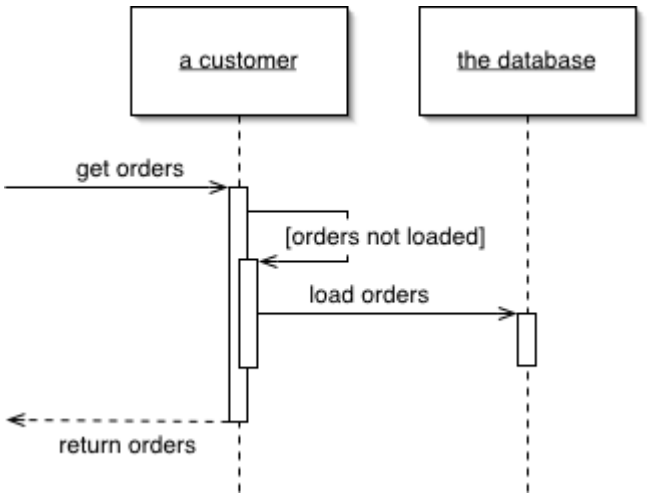


Paginado de Datos en la Persistencia

De un modo similar a lo que ocurre entre cualquier cliente y servidor, respecto de la necesidad de manejar el paginado de datos entre el servidor y las pantallas, se nos presenta la misma necesidad dado que ya tenemos una nueva capa adicional de persistencia que recupera datos y debe pasarlos a las otras capas (modelo, vista, u otras). Para esta tarea se presenta en primera instancia el patrón Lazy Load

Lazy load

Se trata de implementar en un objeto, una lógica tal que no contenga todos los datos que se necesiten según indica su estructura. Su estado está parcialmente vacío, pero sabrá cómo conseguir los datos faltantes que necesite en la medida que los vayan pidiendo.



Este patrón es uno de los más utilizados no solo en frameworks de persistencia sino en cualquier objeto que tenga una gran cantidad de datos y que estos sean requeridos en pocas ocasiones. Mejora mucho la performance de la aplicación en cuanto a consumo de memoria.

Instituto de Tecnología ORT
Carrera de Analista de Sistemas
Materia: Programación II

Comunicación y Transferencia de datos entre capas

1. Pasaje de escalares:

Se trata de usar las primitivas del lenguaje para la comunicación transferencia de datos entre capas y/o entre aplicaciones de la misma tecnología.

Ventajas	Desventajas
Abstracción (no conoce la estructura del objeto)	Alto acoplamiento, las interfaces varían con los cambios de estructura internos de las clases.
Soporte de serialización	Manejo pobre de colecciones
Buen uso de memoria	Concurrencia optimista desconectada

2. Pasaje de XML

Es un meta-lenguaje que permite definir marcas utilizado para almacenar datos en forma legible. Da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información. Propone un estándar para el intercambio de información estructurada entre capas de una aplicación o entre diferentes plataformas.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande, con posibilidades mayores. Tiene un papel importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Ventajas	Desventajas
Bajo acoplamiento (el cliente solo conoce el esquema)	Bajo rendimiento por sobreesfuerzo en el parseo
Integración	Pobre uso de memoria
Soporte de colecciones	Concurrencia Optimista Desconectada
Soporte de serialización	Complejidad en el desarrollo del parser.

3. JSON

Acrónimo de JavaScript Object Notation, es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera ya un formato independiente del lenguaje.

Ventajas	Desventajas
Resulta mucho más sencillo escribir un parser para él.	Cuando la fuente no es conocida, puede presentar vulnerabilidades de seguridad.
Fácil evaluación de un texto JSON en el lenguaje nativo (JScript).	Debido a lo anterior en muchas ocasiones se usa en forma combinada con XML.
Soporte de grandes colecciones de datos	
Soporte de serialización	

4. Pasaje de estructuras tabulares (DataSet, Resulset, etc)

Ventajas	Desventajas
Funcionalidad nativa de algunas plataformas	Performance (gran esfuerzo en la inicialización, implica procesos de "marshaling")
Soporte de colecciones con relaciones de conjunto	La representación de una única entidad representa un costo muy alto para una estructura tabular
Bajo acoplamiento (los cambios de esquema no necesitan cambios en la interface)	Son tecnologías propietarias y antiguas al borde de la obsolescencia.
Soporte de serialización	
Manejo de concurrencia transparente	

5. Pasaje de clases DTO (Data Transfer Object) y Value Object (VO).

Son dos denominaciones para un concepto muy similar. Se trata de un objeto que solo transferencia entre capas, procesos o aplicaciones. La motivación de su uso tiene relación con el hecho que la comunicación se realiza generalmente mediante interfaces remotas (por ejemplo, servicios web), donde cada llamada es una operación costosa. Como la mayor parte del costo de cada llamada está relacionado con la comunicación de ida y vuelta entre el cliente y servidor, una forma de reducir el número de llamadas es usando un objeto (el DTO/VO) que agrega los datos que habrían sido transferidos por cada llamada, pero que son entregados en una sola llamada.

La diferencia entre un objeto de transferencia de datos y un objeto de negocio (*business object*) o un objeto de acceso a datos (data access object, DAO) es que un DTO/VO no tiene más comportamiento que almacenar y entregar sus propios datos (métodos mutadores y accesores). Los DTO son objetos simples que no deben contener lógica de negocio que requiera pruebas generales.

Ventajas	Desventajas
Bajo acoplamiento (la modificación del DTO no requiere cambios en la interface)	Bajo nivel de integración (aunque es posible integrar mediante serializacion)
Soporte de colecciones	Concurrencia Optimista
Diseño semántico	

Bibliografía:

- Design Patterns – Adison Wesley [GoF]
- Pattens of Enterprise Application Architecture – Adison Wesley (Martin Fowler)
- https://www.oracle.com/technetwork/java/index.html
- http://msdn.microsoft.com/architecture
- Wikipedia.