PATRONES DE DISEÑO EMPRESARIALES – PRIMERA PARTE

ELSA ESTEVEZ
UNIVERSIDAD NACIONAL DEL SUR
DEPARTAMENTO DE CIENCIAS E INGENIERIA DE LA COMPUTACION

CONTENIDO

Para que sirve un patrón de diseño?

Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a aquel problema, de forma tal que dicha solución pueda utilizarse un millón de veces más, sin hacer exactamente lo mismo dos veces.

Christopher Alexander

Aspectos importantes:

- Un patrón define el núcleo de la solución de un problema y no la solución exacta a aplicar
- La solución provista por un patrón de diseño nunca se puede aplicar a ciegas, siempre es necesario considerar el contexto del problema.

CONOCIMIENTO SOBRE PATRONES

CONOCIMIENTO	
Requerido	Opcional
Qué patrones existen	Los detalles de la solución
Qué problemas resuelven	Las variantes posibles
Una idea de la manera en que resuelven el problema	

PROPIEDADES

Independencia relativa

- Cada patrón es relativamente independiente, pero no están aislados unos de otros.

Auto-definición

- Los límites entre patrones muchas veces son muy difusos, pero se intenta que sean tan auto-definidos, tanto como se pueda.

Ayuda en la comunicación

- Para desarrolladores experimentados, el valor de un patrón de diseño no está en obtener una nueva idea, sino en ayudar a comunicar la idea.

APLICACIONES EMPRESARIALES

EJEMPLOS?

APLICACIONES EMPRESARIALES - EJEMPLOS

Aplicaciones Empresariales	Aplicaciones No Empresariales
Sistema de sueldos	Controlador de ascensores
Administración de pacientes	Procesador de texto
Seguimientos postales	Inyección de combustible en vehículos
Sistema de seguros	Controlador de planta
Sistema de clientes	Switch telefónico
Sistema para licencias de conductor	Compilador
Sistema de stock	Sistema operativo
Venta electrónica	Juego
Sistema de pedidos	Software de lavarropa

CARACTERISTICAS – 1

1) Datos persistentes – La información persistida generalmente es el núcleo del sistema y su ciclo de vida es más amplio que el del resto de los componentes.

2) Gran cantidad de datos – Un sistema medio contará con alrededor de 1GB de datos, organizados en decenas de millones de registros, siendo esta información la mayor parte del sistema.

3) Acceso concurrente a datos – Generalmente muchas personas acceden a los datos concurrentemente. Debido a esto, hay problemas que deben abordarse para asegurar que los usuarios pueden acceder a la información de manera confiable.

CARACTERISTICAS – 2

4) Gran cantidad de interfaces de usuario – Para manejar tanta información es necesario disponer de cientos de pantallas que permitan presentar los datos de distinta forma y, eventualmente, para distintas audiencias.

5) Integrado con otras aplicaciones empresariales – Las aplicaciones empresariales raramente existen de manera aislada; generalmente requieren integrarse con otras aplicaciones de la empresa o fuera de la misma.

6) Lógica de negocio compleja – Las reglas de negocio son prioritarias y muchas veces impuestas y por más que intentemos darle integridad lógica, no se puede hacer mucho. Esto es lo que conduce a una “compleja ilógica de negocio” que hace que el software de negocios sea tan complejo.

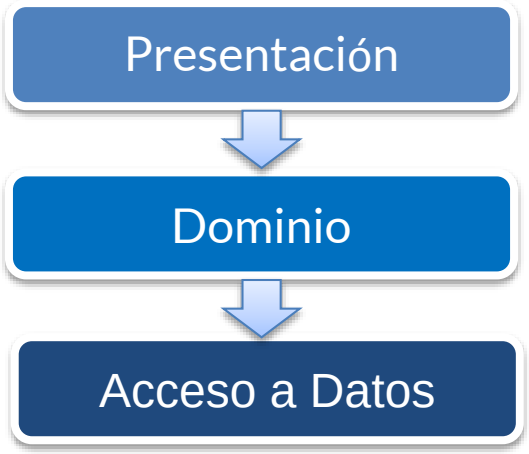
CAPAS

Una posible separación en capas a considerar:

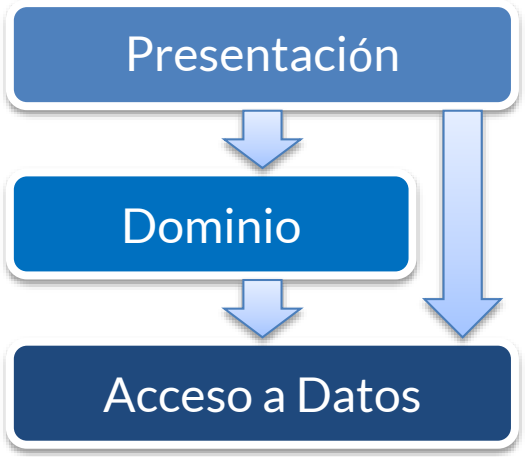
Presentación	• Interpretar los comandos del usuario y mostrar información al usuario.
Dominio	• Tareas que la aplicación necesita realizar para cumplir con los requerimientos del negocio que resuelve la aplicación.
Acceso a Datos	• Comunicarse con otros sistemas que realizan tareas especificas – base de datos, mensajería, etc.

DETALLES DE IMPLEMENTACION

Disposición de las capas



Opción 1



Opción 2

¿Cómo implementar una capa?

Layering

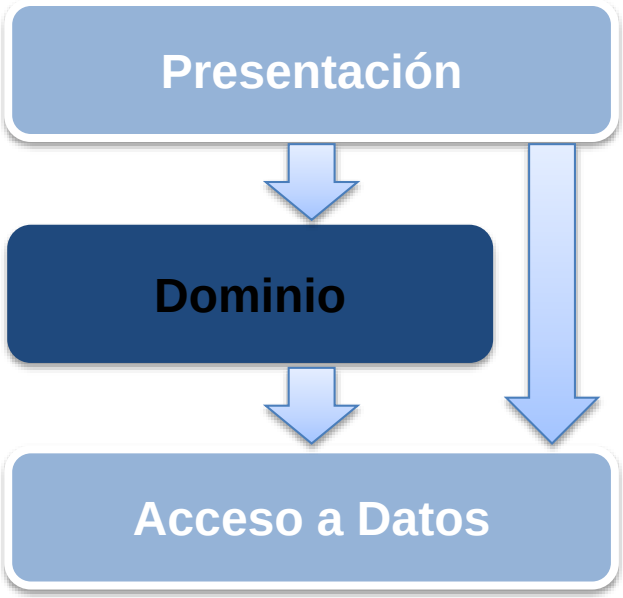
- → Métodos
- → Clases
- → Componentes

Depende de la complejidad de la aplicación

CONTENIDO

PATRONES DE CAPA DE DOMINIO

1) Transaction Script
2) Domain Model
3) Table Module
4) Service Layer

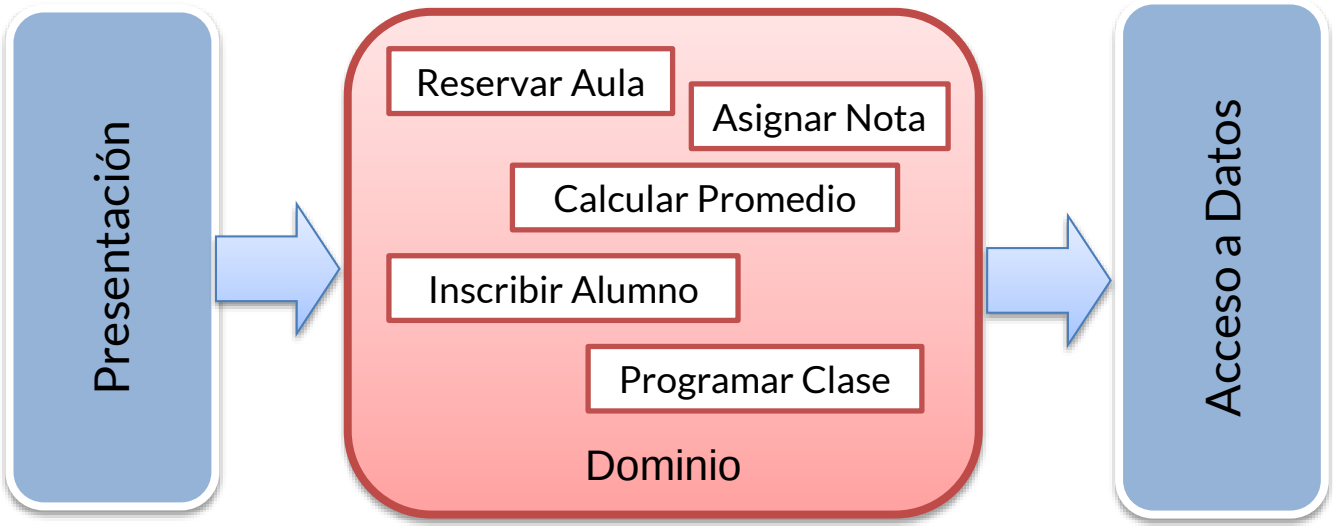


1 – TRANSACTION SCRIPT

DESCRIPCION - organiza la lógica de negocio por procedimiento, donde cada procedimiento maneja un único pedido desde la capa de presentación.

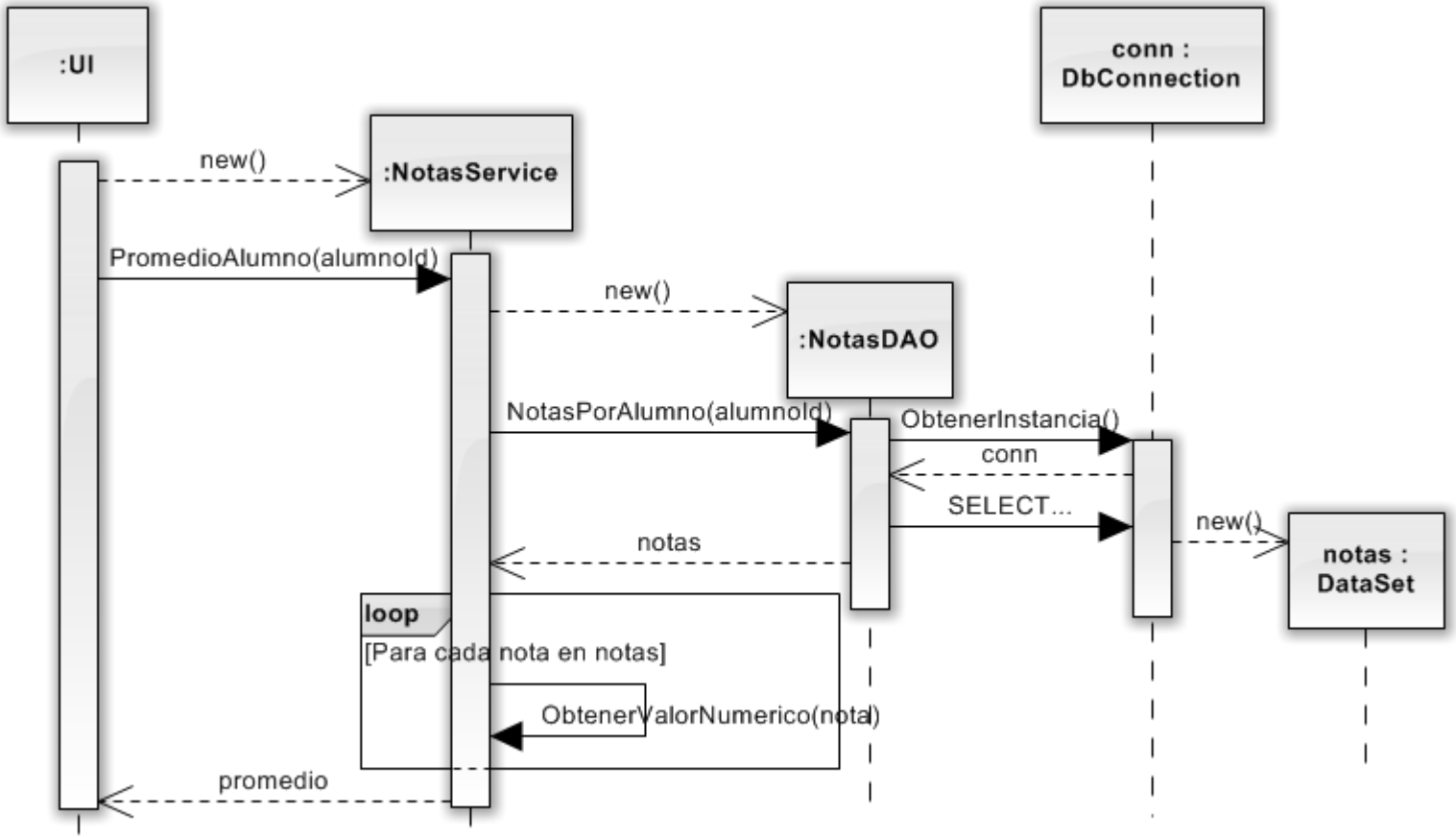
- un Transaction Script organiza cada transacción de negocio (funcionalidad) en un único procedimiento, haciendo las llamadas pertinentes a la capa de acceso a datos.

- cada transacción de negocio tendrá su propio Transaction Script, aunque sub-tareas comunes pueden ser aisladas en sub-procedimientos.



TRANSACTION SCRIPT – COMO TRABAJA?

Un Transaction Script se podría resumir en validar los datos de entrada, consultar la base de datos, realizar cálculos y guardar los resultados en la base de datos



TRANSACTION SCRIPT – UBICACION

¿Dónde poner los Transaction Scripts?

- una server page
- un script CGI (Common Gateway Interface)
- un método
- una clase
- un conjunto de clases

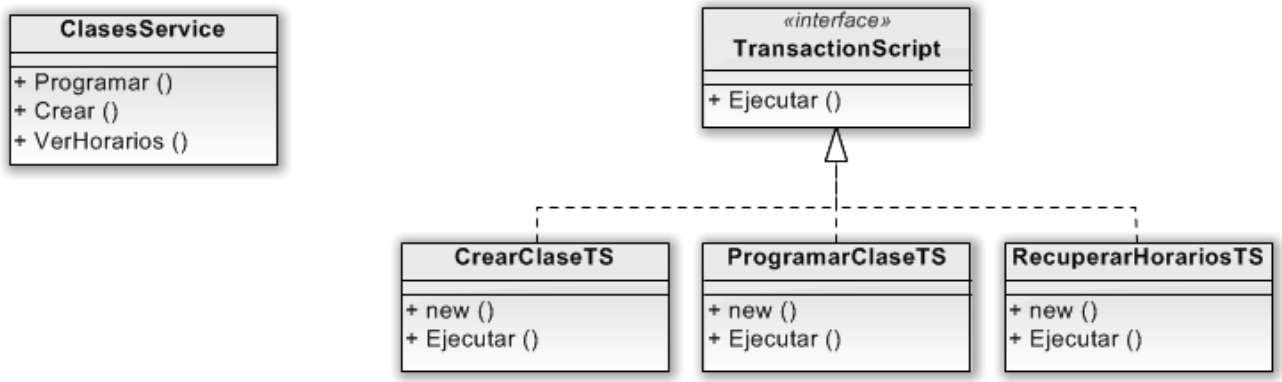
TRANSACTION SCRIPT – RECOMENDACIONES 1

- separar los Transaction Script tanto como sea posible - al menos, ponerlos en distintos métodos; mejor aún si están en clases distintas a las usadas en Presentación y Acceso a Datos.

- no realizar ninguna llamada desde un Transaction Script a lógica de presentación – para facilitar el testeo y mantenimiento

TRANSACTION SCRIPT – RECOMENDACIONES 2

- poner los Transaction Scripts en clases de una de la siguiente forma:
  - tener varios TS en una única clase, donde cada clase define un área sujeto de TS relacionados. Esto es directo y la más apropiada en muchos casos. Ejemplo: NotasService, ClasesService, etc.
  - tener cada TS en su propia clase utilizando el patrón Command[1] (GoF)



[1] Este patrón permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto, con lo que además facilita la parametrización de los métodos.

TRANSACTION SCRIPT – USO

Ventajas

- Dada su simplicidad, es natural utilizarlo en aplicaciones que tengan poca lógica de negocios, ya que involucra poco overhead tanto en performance como en curva de aprendizaje.

Limitaciones

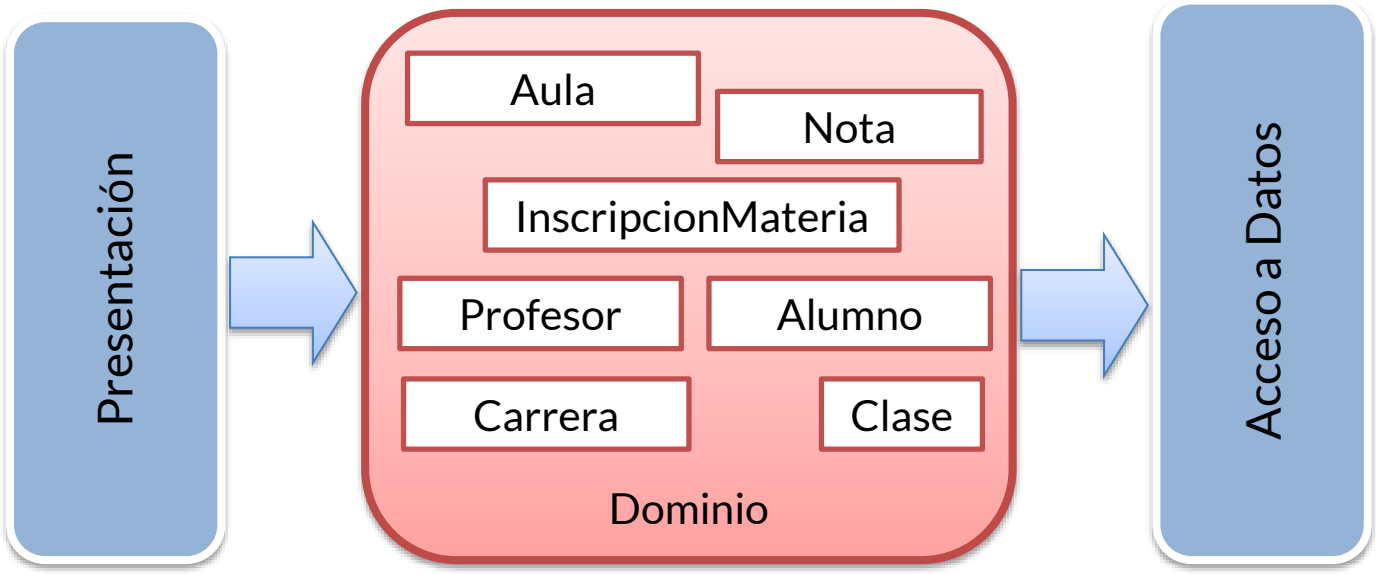
- A medida que la lógica de negocio se torna más compleja, es difícil mantenerlo con un buen diseño.

- Generalmente se tiene problemas con la duplicación de código. Puesto que el foco está en resolver una transacción, muchas veces se tiende a duplicar código común.

2 – DOMAIN MODEL

DESCRIPCION - Un modelo de objetos del dominio que incorpora tanto comportamiento como datos.

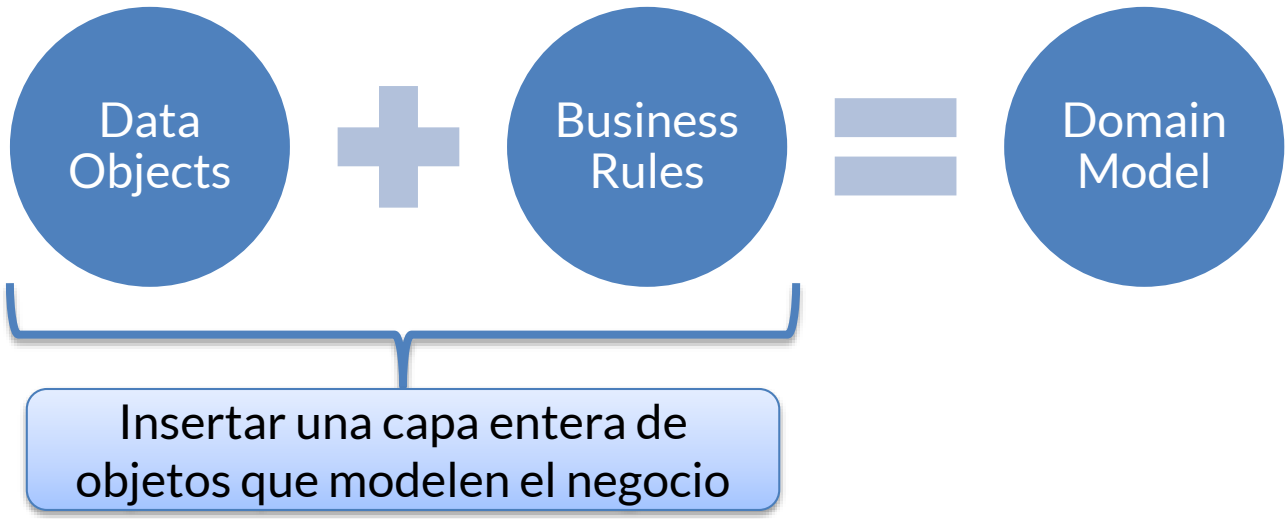
- un Domain Model crea una red de objetos interconectados, donde cada objeto representa a un concepto significativo, ya sea tan grande como una corporación o tan pequeño como un ítem de una factura.



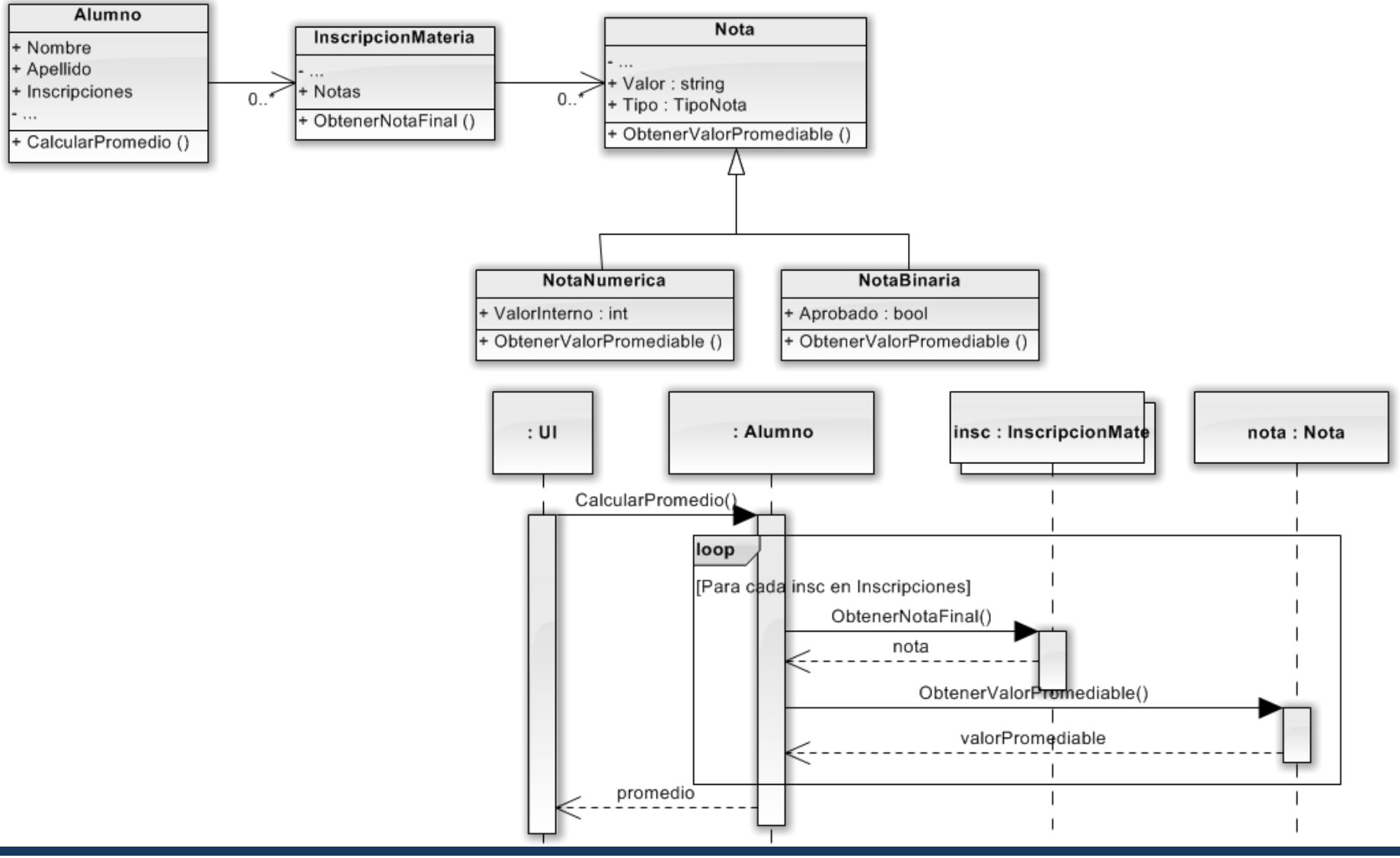
DOMAIN MODEL – COMO TRABAJA?

Muchas veces se utilizan dos tipos de objetos para la capa de negocios: objetos que representan los datos y objetos que capturan las reglas que el negocio utiliza.

Con este patrón se combinan estos dos tipos de objetos para poner los procesos de negocio cerca de los datos con los que trabaja.



DOMAIN MODEL – EJEMPLO



DOMAIN MODEL – CARACTERISTICAS

Aunque muchas veces encontramos Domain Models (DM) que son muy similares a modelos de base de datos, existen varias diferencias entre ellos:

- DM mezcla datos y procesos
- DM tiene atributos multivaluados
- DM tiene una compleja red de asociaciones
- DM usa herencia

DOMAIN MODEL – TIPOS

- Simple – luce muy parecido al modelo de base de datos, normalmente con un objeto de dominio por cada tabla de la base de datos. Puede utilizar el patrón *Active Record* [1]para el acceso a datos.

- Rico – puede lucir diferente al modelo de base de datos, incorporando herencia, strategies, otros patrones de diseño orientados a objetos [GoF] y redes complejas de pequeños objetos interconectados. Requiere utilizar el patrón *Data Mapper*[2] para el acceso a datos.

[1]Guarda los datos de un objeto en memoria en una base de datos relacional
[2]Guarda en un almacenamiento, los datos de una entidad del mundo real a través de operaciones CRUD – Create, Read, Update, Delete

DOMAIN MODEL – USO

Existen varios factores que pueden determinar el uso o no de un Domain Model:

- Complejidad de comportamiento del sistema

  Si tenemos reglas de negocio complicadas y cambiantes que involucran validaciones, cálculos, etc., sería preferida la utilización de un Domain Model. En cambio, si solamente se tienen chequeos de campos requeridos y un par de sumas para realizar, probablemente sea suficiente con Transaction Script.

- Comodidad del equipo en el uso de objetos de domino

  Aprender a diseñar y usar un Domain Model es un ejercicio significativo. Principalmente lleva práctica y coaching, pero una vez que se utiliza correctamente pocos profesionales quieren volver a un Transaction Script para situaciones algo o muy complejas.

- Posibilidad de utilizar un Data Mapper para el acceso a datos

  En algunas circunstancias, no es posible o lleva mucho esfuerzo utilizar un Data Mapper y esto complica la utilización de Domain Model.

3 – TABLE MODULE

DESCRIPCION – se basa en tener una única instancia que maneje la lógica de dominio para todas las filas en una tabla o vista de la base de datos.

- un Table Module organiza la lógica de dominio con una clase por tabla de base de datos, y una única instancia de una clase contiene los distintos procedimientos que actuarán sobre los datos. Así se logra también agrupar los datos con el comportamiento que los utiliza

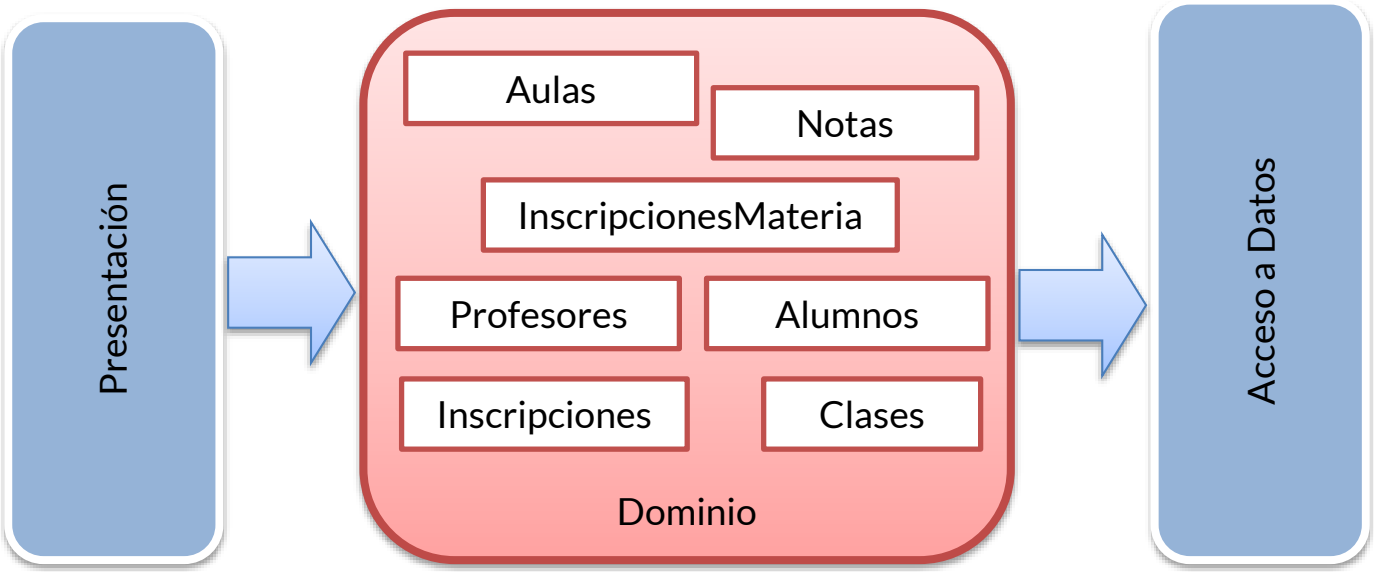


TABLE MODULE AND DOMAIN MODEL

En que se diferencian?

Si tenemos muchas instancias de una entidad, Domain Model tendrá un objeto por cada instancia; mientras que Table Module tendrá un objeto que maneje todas las instancias.

Si tenemos muchos alumnos, Domain Model tendrá un objeto Alumno por cada alumno; mientras que Table Module tendrá un objeto que maneje todos los alumnos.

TABLE MODULE – COMO TRABAJA? 1

Table Module parece a un objeto regular, pero posee características especiales:

Sin identidad

- Table Module no tiene noción de identidad de los objetos con los que trabaja.
- Ejemplo: Obtener el nombre de un alumno.

```
string alumnosModule.ObtenerNombre(long alumnoID)
```

Estructura de datos de respaldo

- generalmente se usa un Table Module con una estructura de datos de respaldo que sea orientada a tablas.
- normalmente estos datos tabulares resultan de una consulta SQL y se mantiene en un Record Set que imita a la tabla en la base de datos.
- la forma de uso más común es tener una Table Module por cada tabla en la base de datos. Sin embargo, si se presentan vistas o consultas de base de datos relevantes, se puede tener un Table Module para ellas también

TABLE MODEL – COMO TRABAJA? 2

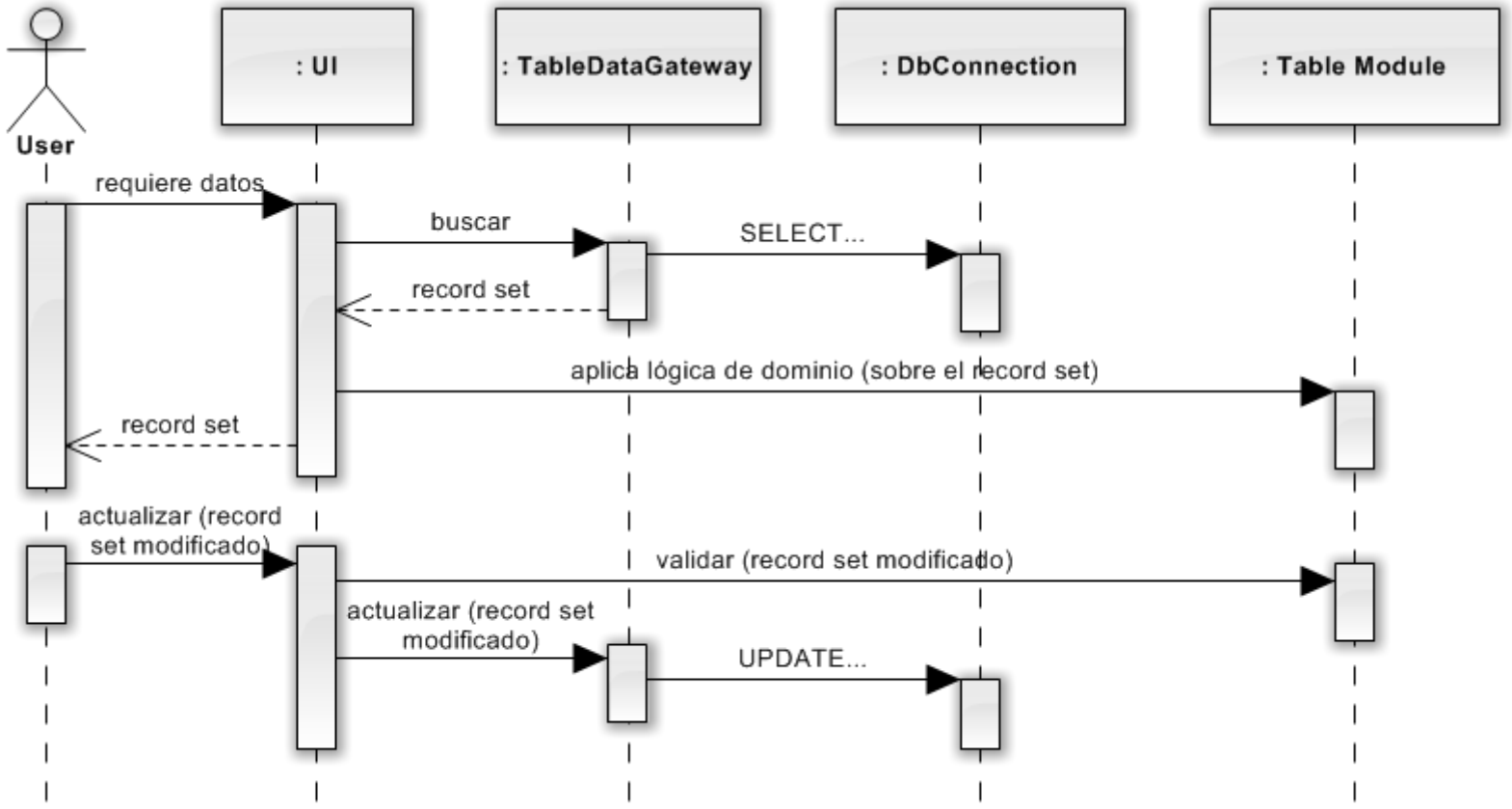
Table Module parece a un objeto regular, pero posee características especiales:

Implementación

- colección de métodos estáticos - poco usada ya que siempre trabajaría sobre la tabla entera

- instancia:
  - permite inicializar el Table Module con un record set existente (probablemente el resultado de una consulta)
  - permite el uso de herencia - se podría construir un Table Module `AlumnoPosgrado` que contenga comportamiento adicional al alumno regular

TABLE MODEL – EJEMPLO 1



Interacción típica de las capas alrededor de un Table Module

TABLE MODEL – EJEMPLO 2

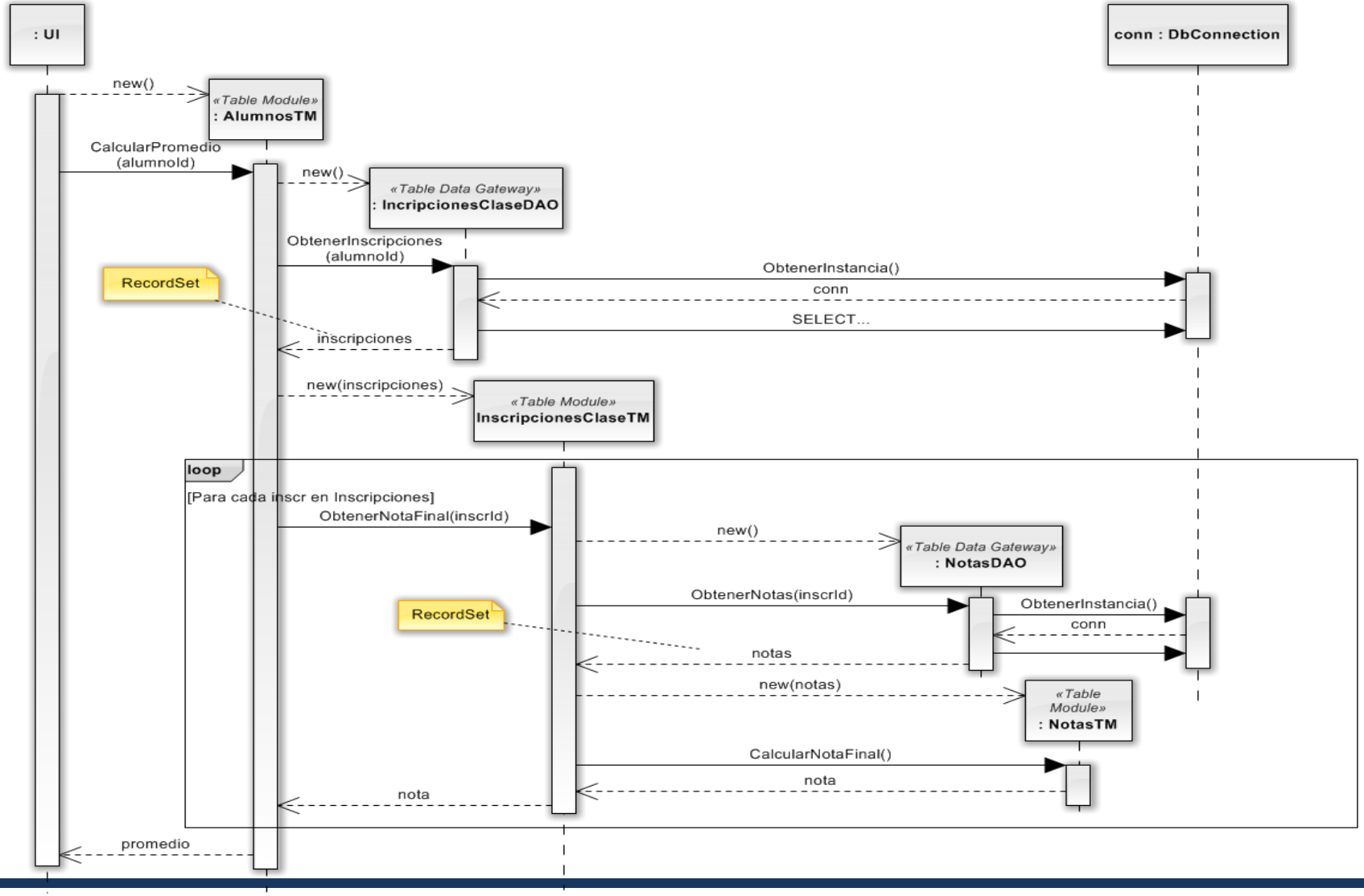


TABLE MODEL – USO 1

Las recomendaciones ante distintos escenarios, podrían ser:

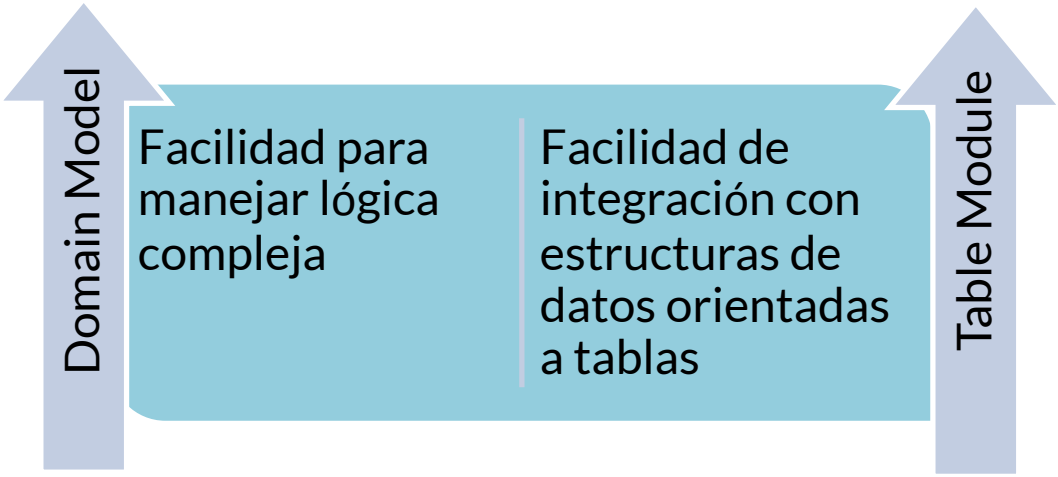


TABLE MODEL – USO 2

Ventajas

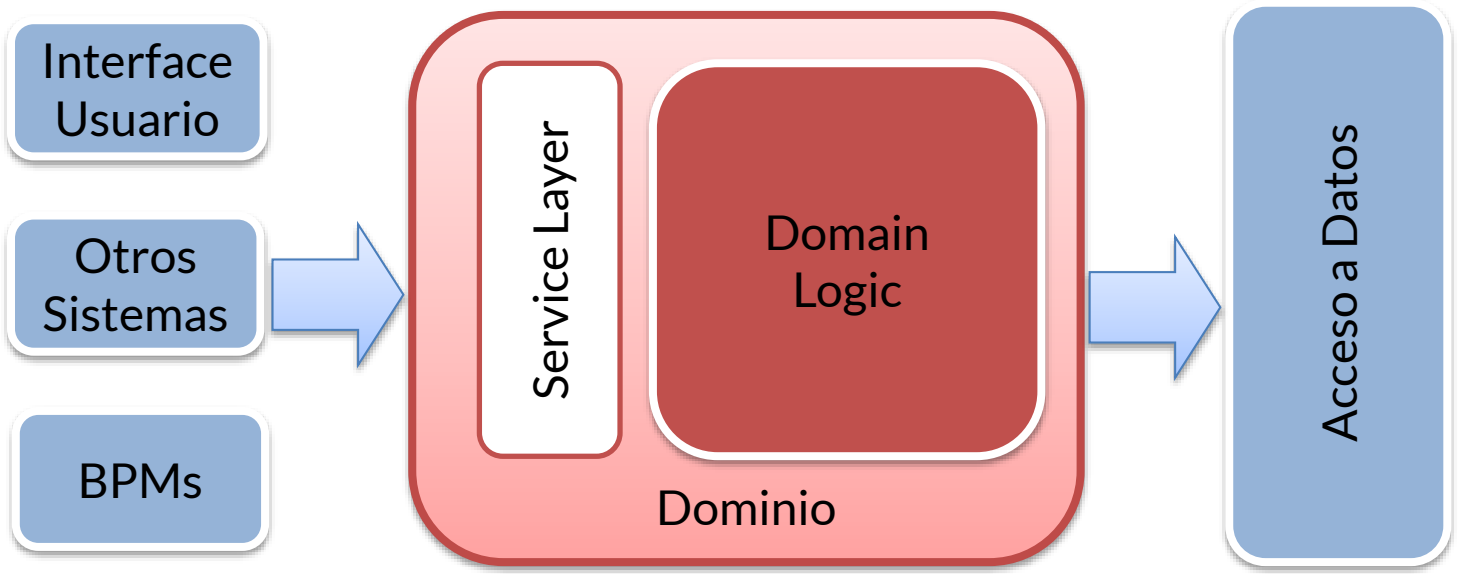
- dado que está basado en la manipulación de datos orientados a tablas, cobra mayor sentido utilizarlo cuando se está accediendo a datos tabulares usando Record Set
- trabaja mejor que Domain Model + Active Record cuando otras partes de la aplicación están basadas en una estructura de datos orientada a tablas

Desventajas

- en casos de alta complejidad de lógica de negocios, Domain Model es una mejor opción
- con Table Module se pierde el poder de la orientación a objetos en organizar lógica compleja - no se pueden tener relaciones entre instancias, ni polimorfismo

4 – SERVICE LAYER

DESCRIPCION - Define el límite de una aplicación con una capa de servicios que establecen el conjunto disponible de operaciones y coordina la respuesta de la aplicación en cada operación.



SERVICE LAYER – MOTIVACION

Ejemplo de razones para proveer una capa de servicios (Service Layer)

- Proveer una API sobre la lógica de negocio - que puede ser consumida desde distintos clientes: interfaces de usuarios, otras aplicaciones, BPMs, etc.

- Manejar transacciones a través de múltiples recursos - por ejemplo, transacciones distribuidas; y coordinar distintas respuestas para una operación - workflows, lógica de interacción con otros sistemas

SERVICE LAYER – FUNCIONALIDAD

¿Qué se resuelve en la Service Layer? La lógica de aplicación

- relacionada a responsabilidades de la aplicación – por ejemplo, envío de notificaciones, integración con otras aplicaciones, etc.)
- también se la conoce como "lógica de workflow" - tipo de lógica involucrado en una Service Layer.
- Service Layer factoriza la lógica de aplicación en una capa separada, promoviendo los beneficios conocidos de layering y produciendo clases de negocio puras y más reusables de aplicación en aplicación.
- Desventajas de poner lógica de aplicación dentro de las clases de la lógica de dominio:
  - las clases de domino son menos reutilizables entre aplicaciones
  - mezclar ambos tipos de lógica en una misma clase hace más difícil reimplementar la lógica de aplicación en, por ej., una herramienta de workflows.

SERVICE LAYER – REMOTO O NO 1

La interface de una clase Service Layer es de “granularidad gruesa” por definición, ya que declara un conjunto de operaciones disponibles a las capas clientes - como una API del negocio. Por lo tanto, son muy apropiadas para invocación remota.

Costo:

- tratar con distribución de objetos – puede ser muy complicado

- trabajo extra con Data Transfer Objects (DTOs) - El costo de transformar la entrada y salida de la Service Layer en DTOs puede ser muy alto, especialmente para dominios complejos. Existen herramientas que pueden ayudar a sobrellevar este costo, pero no implica que sea gratis.

SERVICE LAYER – REMOTO O NO 2

Consejo:

- comenzar con una Service Layer invocable localmente, cuyos métodos hablen en términos de los objetos del dominio

- hacerla remota si fuera necesario, poniendo Remote Facades sobre la Service Layer, o haciendo que la Service Layer implemente interfaces remotas

- si la aplicación tiene una capa de presentación web, no hay nada que diga que la lógica de negocio tiene que correr en un proceso separado. Por el contrario, se puede ahorrar esfuerzo de desarrollo y tiempo de respuesta sin sacrificar escalabilidad con una aproximación local.

SERVICE LAYER – SERVICIOS Y OPERACIONES

Identificación de servicios y operaciones:

- las operaciones necesarias en una Service Layer están determinadas por las necesidades de los clientes de dicha Service Layer; generalmente la interface de usuario

- basarse en:
  - modelo de casos de uso
  - pantallas
  - necesidades de interacción con otros sistemas

SERVICE LAYER – USO

Indicado

- más de un tipo de cliente sobre la lógica de negocio
- se necesita hacer uso de múltiples recursos transaccionales

No indicado

- un único tipo de cliente - generalmente la interface de usuario
- se trabaja con una única base de datos

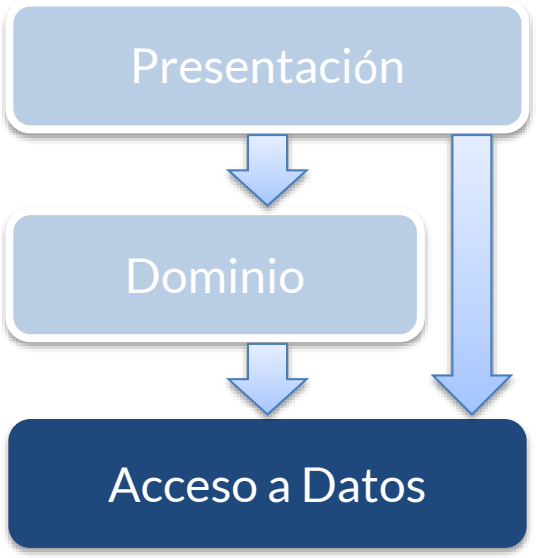
CONTENIDO

ACCESO A DATOS – MOTIVACION

¿Cómo se puede resolver el desajuste entre modelo de objetos y modelo relacional?

Existen distintos tipos de patrones que consideran la problemática de la persistencia de datos:

- Patrones Arquitectónicos
- Patrones de Comportamiento Objeto-Relacional
- Patrones Estructurales Objeto-Relacional
- Patrones de Mapeo de Metadatos Objeto-Relacional



PATRONES – DESCRIPCION 1

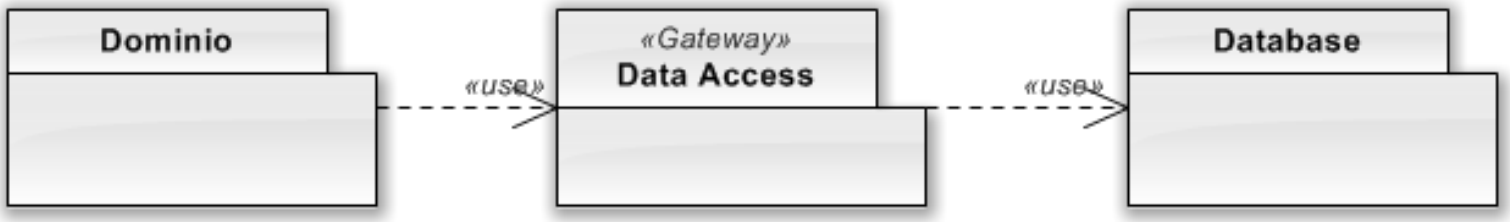
TIPO PATRON	OBJETIVO	PATRONES
Patrones Arquitectónicos	¿Cómo la logica del dominio se comunica con la base de datos?	○ Table Data Gateway ○ Row Data Gateway ○ Active Record ○ Data Mapper
Patrones de Comportamiento Objeto-Relacional	¿Cómo realizar el manejo de los objetos (cargarlos/guardarlos)?	○ Unit of Work ○ Identity Map ○ Lazy Load

PATRONES – DESCRIPCION 2

TIPO PATRON	OBJETIVO	PATRONES
Patrones Estructurales Objeto-Relacional	¿Cómo mapear conceptos de orientación a objetos a una base de datos relacional?	○ Identity Field ○ Foreign Key Mapping ○ Association Table Mapping ○ Dependent Mapping ○ Embedded Value ○ Serialized LOB ○ Inheritance Patterns ○ Inheritance Mapper
Patrones de Mapeo de Metadatos Objeto-Relacional	¿Cómo definir declarativamente el mapeo objeto-relacional y encapsularlo, proponiendo una interface orientada a objetos?	○ Metadata Mapping ○ Query Object ○ Repository

1 – TABLE DATA GATEWAY

DESCRIPCION – Un objeto que actúa como Gateway a una tabla de base de datos. Una instancia maneja todas las filas en la tabla



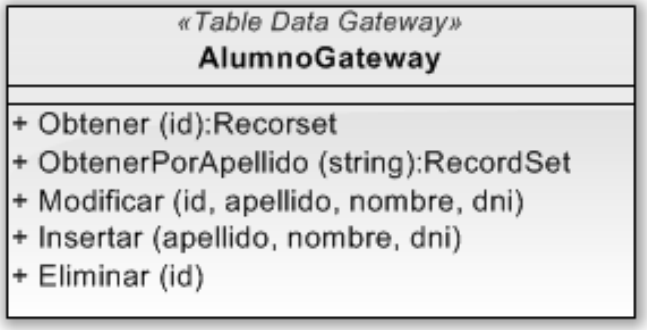


TABLE DATA GATEWAY – CARACTERISTICAS

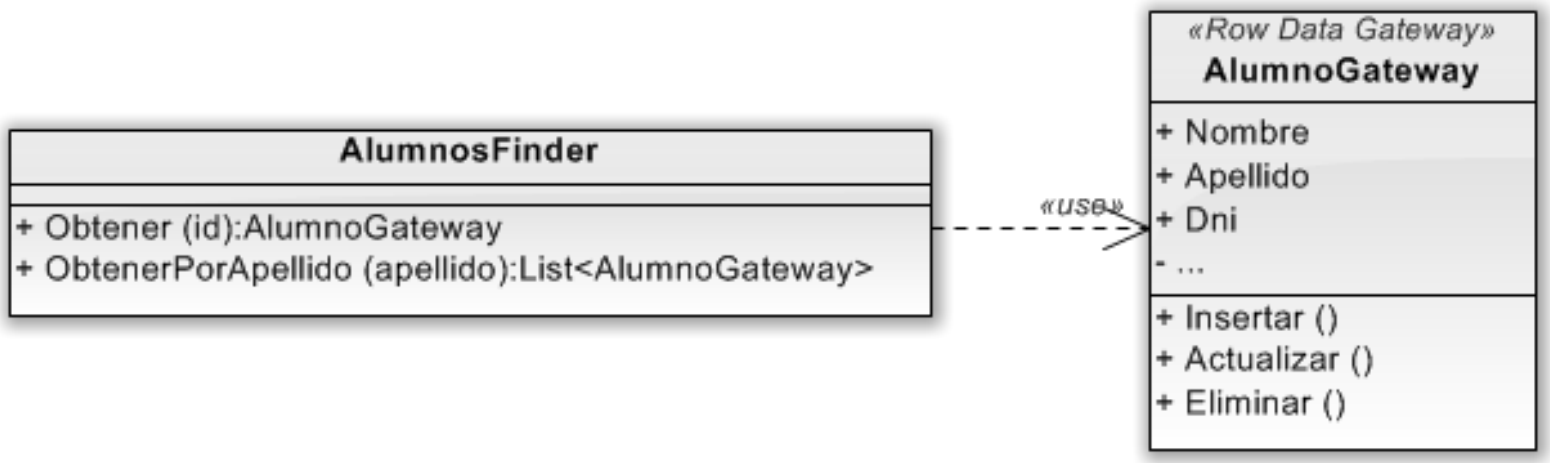
- Su interfaz consiste de:
  - Métodos *find* para obtener información desde la BD
  - Métodos de actualización: *insert*, *update*, *delete*
- cada método ejecuta una sentencia SQL de acuerdo a sus parámetros
- generalmente es stateless, ya que su rol es transferir datos entre el dominio y la base de datos
- en las consultas, siempre retorna un conjunto de datos: Map, RecordSet, Data Transfer Objetcs (DTOs)

TABLE DATA GATEWAY – UTILIZACION

- Dado que su implementación es muy sencilla, sería apropiado para aplicaciones muy pequeñas que no cuentan con una lógica de acceso a datos compleja
- Si se ha decidido utilizar Table Modules para resolver la lógica de domino. Table Data Gateway es el socio indicado para estos casos, ya que produce los record sets que requiere un Table Module
- También es apropiado para Transaction Scripts, siempre y cuando el result set sea conveniente para el Transaction Script
- La misma interfaz sirve para utilizar sentencias SQL directas o stored procedures
- El resto de los casos

2 – ROW DATA GATEWAY

DESCRIPCION – Un objeto que actúa como Gateway a un único registro en una tabla de base de datos. Hay una instancia por fila.

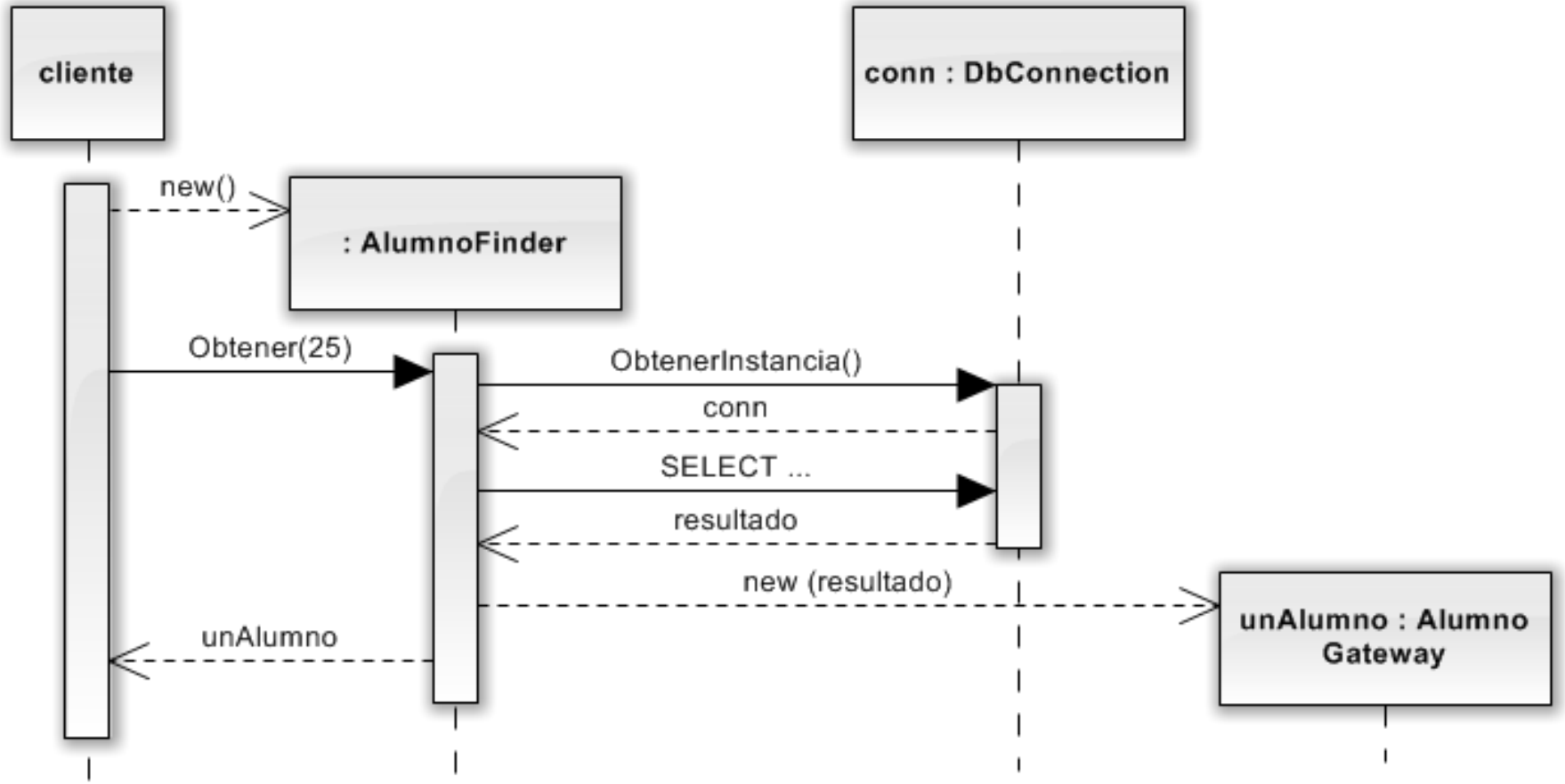


Un Row Data Gateway permite obtener objetos que lucen como los registros de una tabla, pero pueden ser accedidos a través de las facilidades del lenguaje de programación utilizado. De esta manera, se encapsula toda la lógica específica (generalmente SQL) de acceso a datos dentro del gateway.

ROW DATA GATEWAY – CARACTERISTICAS

- imita exactamente un registro de la base de datos, de manera tal que cada columna de la tabla es un campo del objeto
- realiza todas las conversiones necesarias entre los tipos de datos de la base de datos y los tipos del lenguaje de programación
- trabaja particularmente bien con Transaction Script
- métodos de búsqueda:
  - métodos estáticos
  - objetos de búsqueda (buscadores)
- si bien es bastante tedioso de escribir, es muy utilizado por los generadores de código basados en Metadata Mapping

ROW DATA GATEWAY – FUNCIONAMIENTO



Funcionamiento del Row Data Gateway

ROW DATA GATEWAY – UTILIZACION

- 👍 Principalmente, cuando se usa Transaction Script. Así, se factoriza el código de acceso a los datos y se lo reutiliza entre distintos scripts.

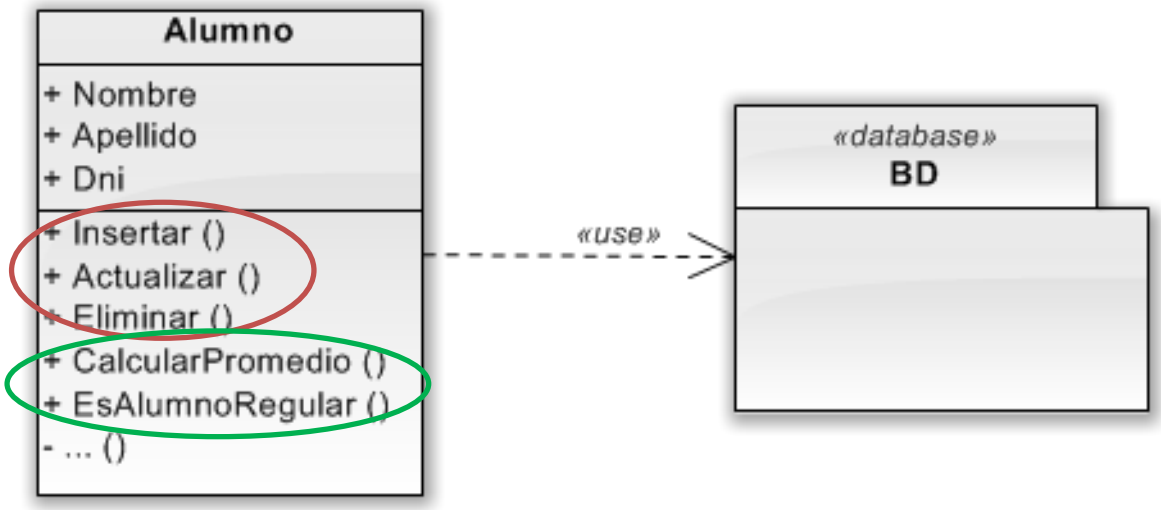
- 👎 No sería conveniente utilizarlo en conjunto con Domain Model.

Opciones:

- Si el mapeo es simple, usar Active Record
- Si el mapeo es más complejo, usar Data Mapper
- Si se utiliza para aislar a los objetos de domino de la estructura de base de datos, se tendrán tres representaciones de los datos: 1) una en los objetos de dominio, 2) una en los Row Data Gateways y 3) una en la base de datos

3 – ACTIVE RECORD

DESCRIPCION – Un objeto que wrappea una fila de una tabla de la base de datos, encapsula la lógica de acceso a datos y agrega lógica de dominio sobre aquellos datos.



Active Record utiliza la opción más obvia - pone la lógica de acceso a datos en el objeto de dominio. De esta manera, todos los objetos de dominio conocen como guardar y leer sus datos.

ACTIVE RECORD – CARACTERISTICAS 1

- La esencia de un Active Record es un Domain Model en el cual las clases se corresponden con la estructura de tablas de la base de datos subyacente.
- Cada Active Record es responsable tanto de guardarse y cargarse en/desde la base de datos, como de cualquier lógica de dominio que actúe sobre los datos.
- Típicamente tiene métodos para:
  - construir una instancia del Active Record a partir de una fila de un result set SQL
  - construir una nueva instancia que luego será insertada en la base de datos
  - métodos de búsqueda estáticos para factorizar consultas SQL repetitivas y retornar instancias del Active Record
  - insertar y actualizar los datos del Active Record en la base de datos
  - obtener y asignar valores a los campos del Active Record
  - implementar algunas partes de la lógica de negocio.

ACTIVE RECORD – CARACTERISTICAS 2

- Los métodos get y set pueden realizar otras funciones como, por ejemplo, convertir tipos orientados a SQL a tipos propios del lenguaje de programación.
- Se diferencia de Row Data Gateway en que Active Record contiene lógica de negocio. ¿Cuál es la diferencia entre Row Data Gateway y Active Record? La clave está en si existe lógica de dominio en estos objetos

  ✓ Row Data Gateway → sin lógica de dominio
  ✓ Active Record → con lógica de dominio

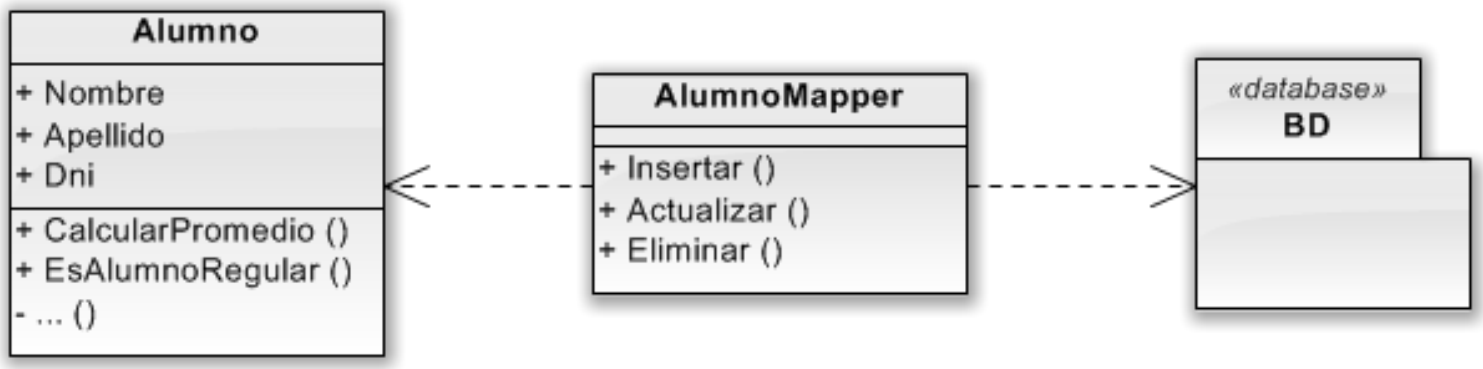
- Puede utilizar métodos de búsqueda estáticos o puede utilizar objetos de búsqueda (finders).

ACTIVE RECORD – UTILIZACION

- Es una buena elección para la lógica de dominio que no sea demasiado compleja, tales como operaciones CRUD. También es aconsejable para cuando se necesitan validaciones basadas en los datos de un único registro.
- Tiene la ventaja de la simplicidad comparado con Data Mapper.
- Funciona bien solamente si los objetos Active Record se corresponden directamente con las tablas de la base de datos.
- Es un buen patrón a considerar si se está usando Transaction Script y se comienzan a sufrir sus desventajas (duplicación de código, mantenimiento, etc).
- Si la lógica de dominio es más compleja, se querrán usar relaciones directas a objetos, colecciones, herencia, etc.; cosas que no son fáciles de resolver con Active Record.
- Acopla el diseño de objetos al diseño de base de datos.

4 – DATA MAPPER

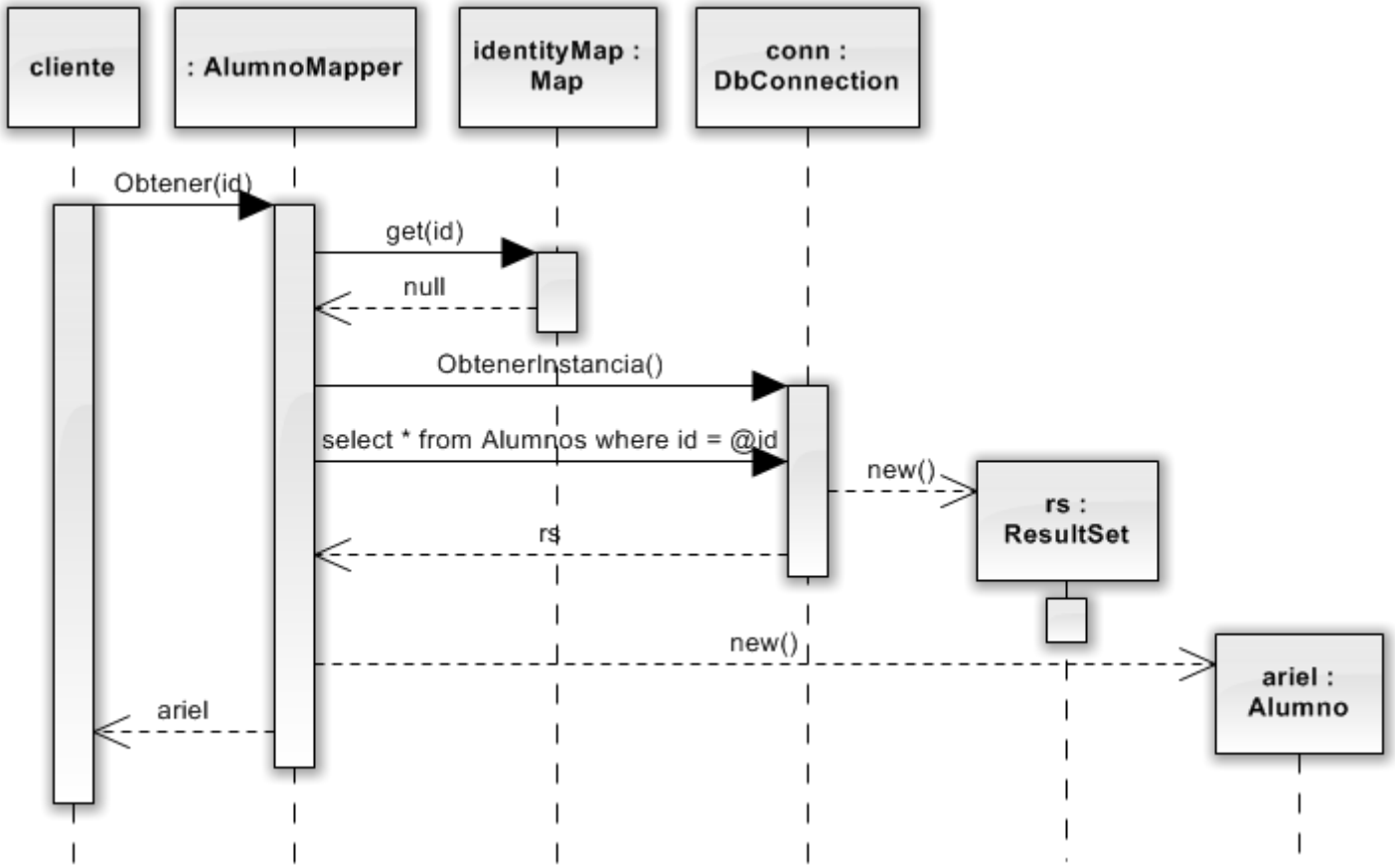
DESCRIPCION – Una capa de Mappers que mueven datos entre los objetos y la base de datos manteniéndolos independientes unos de otros y del mapper en sí mismo.



Los Data Mappers separan los objetos en memoria de la base de datos. Son responsables de transferir los datos entre ambos y de mantenerlos aislados uno de otro.

DATA MAPPER – CARACTERISTICAS 1

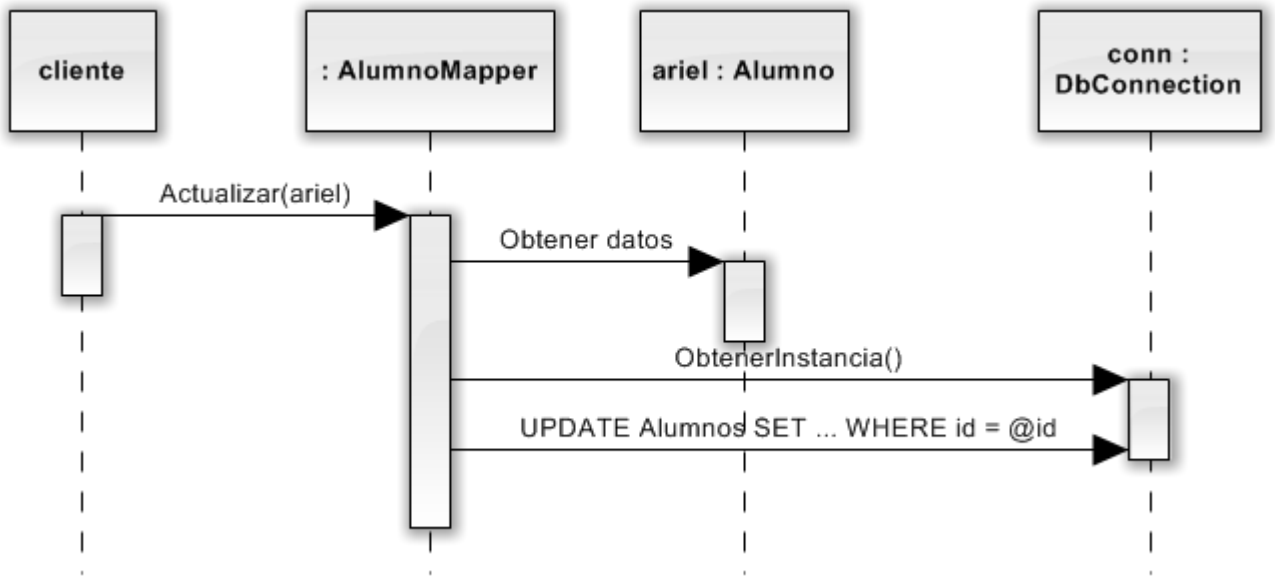
Hay muchas formas de construir una capa de Data Mappers. Un ejemplo sencillo:



Recuperar datos de una base de datos

DATA MAPPER – CARACTERISTICAS 2

Hay muchas formas de construir una capa de Data Mappers. Un ejemplo sencillo:



Actualizando datos

Cantidad de Mappers

- Hardcoded - uno por cada clase de dominio o raíz de un agregado.
- Metadata Mapping - unico mapper para todas las clases. Finders agrupados por clase de dominio

DATA MAPPER – IDENTIDAD DE LOS OBJETOS

- Es necesario que exista en todo momento una única representación en memoria de un objeto de domino, identificada por su identidad

- Hay dos opciones:

  - Registry de Identity Maps

  - Cada clase finder mantenga su propio Identity Map, asegurando que habrá solamente una clase finder por cada entidad de negocio.

DATA MAPPER – FINDERS

Manejo de Finders

- varias veces una clase de dominio necesita acceder a un finder de otra clase de dominio
- nunca agregar una dependencia desde un objeto de dominio al Data Mapper
- utilizar Separated Interface, creando una interfaz con todos los métodos finders requeridos por la clase de domino y ubicándola en el paquete de dominio

DATA MAPPER – MAPEO DE DATOS

Mapear datos a propiedades del dominio

(Se supone que no se utilizarán variables de instancia públicas)

- Los mappers necesitan acceder a las variables de instancia de los objetos de dominio. Esto sería un problema ya que debiéramos exponer miembros públicos para los mappers, aun si no queremos que formen parte de la lógica de domino.

- Algunas soluciones
  - poner los mappers cerca de los objetos de dominio para sacar provecho de alguna regla de visibilidad del lenguaje de programación (no óptima)
  - usar Reflection - puede afectar la performance, aunque muchas veces es despreciable con respecto al tiempo de acceso a la base de datos
  - utilizar métodos especiales, nombrados con alguna convención que permita identificarlos.

DATA MAPPER – MAPEO DE DATOS

Mapeos basados en Metadatos

- Código explícito
  - Implica tener un mapper por cada objeto de negocio.
  - El mapeo se realiza a través de asignaciones y sentencias SQL que mantiene internamente

- Metadata Mapping
  - Mantiene los metadatos de mapeo como datos, ya sea en la misma clase de dominio o en una clase separada
  - Existen distintas variantes para metadatos: XML, atributos/anotaciones, fluent interfaces, modelos gráficos.

Ejemplos

- Java - JPA, Hibernate, EJB, MyBatis, TopLink
- .Net - Entity Framework, Nhibernate, Linq to SQL
- PHP - CakePHP, Propel, Doctrine
- Android - OrmLite, SugarORM

DATA MAPPER – UTILIZACION

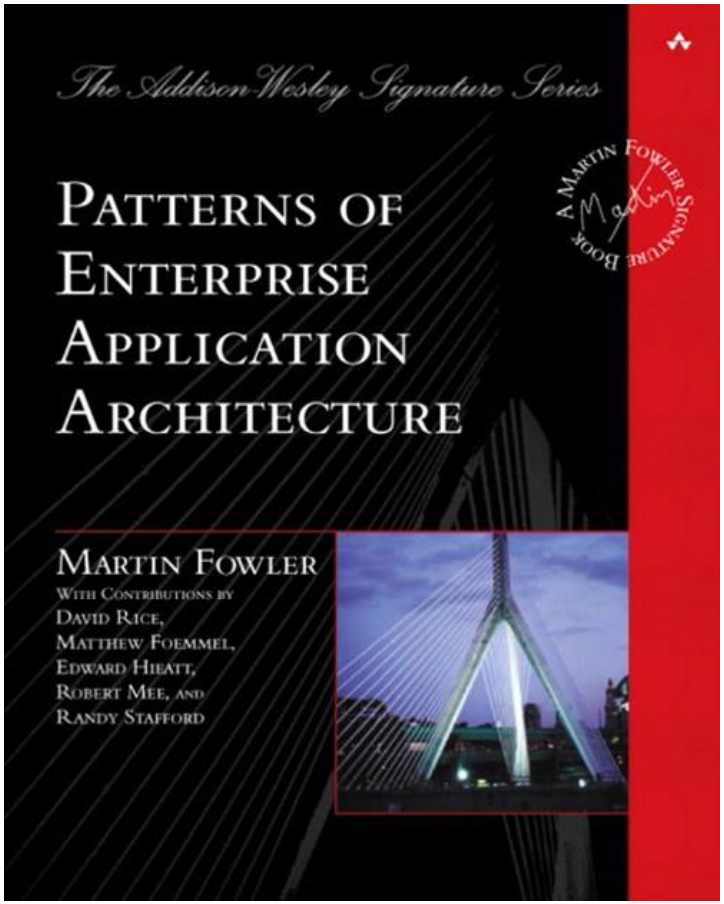
- cuando queremos que el esquema de base de datos y el modelo de dominio evolucionen independientemente
- cuando utilizamos Domain Model. Si bien se podría utilizar Active Record, cuando las cosas se ponen más complejas, Data Mapper toma ventaja.
- si se tiene un modelo de negocio simple
- si no se quiere pagar el costo de una capa extra
- si hay que construirlo desde cero

DATA MAPPER – COMENTARIOS

Un data mapper es complicado de construir.

Hay productos que ya resuelven la problemática, están testeados e implementan muchos de los patrones vistos.

BIBLIOGRAFIA



Patterns of Enterprise Application Architecture (2002)

Martin Fowler
www.martinfowler.com

Elsa Estevez
ece@cs.uns.edu.ar