

Capa de Negocio - Modelo de Dominio

Pablo Sánchez

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria
Santander (Cantabria, España)
p.sanchez@unican.es



Advertencia

Todo el material contenido en este documento no constituye en modo alguno una obra de referencia o apuntes oficiales mediante el cual se puedan preparar las pruebas evaluables necesarias para superar la asignatura.

Este documento contiene exclusivamente una serie de diapositivas cuyo objetivo es servir de complemento visual a las actividades realizadas en el aula para la transmisión del contenido sobre el cual versarán las mencionadas pruebas evaluables.

Dicho de forma más clara, **estas transparencias no son apuntes y su objetivo no es servir para que el alumno pueda preparar la asignatura.**


Índice


- 1 **Introducción**
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
- 5 Sumario


Objetivos del Tema

- ➊ Comprender en profundidad cuáles son las responsabilidades de la capa de negocio.
- ➋ Conocer los principales patrones relacionados con el desarrollo de una capa de negocio.
- ➌ Conocer las ventajas e inconvenientes del patrón *Domain Model*.
- ➍ Ser capaz de aplicar el patrón *Domain Model*.
- ➎ Ser capaz de aplicar los principios de *Domain-Driven Design*.

Bibliografía

 Fowler, M. (2002).
Patterns of Enterprise Application Architecture.
Addison-Wesley.

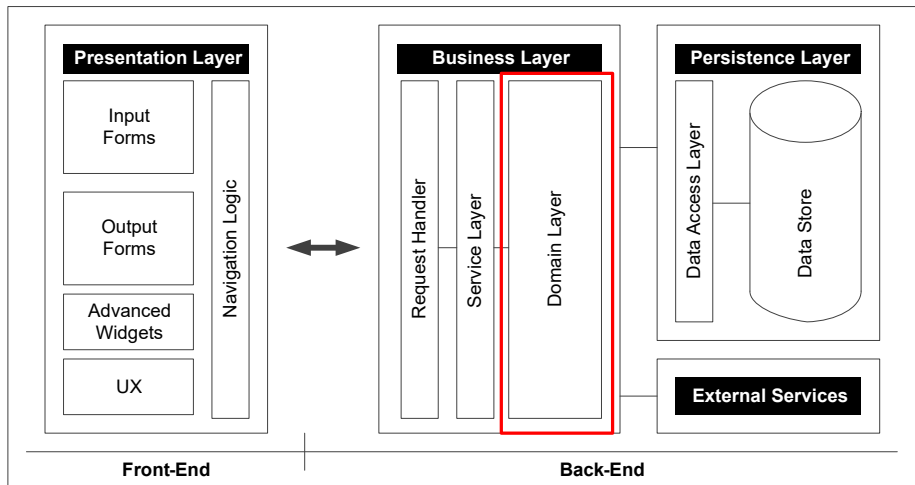
 Evans, E. J. (2003).
Domain-Driven Design.
Addison-Wesley.

 Esposito, D. and Saltarello, A. (2014).
Microsoft .NET - Architecting Applications for the Enterprise.
Microsoft Press. 2ª Edición.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
- 5 Sumario

Capa de Dominio



Responsabilidades de la Capa de Negocio

- 1 Atender las peticiones de los clientes.
- 2 Asegurar el cumplimiento de **reglas de negocio** existentes.
- 3 Asegurar la **transaccionalidad** de las operaciones de negocio.
- 4 Validar las peticiones de los clientes.
- 5 Recuperar y almacenar datos del almacén o almacenes persistentes.
- 6 Facilitar la eficiencia del sistema.
- 7 Controlar el acceso a los datos.
- 8 Gestionar la comunicación con los servicios externos.
- 9 Ejecutar operaciones (periódicas) del sistema.
- 10 Gestionar de manera adecuada casos excepcionales.
- 11 Ayudar a satisfacer los requisitos no funcionales.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
- 5 Sumario

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
 - Problema Común
 - Table Module
 - Transaction Script
 - Domain Model + Service Layer
- 4 Domain-Driven Design
- 5 Sumario

Problema Común a los Patrones de la Capa de Negocio

Problema Común

El problema común a los patrones de la capa de negocio es donde colocar la lógica de negocio de manera que se satisfagan las responsabilidades de la capa de negocio.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
 - Problema Común
 - **Table Module**
 - Transaction Script
 - Domain Model + Service Layer
- 4 Domain-Driven Design
- 5 Sumario

Table Module (obsoleto)

Solución Table Module

Crear una clase que gestione la lógica de negocio de una tabla (o vista) completa. Conocido también como el *Smart UI Antipattern*.

Table Module (obsoleto)

Ventajas

- 1 Facilidad de uso en lenguajes 4GL.

Desventajas

- 1 Manipula mal instancias en solitario.
- 2 Dificulta la gestión de datos residentes en varias tablas.
- 3 Genera problemas de integración con otras aplicaciones.
- 4 No utiliza orientación a objetos.
- 5 Crea rápidamente problemas de redundancia en la lógica de negocio.
- 6 Genera problemas con reglas de negocio complejas.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
 - Problema Común
 - Table Module
 - **Transaction Script**
 - Domain Model + Service Layer
- 4 Domain-Driven Design
- 5 Sumario

Transaction Script (obsoleto)

Solución Transaction Script

Crear un conjunto de funciones que respondan a los eventos que se puedan generar desde la interfaz de usuario, ejecutando para ello la lógica de negocio que sea necesaria.

Transaction Script (obsoleto)

Ventajas

- 1 Facilidad de implementación en aplicaciones sencillas.

Desventajas

- 1 No utiliza orientación a objetos.
- 2 Crea rápidamente problemas de redundancia en la lógica de negocio.
- 3 Genera problemas con reglas de negocio complejas.
- 4 Genera problemas de evolución.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
 - Problema Común
 - Table Module
 - Transaction Script
 - Domain Model + Service Layer
- 4 Domain-Driven Design
- 5 Sumario

Domain Model

Solución Domain Model

Modela el dominio del problema utilizando orientación a objetos y distribuye las reglas de negocio de manera adecuada entre las clases de dominio que corresponda.

Domain Model

Ventajas

- 1 Orientado a objetos.
- 2 Permite la utilización de patrones de diseño.
- 3 Reduce la redundancia de código.
- 4 Soporta mejor las reglas de negocio complejas.
- 5 Ofrece una mayor facilidad de evolución.

Desventajas

- 1 Impedancia objeto-relacional u objeto-xxx.
- 2 Mayor complejidad de diseño e implementación.

Service Layer

Solución Domain Model

Aislar al modelo de dominio de otras capas y aplicaciones mediante la creación de una capa de servicio que:

- 1 Especifique las operaciones a las cuales el modelo de dominio es capaz de responder.
- 2 Permita redirigir peticiones a las operaciones del modelo de dominio que corresponda, devolviendo la respuesta que corresponda.
- 3 Permite encapsular la *lógica de la aplicación* o *workflow*.

Service Layer

Lógica de la Dominio

La *lógica de dominio* es el conjunto de procedimientos, restricciones y normas que rigen el funcionamiento de una determinada organización o dominio, y que son independientes de la existencia o no de una aplicación software.

Lógica de la Aplicación

La *lógica de la aplicación* es el conjunto de procedimientos, restricciones y normas que rigen el funcionamiento de una determinada aplicación. Es la encargada de gestionar elementos como transacciones, accesos a almacenes persistentes o seguridad.

Service Layer

- 1 Aisla al modelo de dominio de servicios concretos.
- 2 Favorece la gestión de la lógica de la aplicación.
- 3 Favorece la gestión de la concurrencia.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 **Domain-Driven Design**
- 5 Sumario

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
 - **Introducción**
 - Entities
 - Value Objects
 - Services
 - Aggregates
 - Repositories
 - Otros elementos DDD
- 5 Sumario

Domain-Driven Design

Domain-Driven Design

Domain-Driven Design es una técnica de desarrollo sw donde todo el diseño de un producto sw gira en torno a un elemento central y fundamental que es el *modelo de dominio*, el cual captura el dominio y la lógica de negocio de dicho producto sw.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
 - Introducción
 - **Entities**
 - Value Objects
 - Services
 - Aggregates
 - Repositories
 - Otros elementos DDD
- 5 Sumario

Entities

Entity

Una *entity* es un objeto del dominio con una identidad y un ciclo de vida que debe ser reconocido y monitorizados.

Características de un Identificador

- 1 Únicos para cada objeto e inmutables.
- 2 Pueden ser un único atributo o combinación de atributos.
- 3 Si no se encuentra un identificador natural, puede ser generado por la aplicación.
- 4 Si son generados, se esconden (normalmente) al usuario.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
 - Introducción
 - Entities
 - Value Objects
 - Services
 - Aggregates
 - Repositories
 - Otros elementos DDD
- 5 Sumario

Value Objects

Value Object

Elementos del dominio sin identidad definida, cuya existencia no es necesario monitorizar, y que simplemente representan valores de alguna propiedad de una entidad.

Características de un *Value Object*

- 1 Dos instancias con los mismos valores se consideran idénticas, aunque tengan identidades distintas.
- 2 Se recomienda hacer los *value objects* inmutables.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
 - Introducción
 - Entities
 - Value Objects
 - **Services**
 - Aggregates
 - Repositories
 - Otros elementos DDD
- 5 Sumario

Services

Service

Un *service* encapsula operaciones de dominio que no son responsabilidad natural de ninguna *entity* o *value object*.

Características de un *Service*

- 1 Forma parte del *lenguaje universal* del dominio.
- 2 Su interfaz está definida en base a otros elementos del modelo de dominio.
- 3 Su operación u operaciones no tienen estado (*stateless*).

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
 - Introducción
 - Entities
 - Value Objects
 - Services
 - **Aggregates**
 - Repositories
 - Otros elementos DDD
- 5 Sumario

Aggregates

Aggregate

Un *aggregate* es un conjunto cohesionado de *entities* y *value objects* con una frontera clara con el resto del modelo de dominio y un conjunto de invariantes bien definidos que deben preservarse dentro de dicho conjunto.

Aggregate Root

Dentro de un *aggregate*, el *aggregate root* es una *entity* que controla el ciclo de vida del resto de los elementos del *aggregate* y que contiene la información necesaria para encargarse de la preservación de los invariantes.

Características de los *Aggregates*

- 1 El *aggregate root* debe tener identidad global.
- 2 El resto de elementos del *aggregate* sólo necesita tener identidad local.
- 3 El *aggregate root* es el encargado de preservar los invariantes.
- 4 El tiempo de vida de un *aggregate* debe ser uniforme a sus miembros.
- 5 Los elementos externos al *aggregate* sólo deben referenciar al *aggregate root*.
- 6 Los elementos internos de un *aggregate* sólo pueden referenciar como elementos externos *aggregate roots*.
- 7 Los elementos internos de un *aggregate* sólo deben utilizarse de manera transitoria fuera del *aggregate*.
- 8 Sólo los *aggregate roots* pueden ser recuperados de persistencia.
- 9 Si se elimina un *aggregate root*, se debe eliminar todo el *aggregate*.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
 - Introducción
 - Entities
 - Value Objects
 - Services
 - Aggregates
 - **Repositories**
 - Otros elementos DDD
- 5 Sumario

Repositories

Repository

Un *repository* representa una clase que, a nivel conceptual, representa una colección con todas las instancias de un determinado elemento del dominio. El repositorio debe además proporcionar métodos para recuperar conjuntos de instancias concretas dentro de dicha colección, añadir nuevas instancias o eliminarlas.

Características de los *Repositories*

- 1 Sólo existirán *repositories* para los *aggregates root*.
- 2 Normalmente, existirá un *repository* por cada *aggregate root*.
- 3 Cada *repository* contendrá las operaciones CRUD básicas, más un método para recuperar todas las instancias de un elemento del dominio.
- 4 Además, cada *repository* contendrá otros métodos de búsqueda o cálculo siempre y cuando sean necesarios.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
 - Introducción
 - Entities
 - Value Objects
 - Services
 - Aggregates
 - Repositories
 - Otros elementos DDD
- 5 Sumario

Otros elementos DDD

- ❶ Factories.
- ❷ Domain Events.
- ❸ Modules.

Índice

- 1 Introducción
- 2 Capa de Negocio
- 3 Patrones para la Capa de Negocio
- 4 Domain-Driven Design
- 5 Sumario

¿Qué Tengo que Saber de Todo Esto?

- 1 Comprender las responsabilidades de la capa de negocio.
- 2 Conocer cómo funcionan *Table Module* y *Transaction Script*.
- 3 Comprender por qué *Table Module* y *Transaction Script* se usan muy poco.
- 4 Comprender cómo funciona *Domain Model* y *Service Layer*.
- 5 Comprender los principios de *Domain-Driven Design (DDD)*
- 6 Ser capaz de crear modelos de dominio conforme a los principios *DDD*.