Arquitectura Software
PEAA: Organización Lógica Dominio

Diseño de Sistemas Software

Carlos Pérez
Cristina Cachero

Lógica Dominio

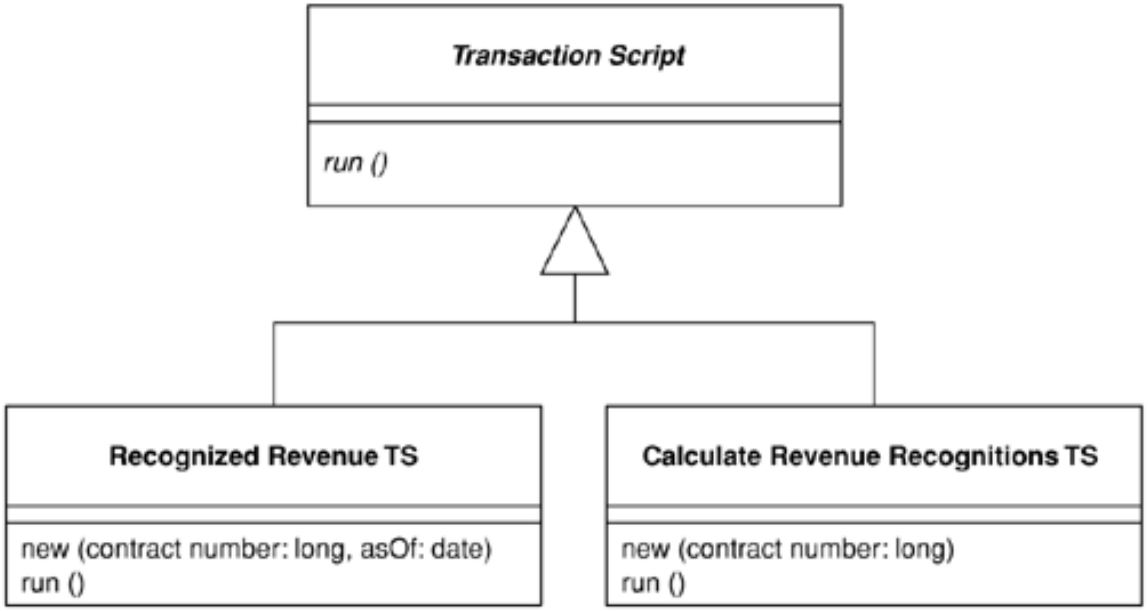
- Tres patrones:
  - Transaction Script: organiza la lógica de negocio en procedimientos, donde cada procedimiento maneja una sola petición de la presentación. (No necesariamente OO)
  - Domain Model: modelo de objetos del dominio que incorpora tanto comportamiento como datos
  - Table Module: una sola instancia que maneja la lógica de negocio para todas las filas en una tabla o vista de la BD

Lógica Dominio: Transaction Script

- Su nombre viene de que la mayoría de las veces tendrás un script de transacción para cada transacción de BD (aunque no siempre!)
- Procedimiento que:
  - toma la entra de la presentación
  - la procesa (validaciones, cálculos)
  - guarda datos en la BD
  - invoca operaciones de otros sistemas.
  - Devuelve datos a la presentación (cálculos para organizar y formatear respuesta)
- Un solo procedimiento para cada acción ejecutable por un usuario (transacción de negocio)
  - Cada paso puede ir separado en subrutinas, y las subrutinas pueden ser compartidas entre distintos Transaction Scripts.
  - Sin embargo, la fuerza siguie siendo un procedimiento para cada acción
  - Ejemplo: si necesitas reservar una habitación de hotel, la lógica para comprobar disoponibilidad, calcular precios y actualizar la BD se encontrará dentro de un procedimiento llamado ReservaHabitacionHotel().

Lógica Dominio: Transaction Script

- En el paradigma OO, ¿cómo organizo mis transaction scripts en clases? (puede ayudar en caso de necesitar manejar concurrencia)
- Dos opciones:
  - Varios transaction scripts relacionados en una sola clase.
  - Cada transaction script en su propia clase: ¿Patrón GOF?



Lógica Dominio: Transaction Script

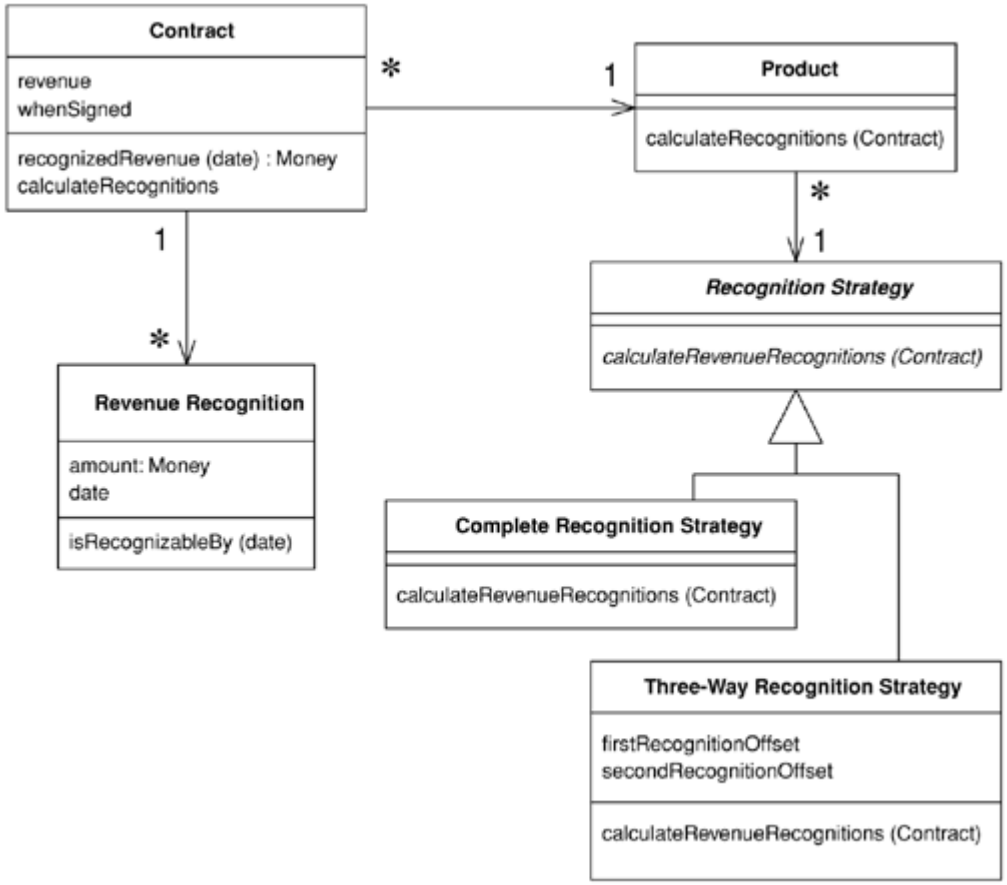
- Ventajas:
  - Simple de enteder: no necesitas preocuparte de lo que hacen otras transacciones
  - Funciona bien con capa sencilla de acceso a datos (thin DB wrapper) mediante el uso de patrones Row Data Gateway o Table Data Gateway.
  - Las fronteras de la transacción son obvias: Comienza el script abriendo la transacción y finaliza cerrándola. Son fácilmente manejables por las herramientas de desarrollo.

Lógica Dominio: Transaction Script

- Desventajas:
  - Según aumenta la lógica de negocio puede aparecer duplicación de código a medida que diferentes transacciones necesitan hacer cosas parecidas
    - Parte de esta duplicación se puede eliminar factorizando subrutinas comunes, pero esto no siempre es evidente ni fácil de hacer -> resultado es un conjunto de rutinas sin estructura clara, difíciles de mantener.

Lógica Dominio: Domain Model

- La solución ante el aumento de complejidad de la lógica de negocio es introducir un Modelo de Dominio
  - Red de objetos interconectados donde cada objeto representa algún individuo significativo del dominio



Lógica Dominio: Domain Model

- Insertas una capa de objetos que modelan el área de negocio en la que estás trabajando, donde algunos objetos representarán datos y otros representarán procesos.
- Un modelo de dominio a menudo se parece a un modelo de base de datos ¿Diferencias?
- Dos estilos de modelo de dominio:
  - Modelo parecido a diseño de BD donde cada tabla tiene asociado un objeto de dominio. Se puede conectar a BD con un simple patrón Active Record
  - Modelo de dominio enriquecido con herencia, estrategias, redes complejas de objetos interconectados, … Mejor para lógica compleja, pero más difícil de mapear a BD. Requiere para su conexión a BD un patrón Data Mapper .
- Requiere mínimo acoplamiento entre ésta y otras capas del sistema: una fuerza que guía la división en capas es mantener tan pocas dependencias como sea posible entre la capa de dominio y el resto de capas.

Lógica Dominio: Domain Model

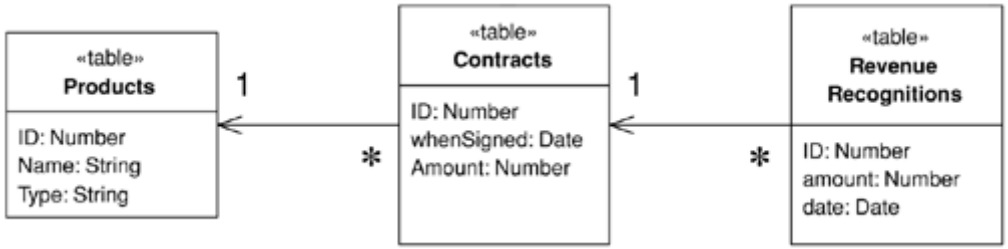
- Diferentes ámbitos:
  - Aplicación mono-usuario donde todos los objetos conviven en memoria
    - Problema: consumo de memoria, tiempo de inicialización.
  - Alternativa: tener en cada momento en memoria solo los objetos necesarios (gráfico de objetos)
    - BD OO -> hace el trabajo por ti.
    - BD relacional: debes hacerlo tú mismo de manera programática. Si hay que utilizar el mismo gráfico entre llamadas al servidor, habrá que guardar el estado del servidor

Lógica Dominio: Table Module

- Problema del modelo de dominio:
  - Interfaz con BD relacionales (dos representaciones distintas de los datos).
- Un Módulo de Tabla organiza la lógica de dominio con una clase por tabla en la BD, y una sola instancia de una clase contiene los distintos procedimientos que actuarán sobre los datos.
- ¿Diferencia primordial entre Módulo de Tabla y Modelo de Dominio a la hora de la instanciación?
  - Si tienes muchos pedidos, el modelo de dominio generará un objeto de dominio por pedido, mientras que un Módulo de Tabla tendrá un objeto para manejar todos los pedidos (no tiene noción de identidad de objeto).
    - anEmployeeModule.getAddress(**long employeeID).**
- A menudo se utiliza un módulo de tabla con una estructura de apoyo (normalmente un Record Set, resultado de una llamada SQL)
  - El objeto módulo de tabla proporciona una interfaz para actuar sobre esos datos (comportamiento asociado a los datos)-> Beneficios de la encapsulación

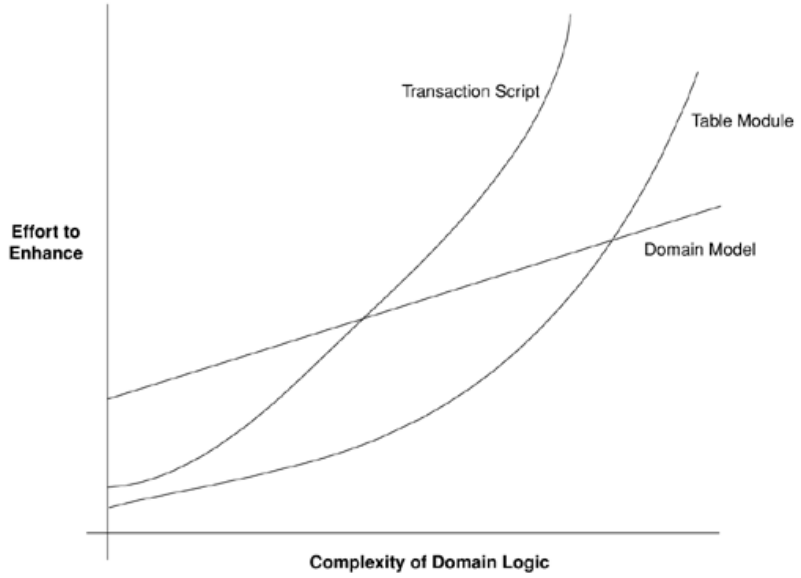
Lógica Dominio: Table Module

- Lo usual es tener un módulo de tabla por cada tabla en la BD, aunque tb pueden definirse módulos de tabla para vistas, consultas, …
- Puede implementarse como una instancia (objeto) o como un conjunto de métodos estáticos
  - Como instancia puedes asociarla a un record set específico, realizar herencia, …



Lógica Dominio: Table Module

- Cuándo utilizar (trade-off)
  - Table Module: cuando trabajamos con record sets-> facilita la integración con BD
  - Domain model: cuando la lógica es complicada, cuando la estructura de objetos se parece mucho a la estructura de la BD (junto con Active Record).
  - the better the team is, the more I'm inclined to use a Domain Model



Lógica Dominio: Capa de Servicio

- Sirve para separar la lógica de dominio (e.g. políticas de cálculo de pagos) de la lógica de aplicación (e.g. avisar a una aplicación externa de que se ha producido un pago)
  - Ponerlo todo en la clase de dominio perjudica la reusabilidad de los objetos de dominio.
  - La solución es poner cada tipo de lógica de negocio en una capa distinta.

Lógica Dominio: Capa de Servicio

- Implementación de capa de servicio:
  - Aproximación 'Fachada de dominio': conjunto de fachadas ligeras definidas sobre el modelo de dominio. Esas clases no implementan lógica, sino que establecen el conjunto de operaciones a través de las cuales los clientes interaccionan con la aplicación
  - Aproximación 'script de operación': la capa de servicio se implementa como clases más pesadas que directamente implementan la lógica de aplicación pero delegan la implementación de la lógica de dominio a clases de dominio. Las operaciones serán scripts, que se agruparán en clases relacionadas con áreas de la aplicación (servicio de aplicación).
    - La capa de servicio se compone de esas clases de servicio de aplicación, que deberían extender una Layer Supertype que abstraiga sus responsabilidades y comportamientos comunes.

Lógica Dominio: Capa de Servicio

- Cuándo dividir la capa de lógica de negocio en dos capas:
  - Cuando tienes aplicaciones con más de un cliente, respuestas complejas en sus casos de uso que involucran recursos transaccionales, etc.
  - La capa de servicio también es un buen lugar para ubicar control de transacciones y seguridad.
- La capa de servicio combina *scripting* y modelo de dominio

Bibliografía

- [Fowler03] Fowler, Martin: Patterns of Enterprise Application Architecture, Addison Wesley, 2003.