Reading P of EAA (2)
Chapter2: Organizing Domain Logic

TAKAI Naoto <takai@commentout.com>

Three Patterns

- *Transaction Script*
  - The simplest approach
- *Domain Model*
  - The object-oriented way.
- *Table Module*
  - A middle ground between a *Transaction Script* and a *Domain Model*.

Transaction Script

- Advantages
  - Simple procedural model.
  - Woks well with a *Row Data Gateway* or *Table Data Gateway*.
  - Obvious how to set the transaction boundaries.
- Disadvantages
  - When complexity of the domain logic increases, being a tangled web of routines without a clear structure.

Domain Model

- Object-oriented way
  - Many techniques to handle increasingly complex logic.
  - *Domain Model* as opposed to a *Transaction Script* is essence of the paradigm shift.
  - It takes time for people new to object models.
- Database mapping
  - The richer *Domain Model*, the more complex database mapping.
  - Usually with *Data Mapper*.

Table Module

- Work with a *Record Set*
  - Many GUI environments are built to work on a results of a SQL query organized in a *Record Set*.
- Middle ground between a *Transaction Script* and a *Domain Model*
  - *Table Module* provides more structure and makes it easier to find and remove duplication.
  - Inheritance, strategies, and other OO patterns can't be used.

Making a Choice (1)

- How do you choose between the tree pattern?
  - Much depends on how complex domain logic is.
  - Nobody knows how to measure the complexity of domain logic.
  - A team that is familiar with *Domain Model* will lower the initial cost of using this pattern.
  - If you have an environment where tools work around a *Record Set*, then makes a *Table Module* much more attractive.

Making a Choice (2)

- Decision isn't completely cast in store
  - If you started with *Transaction Script,* don't hesitate to refactor to *Domain Model*.
  - If you start with *Domain Model*, going to *Transaction Script* is less worthwhile unless you can simplify your data source layer
- These three patterns are not mutually exclusive choices

Service Layer (1)

- Split the layer in two
  - With *Domain Model* or *Table Module*.
  - *Transaction Script* is not complex enough.
  - The presentation logic interacts with the domain through the *Service Layer*.
  - *Service Layer* is a good spot to place transaction control and security.

Service Layer (2)

- How much behavior to put in it?
  - The minimal case is to make the *Service Layer* a facade.
    - The *Service Layer* provides an API that is easier to use because it's typically oriented around use cases.
  - Most business logic is placed in *Transaction Scripts*.
    - The domain objects are very simple; if it's a *Domain Model*, you can use a simpler data source layer such as *Active Record*.
  - A more even mix of behavior: use-case controller
    - The use case controllers tend to encourage duplicate code.

Service Layer (3)

- You should not necessarily make a fixed layer of them.
  - Procedural service objects can sometimes be a very useful way to factor logic.
- My preference is to have the thinnest *Service Layer*.
  - However, I know many good designers who always use a *Service Layer* with a fair bit of logic.