SOFTWARE ENGINEERING

In DDD, is a Domain Service essentially just a Facade and/or Mediator Pattern?

Asked 2 years, 3 months ago Active 2 years, 3 months ago Viewed 3k times

12

2

In Domain Driven Design, the Domain Layer can have several (traditional) services. For example, for the User domain, we may have:

- A UserFactory, that builds User objects in different ways
- A UserRepository, which is responsible for interacting with the Persistence Services in the Infrastructure Layer

Is a UserService in the Domain Layer simply a Mediator and/or Facade to those two services and the Infrastructure Layer, or is there more to it?

object-oriented object-oriented-design domain-driven-design

asked Dec 21 '17 at 3:33

e_i_pi

659 ● 6 ● 16

1 See also Services in DDD and Services in DDD – Erik Eidt Dec 21 '17 at 16:30

I've read the Level Gorodinski posts a great deal, never seen that second link though. Great read, definitely touches on some important points! –

e_i_pi Dec 21 '17 at 19:10

2 Answers

Active | Oldest | Votes

9

`Domain services` are best described by what they are not:

- they are neither `Entities` nor `Aggregate roots`
- they are not `Value objects`
- carry domain knowledge that doesn't naturally fit *only one* `Entity` or *one* `Value object`

An example of a `Domain service` is a `Saga/Process manager`: it coordinates a long running process involving multiple `Aggregate roots`, possible from different `Bounded contexts`.

That being said, *what* is a `Domain service` and *how* it is implemented are two orthogonal things.

> Is a UserService in the Domain Layer simply a Mediator and/or Facade to those two services and the Infrastructure Layer, or is there more to it?

Some domain services like a `UserRepository` (composed of an interface defined in the `Domain layer` and a concrete implementation in the `Infrastructure layer`) *can* be implemented using the `Facade` design pattern. Other domain services are not.

There is no hard rule about *how* to implement them, other than the important rule that the `Domain layer` must not depend on other layers (and S.O.L.I.D.).

answered Dec 21 '17 at 7:07

Constantin Galbenu
2,760 ● 4 ● 12

Thank you, I think I finally understand the Domain Layer. Along with holding the data objects (Aggregates, Entities and Value Objects) it also holds the business rules - but not the concrete implementation of those rules. Domain Services define what you can do to Domain data objects, but have no knowledge of how those operations function internally. – e_i_pi Dec 21 '17 at 10:59

1 @e_i_pi business rules are protected only by Aggregates and their nested entities. Domain services are not involved in this. – Constantin Galbenu Dec 21 '17 at 11:16

1 @e_i_pi where the operation involves more than one Aggregate. For example, given the list of BankAccounts (Aggregates) of a Person (another Aggregate) a domain service would compute the total balance of those accounts. – Constantin Galbenu Dec 21 '17 at 11:26

1 @e_i_pi: I think you have a few misconceptions. So, the whole of Domain Layer is a software model of your domain. You said - "Along with holding the data objects (Aggregates, Entities and Value Objects)" - these are not "data objects" in the sense that they just hold data; these actually implement domain rules, they define the behavior. Now, *Domain Services*, according to the DDD Book by E. Evans, are those aspects of the *domain* that don't fit naturally into an object (an entity or a value object). – Filip Milovanović Dec 21 '17 at 14:17

1 Rather, a *Domain Service* "is defined purely in terms of what it can do for a client", defined in terms of other elements of the domain model (so it orchestrates those elements somehow, and enforces domain rules that govern that orchestration). The domain service itself is stateless. There's also the concept of *Application Services*, which reside in a higher level layer, and essentially implement user stories, or equivalently use cases, by acting as an interface towards the domain layer. Note that the ratio of the objects vs services will vary depending on the domain being modeled. – Filip Milovanović Dec 21 '17 at 14:17

1

I see services in DDD as result of Dependency Inversion.

If you were to use "plain" dependencies, then your domain code would call database to save or query an entity, or factory, that creates an entity, that are tied to database or external service or some kind of other infrastructure code.

But that is not how domain code should be. Domain code should not depend on infrastructure code. As this dependency makes it harder to test and, possibly, reuse. Which is why you invert that dependency. You make infrastructure code depend on the domain code. And to do that, you need to introduce an abstraction. An abstraction that defines what behavior the domain code expects to be implemented by the infrastructure.

And services in DDD are that abstraction. In majority of cases, for domain code, those services should be plain interfaces. And the implementation should be in the infrastructure code, which has dependency on those interfaces.

answered Dec 21 '17 at 7:43

Euphoric

31.6k ● 6 ● 61 ● 90

Thanks for your answer, both answers together have given me the "aha!" moment. I think without your answer I wouldn't have entirely grasped the concept, but I prefer Constantine's answer as an indicator to future readers. – e_i_pi Dec 21 '17 at 10:57