DDD Decoded - Domain Services Explained

published on 16 August 2016 in Domain driven design

Domain services are a bit confusing at first, when you don't know exactly what a an Application Service is. Also, many developers try to cram a lot of business rules in their Aggregates, when a Domain Service (DS) would be more appropriate. But let's start with the beginning....

Why do we need Domain Services?

Some business rules don't make sense to be part of an Aggregate. For example, you want to check an account balance before debiting an account. Or you want to see if a certain operation is allowed by the business rules (not to be mistaken for user authorization). Or you want to perform a simple calculation according to domain rules. Basically, any business rule required to move forward a business case, which doesn't belong to an aggregate should be a Domain Service. This is the DDD term for **business behaviour** outside an aggregate or a value object.

How to identify a Domain Service (DS)?

The easiest way is to simply check if the rules are having to do with business constraints required to maintain an aggregate invariants, including its value objects. If they're not, they're a DS, even if it seems

Value Object (VO) the other encapsulates the "amount must be greater than the balance" rule.

What about the Anaemic Domain anti-pattern?

In a business case we can use multiple DS and this might look like the Anaemic Domain. But it's not, the reason being that Aggregates should be as small as possible, not because someone said so, but because they should contain **only the relevant behaviour** for that model. Not *every* related behaviour. If something is 'outside' an Aggregate, then it's probably is a Domain Service.

Implementation tips

A DS is just a name signaling business behaviour. It doesn't mean you have to implement it in one way or another. In many cases you can **implement** all the domain services from that Bounded Context as (static) functions in one class, while in other cases you might want one class per DS. As with everything, it depends.

A DS should be visible and consumed inside that Bounded Context only! Yes, you can define interfaces that can be used by your application service, they're great for testing, however their concrete implementation should be in the same BC as the business cases where it's used.

External services

implement a new class inside that BC and reconfigure the DI Container. Simple stuff.

What if we need a service which is part of our app but part of another BC? For a monolith, you can cut corners and use it directly (assuming you weighted in the consequences). For a distributed app, I'd suggest the External Service approach. In the end, for the BC it's something outside its boundaries, it doesn't matter where the actual functionality is implemented as long as it's not inside the boundaries. For the BC's point of view, it's an external service.

Conclusion

We have Domain Services simply because we want to keep a concept's model relevant, so any behaviour which doesn't *naturally* fit that model needs to be expressed somehow. Also DS shouldn't care about state, they represent domain behaviour only (their implementation should be stateless).

Share Tweet Pin Email Share

« Previous Next »

;

Sapiens Works

Converting tech into business advantage

Topics About

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data