How to avoid an anemic domain layer and still have rich validation and business rules

Asked 16 years, 2 months ago Modified 13 years, 1 month ago

Viewed 2k times

13

If you have a domain object, and you want to do something useful and central to that domain object's responsibility like ensure it is valid, you sometimes need to access the state of related objects in order to perform this validation.

How to avoid the domain object needing to call out to a Repository or Data Access Layer? You can't always walk the collection relationships, even with lazy loading, because of performance, and you often want to execute queries in the domain object. You can dependency inject repository implementation into the domain, but not really pure and complicates testing.

I've always relaxed things and allowed access out from domain to a repository using DI. I have seen no clear examples of how to have a 'pure' domain layer in a complex application which is not also anemic and has a service/application layer doing all the grunt and messing with what should be innards of the domain objects.

domain-driven-design

Share

Improve this question

Follow

asked Oct 7, 2008 at 10:41

Keith Patton

948 ● 5 ● 12

2 Answers

Sorted by: Highest score (default)

12

- If the object is a value object, it should be immutable and validated during construction.
- If the object is a root aggregate, and that its own state is sufficient to tell you if it is valid or not, you could add a validation method on it, which cascades through the aggregation.
- Lastly, and I think it is your main concern, if you need to access several related objects (that are not in the same aggregate) to ensure one of them is valid, you definitively needs to deport this logic in a specific validation service.

I really think that injecting services and repositories into entities are not the best choice. Creating dedicated services seems more appropriate, and I don't see why it will leads you to have anemic domain objects.

In short, if you can validate you object state without relying on services or repositories, let the object takes care of it, at the aggregate root level. When you need to

query services or repositories, or when you need other entities, then strongly consider moving this logic outside the object.

Share Improve this answer

Follow

answered Oct 7, 2008 at 11:12

Romain Verdier

13k ● 7 ● 59 ● 77

> Injecting onjects into the entity is the main ideea of keeping the domain layer decoupled . Injecting repositories in the entity is the best choice. And what do you mean by dedicated services ? Domain services are used only if the context of command spans over several entities. There shouldnt be services dedicated to an entity. "--(2x minus)" – Tudor Oct 4, 2012 at 23:08

1

I answered a similar question just a few hours ago. The answer contains some guidance I use when a try to enrich my model with logic and behavior without making it dirty with dependencies to infra-tech related stuff. Having trouble putting real-world logic into the DDD domain layer

the answer also links to other useful resources.

Good Luck and feel free to mail me or ask me about DDD and avoiding Anemic models. It is a interesting topic where people tend to solve this in different ways.

Share Improve this answer

Follow

edited May 23, 2017 at 11:47

Community Bot

1 ● 1

answered Sep 6, 2011 at 9:54

Magnus Backeus
2,286 ● 18 ● 24