Injecting domain service into the AggregateRoots in DDD

Asked 7 years, 11 months ago Active 4 years, 10 months ago Viewed 3k times

8

4

The commonly known advise in DDD is that an Aggregate Roots don't user a domain service. The domain service is to coordinate two Aggregate Roots to achieve a behavior.

It really surprised me when I saw this blog written by Rinat Abdullin with the title Building Blocks Of CQRS. Under the Domain Service section, you will read that a domain service is injected to an Aggregate Root.

Can an Aggregate Root accept a domain service?

domain-driven-design domainservices

Share Improve this question Follow

edited Jan 23 '15 at 15:58
SystematicFrank
14.6k 5 51 95

asked Jun 8 '13 at 3:16
wonderful world
8,704 14 76 142

4 Answers

Active Oldest Votes

11

✓

Please, **disregard that article**. It was written a long time ago and is plain wrong. If implementing a module with *AggregateRoot* and *DomainService* patterns, I would recommend to have a higher logic (e.g. request handler) which is responsible for:

1. Loading the aggregate
2. Performing calculations with the help of domain services
3. Mutating the aggregate state accordingly.

Share Improve this answer Follow

answered Feb 4 '15 at 7:57
Rinat Abdullin
21.5k 8 54 80

4 Rinat, don't you want to place a banner on these old articles of yours, that will repeat what you wrote in the answer? People often refer to your blog for ideas and best practices but you certainly know that some of described practices are not really best... No offence, just a suggestion. – Alexey Zimarev Feb 26 '16 at 12:02

10

In a way yes. If the AR really needs a service to do **some** of its job, then you can inject it as a method argument. If the AR needs a service for **most** of its behavior then probably it's modeled incorrectly.

Share Improve this answer Follow

answered Jun 17 '13 at 8:30

MikeSW

15.3k 3 33 50

If both the services live in the same bounded context, the aggregate roots can talk to each other via messages, so why injecting a service is necessary? – wonderful world Jun 22 '13 at 18:12

2 It's required ONLY if the AR needs it to perform some behavior. I don't see how messaging would help here. – MikeSW Jun 23 '13 at 7:52

I might have used the wrong term here. I meant domain events. – wonderful world Jun 23 '13 at 11:29

1 I understood you well, my answer is still the same – MikeSW Jun 23 '13 at 12:50

1 @MikeSW +1. So I would inject a service using an interface if I need it to help perform the work I need the AR to do (ie. ICalculator or IPaymentGateway). If the service is something that is actually inside the aggregate (like calculator is some class in my aggregate) then I can just "new" that up (unless it makes unit testing harder). Even better, if I don't need side-effects of the service, then I can use side-effect free Domain Events (I collect the domain events of the AR and then the application will dispatch them all near the end of the transaction, when done with AR). Is this correct? – prograhammer Dec 22 '15 at 2:38

5

I find the following explanation quite good. It's based on the book by Vaughn Vernon and 'injects' the domain service in the domain model through the method call that actually needs this service.

```
public class PurchaseOrder
{
    public string Id { get; private set; }
    public string VendorId { get; private set; }
    public string PONumber { get; private set; }
    public string Description { get; private set; }
```

```
        public decimal Total { get; private set; }
        public DateTime SubmissionDate { get; private set; }
        public ICollection<Invoice> Invoices { get; private set; }

        public decimal InvoiceTotal
        {
            get { return this.Invoices.Select(x => x.Amount).Sum(); }
        }

        public bool IsFullyInvoiced
        {
            get { return this.Total <= this.InvoiceTotal; }
        }

        bool ContainsInvoice(string vendorInvoiceNumber)
        {
            return this.Invoices.Any(x => x.VendorInvoiceNumber.Equals(
                vendorInvoiceNumber, StringComparison.OrdinalIgnoreCase));
        }

        public Invoice Invoice(IInvoiceNumberGenerator generator,
            string vendorInvoiceNumber, DateTime date, decimal amount)
        {
            // These guards maintain business integrity of the PO.
            if (this.IsFullyInvoiced)
                throw new Exception("The PO is fully invoiced.");
            if (ContainsInvoice(vendorInvoiceNumber))
                throw new Exception("Duplicate invoice!");

            var invoiceNumber = generator.GenerateInvoiceNumber(
                this.VendorId, vendorInvoiceNumber, date);

            var invoice = new Invoice(invoiceNumber, vendorInvoiceNumber, date, amount);
            this.Invoices.Add(invoice);
            DomainEvents.Raise(new PurchaseOrderInvoicedEvent(this.Id,
    invoice.InvoiceNumber));
            return invoice;
        }
    }

    public class PurchaseOrderService
    {
        public PurchaseOrderService(IPurchaseOrderRepository repository,
            IInvoiceNumberGenerator invoiceNumberGenerator)
        {
            this.repository = repository;
            this.invoiceNumberGenerator = invoiceNumberGenerator;
        }
        readonly IPurchaseOrderRepository repository;
        readonly IInvoiceNumberGenerator invoiceNumberGenerator;

        public void Invoice(string purchaseOrderId,
            string vendorInvoiceNumber, DateTime date, decimal amount)
        {
            // Transaction management, along with committing the unit of work
            // can be moved to ambient infrastructure.
            using (var ts = new TransactionScope())
            {
                var purchaseOrder = this.repository.Get(purchaseOrderId);
                if (purchaseOrder == null)
                    throw new Exception("PO not found!");
                purchaseOrder.Invoice(this.invoiceNumberGenerator,
                    vendorInvoiceNumber, date, amount);
                this.repository.Commit();
                ts.Complete();
            }
        }
    }
```

Share Improve this answer Follow

edited Jul 20 '16 at 9:59
superjos
10.8k 4 79 120

answered Mar 12 '14 at 6:46
GitteTitter
424 5 7

The PurchaseOrderService is an application service not a domain service, as I understand it. – wonderful world Mar 12 '14 at 18:02

3 @wonderfulworld: IInvoiceNumberGenerator is the domain service – mynkow Sep 19 '14 at 23:11

`IInvoiceNumberGenerator` is the domain service, `PurchaseOrderService` is the application service. You should never inject repositories into your domain service, as they are needed to orchestrate use cases, not business logic. – Robotronx Apr 5 '18 at 10:47

3

It's very hard to *inject* anything into domain objects, and doing so is quite technology specific. In java it requires compile time weaving of aspects into your domain classes. And although I could be mistaken on this, I think that most DDD leaders think that this is, generally, a bad idea. Both Evans and Vernon both actively discourage it, and I like to listen to them. For a full explanation, read Vernon.

Share Improve this answer Follow

answered Jun 8 '13 at 4:29
Software Engineer

13.5k 5 57 83

I agree that Evans and Vernon disagree to Rinat's idea. In the example, he uses a pricing service to find a threshold in the method LockCustomerForAccountOverdraft. That is a Business Rule evaluation which can be done by sending a command. With Evan's and Verson's approach, there should be a higher level domain service called LockCustomer where the coordination of PricingService and CustomerAggregate can be done. I think Evans and Vernon apply SRP so there are more components. Rinat takes a simple approach, so he may okay with breaking the open/closed principle in this case for Customer. – wonderful world Jun 8 '13 at 11:43