The right way to implement associations in DDD?

Asked 6 years, 9 months ago Active 6 years, 9 months ago Viewed 1k times

7

3

We are trying to adopt Domain-Driven Design in our project for the first time. The problem I have is with associations between entities. How do you do them right?

Say, I've got entities `Employee` and `Contract`, a simple one-to-many association. How do I model it?

Option 1: Aggregate.

Problem: The problem here is that, if I understand it correctly, all entities in an aggregate must be loaded when an aggregate object is created. I can't lazy-load entities when they are needed because it would require referencing a repository from an entity, which apparently is bad. But fetching all of an employee's contracts from the database every time would be a big performance issue.

Option 2: Fetching an employee's contracts by using a repository (e.g. `ContractRepository.GetContractsForEmployee()`) and adding `EmployeeId` property to `Contract` class.

Problem: it makes hard to put any business logic into entities. I would like to have a method, say, `Employee.Dismiss()`, but it would also need to update the employee's contract. This means I would need to put this logic in a service. The problem is, I can't think of much logic operating only on an `Employee` and thus the model would become somewhat anemic, with most logic inside services.

How do you deal with these issues in DDD?

oop architecture domain-driven-design

asked Jun 4 '13 at 8:20

the right one. You don't have to come up with lots of behavior for an Entity. If a domain object is very simple in the real business, model it that way. It matters how the Domain defines the concept(the semantics) and not how many methods the object has. – MikeSW Jun 4 '13 at 11:18

MikeSW is correct. A very important aspect of DDD is Ubiquitous Language. The way you speak about the Domain MUST be the way the Domain Expert speaks about the Domain otherwise the code becomes awkward and unintuitive. It sounds trivial but if you get this right then Entities start to suggest behavior. e.g. Employee.Resign(); Contract.Terminate(); – Asher Jun 5 '13 at 10:00

3 Answers

Active | Oldest | Votes

3

This is just my take on it... without knowing your domain.

First, here is a good resource to read (part about Aggregates and Roots).

In DDD terminology, `Employee` and `Contract` are both **entities** (because they both have an identity).

"Aggregates draw a boundary around one or more Entities. and also: *Each Aggregate has a Root Entity, which is the only member of the Aggregate that any object outside the Aggregate is allowed to hold a reference to."*

The question is: do `Employee` and `Contract` form an aggregate, with `Employee` being the root entity? Obviously not, because other domain entities could also have a reference to a `contract`, and the contract id's are globally unique, not only within a `Customer`.

So, taking into account these rules, `Employee` and `Contract` are both **aggregate roots**.

Then: *"Only aggregate roots can be obtained directly with queries; so this means that we should have a repository per aggregate root."*

So in this case, we have an `EmployeeRepository` and a `ContractRepository`.

Taking all of this into account, I would not add a relation between employees and contracts in the domain model; but treat them separately. After all, if you need an `Employee`, you don't necessarily need his `contracts` too, they are both different aspects.

Option 2 is what I would choose: use the `ContractRepository` to get the contracts you are interested in. And if needed you could add a domain service that is responsible for aggregating employees and contracts if needed.

If you also define a `Company` entity, then dismissing an employee could be the job of that entity.

1

We recently got into the DDD approach as well. If I were to do it, I would have the following (attributes are simplified for brevity):

```
public class Employee() {

    String name;
    Set<ContractNumber> contracts;

    public void addContract(ContractNumber contractNumber) {
         this.contracts.add(contractNumber);
    }

}

public class Contract() {
    ContractNumber contractNumber;
    Date effectiveDate;
}

public class ContractNumber() {
    String contractNumber;
}
```

ContractNumber is a Value Object that is referred to from within Employee. In this example, Employee is within a BoundedContext that deals with Employees and their respective contracts. There may be other representations of Employee in other bounded contexts.

As discussed in other answers, there will be repositories for both Employee and Contract.

answered Jun 12 '13 at 5:00

jett
705 ● 6 ● 18

0

You need to find your true invariants.

Here you could have an invariant such as: you cannot dismiss an `Employee` that has already been dismissed.

If this is the only real invariant, then you could make one `Employee` aggregate, which would only have the IDs of the associated

answered Jun 4 '13 at 10:48

Guillaume

1,533 ● 1 ● 10 ● 25