One service for each entity?

Asked 8 years, 8 months ago Active 1 year, 5 months ago Viewed 2k times

2

4

Again - i'm confused about DDD things :)

I have architeture (I'm still working on it) that in short hand looks like that:

```
DataLayer:
 EntityDao -> Implementing domain layer interfaces (NHibernate)
DomainLayer:
 EntityRepository -> Repository for each entity with injected Dao
 DomainObjects/Entitys -> Some logic
UI
 ASP.Net MVC
```

And I'm right now in that point where I feel to create and use some Service class. I have some questions with that:

1.Should I create at least one service for each entity/domain object ?

2.a.Should services have "query" method like Find, FIndAll, FindAll(LINQQuery)?

2.b.Should I stop to use Repositorys in upper layers (UI) to get sets ("Find"-like methods) of entity's and start to use only services?

3.If answer on 2 question is No - Should I use Services and Repository in parallel fashion (When in UI I just need to get all entity's I use Repository.FindAll, when I need to get some "logical" list of that entity's i use Service.FindXXX method)?

4.Somehow I feel that Repositorys don't fit in Domain layer - should I separate them somehow and in DOMAIN leave only domain specific objects like Entity's and Services ? If Yes - give me some structure example how to achieve that.

Examples of some objects:

Dao:

```
        T entity = (T)NHibernateSession.Get(entityType, id);
        return entity;
    }
    public T Load(object id)
    {
        T entity = (T)NHibernateSession.Load(entityType, id);
        return entity;
    }
    public virtual T Update(T entity)
    {
        NHibernateSession.Update(entity);
        return entity;
    }
    ...
```

Repository:

```
public class BaseRepository<T>:IRepository<T>
{
    private DataInterfaces.IDao<T> mDao;

    public virtual T Get(object id)
    {
        return mDao.Get(id);
    }
    public virtual void Delete(T entity)
    {
        mDao.Delete(entity);
    }
    public virtual T Update(T entity)
    {
        return mDao.Update(entity);
    }
    public virtual IQueryable<T> FindAll()
    {
        return mDao.FindAll();
    }
    ...
```

Domain objects, at the moment , it's mainly get/set container - background of this question is to remove that anemic model.

mgibas
288 ● 2 ● 14

Could you give an example of an `EntityDao`, `EntityRepository` and an `Entity`? – Marijn Jul 8 '11 at 8:10

Ah, that's clear. If that's what you mean by DAO and Repository, then Kostassoid is right and they're the same thing actually. – Marijn Jul 11 '11 at 11:07

2 Answers

Active | Oldest | Votes

8

1. One service per entity?

No. You do not need to create one service for one entity. In DDD you would create services for operations that do not naturally map to a single entity (or value object). A good service (from Evans) :

- The operation relates to a domain concept that is not a natural part of an entity or value object.
- The interface is defined in terms of elements of the domain.
- The operation is stateless

So a service can consume many entities and there might be many entities that aren't consumed by a single service at all.

2a. Should services have "query" methods (..)?

No. Generally speaking those are repository methods and are not placed on services. However, there can be operations on a service that return a collection of entities.

2b.Should I stop to use Repositories in upper layers (UI) to get sets ("Find"-like methods) of entity's and start to use only services?

That might be a good idea. Often, when an application uses many repositories in the UI layer, the UI performs domain operations on multiple entities. These operations should typically be implemented in the domain layer; either in the entities themselves, or in services.

3. Should I use Services and Repositories in parallel from the UI?

answered Jul 8 '11 at 8:04

Marijn
9,586 ● 4 ● 48 ● 72

6

My thoughts:

1. Services provide upper level interface for business logic. Generally, you should hide your domain level implementation details (which is lower) behind it's methods, basically commands or queries. A good way of thinking could be to name your service methods after use cases or user stories. And you shouldn't provide more functionality that required by service clients (UI).

2a. If your UI needs all data from entities of some type then yes, otherwise no. And FindAll() is hardly a use case.

2b. You should use Repositories from your Services, that's actually the only place where you should use it.

3 see 2b.

4 Yes. You should leave interfaces for repositories in your Domain, but implementation should be in Data Access. Then you can glue everything with some IoC container. Could be like this:

```
//in domain
public interface IUserRepository {
    User GetById(Guid id);
}
//in data access
public class UserRepository : IUserRepository
{
    public UserRepsitory(/* some orm specific dependencies */) { }
    public User GetById(Guid id) { /* implementation */ }
}
```

So you'r saying that I should remove DAO's ? – mgibas Jul 7 '11 at 13:12

2 Well, basically Repository and DAO are different patterns for solving the same problem, so I would say yes, I can't imagine why would you need another abstraction layer. At least that's what my experience tells me. – Kostassoid Jul 7 '11 at 13:34

About 2a: I use Repositories inside View Helpers and Controllers, in this case the correct would be implement find methods in my Service Layer? But this will not be a change of responsibilities? I think that is fine to acess Repositories inside Controllers or View Helpers just to get data, the incorrect would be to change the data within these layers. Or I'm wrong? – JCM Apr 2 '12 at 1:04

@JCM, I'd go further and use CQRS pattern, meaning not using SL at all for read-only operations. In this case you can extract your complex queries to some abstractions but I wouldn't call them Repositories as it is kind of misleading. If, however, you need to query using SL, then it's usually better to define some kind of DSL to construct a query specification in domain language (like Query.Users.WithRatingAbove(100).And.LivingIn("US").AsList()) which can be translated at SL to Repository queries. – Kostassoid Apr 2 '12 at 10:19