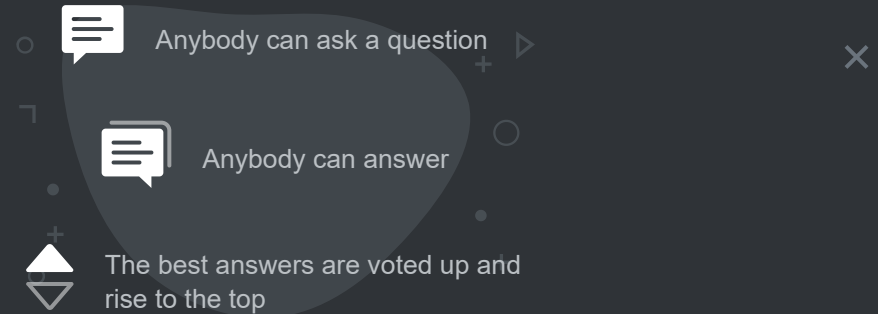






Software Engineering Stack Exchange is a question and answer site for professionals, academics, and students working within the systems development life cycle. It only takes a minute to sign up.

Join this community



## DDD with ORM where should the business logic go?

Asked 8 years, 3 months ago   Active 6 years, 5 months ago   Viewed 7k times

-  10  
I have used an MDA (model driven architecture) tool in the past where we modeled via UML and this generated the business entities (our domain model) and the ORM (mapping etc) amongst other things.
-  A lot of the business code and services working on the domain were part of the model and our repositories were returning the business entities (so it would have been impossible to switch out to another ORM (not that we wanted to)).
-  7  
However, now I am starting a project and I want to think in terms of DDD.
-  So far it feels as though I put my business logic in my domain model and via repositories I would work with the ORM (which ever I chose). However, if I wanted to continue to use the MDA tool for the ORM part of the application, the model created here would be very anemic (i.e not contain any business logic). Similarly if I used Entity framework (.net) or NHibernate for my ORM it too would be an anemic model.? I am not sure where you would put the business logic if I just used NHibernate.

Am I correct in thinking this way, in other words with DDD all the business logic in the domain and just use the ORM for persistence via repositories?

orm

entity-framework

domain-model

nhibernate

domain-driven-design

edited Feb 5 '12 at 23:22



yannis

36.8k ● 39 ● 172 ● 212

asked Dec 11 '11 at 7:22



JD01

1,219 ● 2 ● 13 ● 18

### 3 Answers

Active

Oldest

Votes



12



so it would have been impossible to switch out to another ORM (not that we wanted to)).

That seems wrong. A major advantage of the repository pattern is that you hide the data access logic and that it is easily exchangeable.



So far it feels as though I put my business logic in my domain model and via repositories I would work with the ORM (which ever I chose). However, if I wanted to continue to use the MDA tool for the ORM part of the application, the model created here would be very anemic (i.e not contain any business logic). Similarly if I used Entity framework (.net) or NHibernate for my ORM it too would be an anemic model.? I am not sure where you would put the business logic if I just used NHibernate.



An anemic domain model is considered a bad practice by many, for example by Martin Fowler. You should avoid such a design because such a model leads to procedural design techniques rather than a good object oriented design. You then have data classes and manager/processing classes which means you separated state and behaviour. But an object really should be "state *and* behaviour".

NHibernate does a great job at persistence ignorance. You can hide away the mapping details in XML or with FluentNHibernate and just write plain POCOs. It's very easy to create a rich domain model with NHibernate. I think you can do that with entity framework and the MDA tool, too. As long as this tool produces partial classes you can extend the generated code pretty easily without having to worry that a new generation might destroy your user-written code.

To cut this long story short. When you use NHibernate, nothing, I repeat *nothing*, stops you from embracing a rich domain model. I recommend using it with FluentNHibernate and mapping by hand. The mapping code takes only 5 to 10 minutes to write. I suppose that the very same is true for entity framework and that its tools at least create partial classes that are easily extensible.

Am I correct in thinking this way, in other words with DDD all the business logic in the domain and just use the ORM for persistence via repositories?

For the most part you are correct. You should have a rich domain model. Especially when things become more and more complex, it's easier to maintain and extend when you've designed it properly. But do keep in mind that DDD also knows (Domain Layer and Application Layer) Services to implement business logic and Factories to encapsulate creational logic.

I also tend to differentiate business logic into domain logic and actual application business logic. The domain logic is how the domain interacts and behaves while the application logic, which is a completely different layer, encapsulates how the domain is used for the specific use-case/application. Often times I have to update the domain model to support specific use cases and to make it more powerful.

edited Dec 11 '11 at 8:43

answered Dec 11 '11 at 8:27



Falcon

18.5k ● 4 ● 70 ● 92

2 +1: I also separate the domain logic layer from the application logic layer. I put all the ORM and database stuff in the domain logic layer. The application logic layer knows nothing about the ORM, transactions, and all that stuff: it only sees business logic classes and calls their methods. I find this approach very effective for having a simpler and cleaner application logic layer. – Giorgio Dec 11 '11 at 10:23 ✎

@Falcon: Thanks for the info. When I mentioned the anemic model what I meant was if I create a domain using DDD, one version of my repositories would possibly be the mda version where I would be just moving my entities to the mda entities (i.e. anemic model) and then persisting them in transactions etc. Would this be okay? I doubt I will use the MDA tool but just trying to understand how I could if I wanted. Does this sound right? –

JD01 Dec 11 '11 at 17:10

@JD01: I don't quite understand you, but it sounds like you want to transform domain model entities so you can persist them easily. That's pretty much like using DTOs and automapper (google it) could be a useful tool for such a task. Such an approach does not necessarily interfere with DDD best practices. After all, repositories are also meant to hide the Data Access Logic. You could just transform your Business objects behind the scenes into MDA DTOs and then persist them and and API user wouldn't even notice. I think that's ok. – Falcon Dec 11 '11 at 17:32 ✎

1 @JD01: I suggest you take a look at the following link to see how many Enterprise java guys do it. They basically have a DAO, a DTO and BO (Business object). For me, that's too many layers but the design is ok. [java.sun.com/blueprints/corej2eepatterns/Patterns/...](http://java.sun.com/blueprints/corej2eepatterns/Patterns/) – Falcon Dec 11 '11 at 17:33

@Falcon: Yes, I was thinking along the lines of DTOs being my MDA objects, not that I will go that way. Just getting a grip of what each part of the DDD players do. Thanks once again. – JD01 Dec 11 '11 at 21:29

3

However, if I wanted to continue to use the MDA tool for the ORM part of the application, the model created here would be very anemic (i.e not contain any business logic).

I don't know which MDA tool you're using, but the ones I've worked with always create partial classes so you're free to complete them with as much business logic as you want.



However, I do feel MDA tools are a little bit less appropriate in a DDD context than ORMs, because code generation often tends to produce classes that are muddled with tool-specific noise instead of the streamlined, clearly expressed domain entities we expect. Actually, what you often get is a mix of domain data, persistence logic, constraint validation logic... and I don't know if there's a way to separate these concerns with most MDA tools.

And of course, you can't touch the generated code except via partial classes, which means you can't eliminate potential anti-DDD behaviour that would be integrated. This is problematic in an approach where you want to enforce strict barriers between Aggregates and tailor the relationships between your entities perfectly. Build time in a continuous integration environment can suffer from the additional code generation step as well.

Apart from that, I think Falcon pretty much said it all -absolutely nothing in ORM or MDA tools prevents you from having rich domain entities.

answered Dec 11 '11 at 10:42



guillaume31

7,437 ● 15 ● 29

---

Hi, I am using ECO (enterprise core objects) from capableobjects.com and it is exactly as you described it. – JD01 Dec 11 '11 at 17:01

---

1

What I do in my team is to model my object, domain and add my business logic at the same time. I don't use Model Driven Development which would generate a code from a model but prefer the annotation approach. I mean that at object level inside the class diagram I add ORM stereotypes. This will add persistence annotations directly in the code which are compatible with EJB3/hibernate. The database creation is done by Hibernate and certainly not by the UML templates. This is a lot better because if the code change and the added annotations is not exactly what the hibernate specialist then he/she can change it but the model is still good. I can also change my requirements and my domain model is still ok.



Developers can add business logic inside each method and add a comment, I can also model and add constraints. For example the sales should be over 50k if not etc.... I don't need to code it but just write in the model and this information would be visible to the

developer team. Really cool and flexible UML.

edited Oct 22 '13 at 16:03



Oscar Mederos

1,008 ● 7 ● 20

answered Dec 11 '11 at 10:27



UML\_GURU

258 ● 2 ● 3