Sapiens Works

Converting tech into business advantage

Topics About

DDD Decoded - Bounded Contexts Explained

published on 12 August 2016 in Domain driven design

In DDD, we organize the Domain into Bounded Contexts(BC), with at least 1 BC. An important pattern, the BC is a bit tricky to "get" and to explain, but I assure you that you *need* it for proper DDD and it simplifies things on the long run.

What is a Bounded Context?

What if I told you that you've been using the bounded context principle for a while, without knowing it? You know those UI, Business and Database/Persistence layers? Those are literally bounded contexts, an implementation of the **separation of concerns** principle. A complex Domain can be considered a bunch of functionalities, but we like to group them into 'specialized' modules, where each module is responsible for a high level concern.

If the Domain would be a company, the BCs would be its departments: accounting, HR, marketing, production etc. Each with its own responsibility, different but working together. And since DDD is about being **Domain** driven, we want to design our application similarly to how the real domain(business) is organized.

trait of maintainability. Basically, we group together related models while keeping them separated from other models.

And like everything else in DDD, we aren't the ones designing or deciding BCs, we *identify* them. The business already knows what responsibilities and limits each department has, we're just putting that information together.

How we identify a BC?

Well, we pay attention to the language. The ubiquitous language is not just a DDD buzzword. It means that the business language has the same meaning in a certain context. `Invoice` means something to Accounting, but it can be just the name of a class for a programmer. Just because the name is identical doesn't mean it's the same concept. And Marketing can say `Customer` and Accounting can say `Payer` but they can both mean the same thing for the business as a whole.

The important part here is that names are by themselves irrelevant. We need to understand the *concept*'s definition that makes sense in that context in order to place a business case in a certain BC. Many times is quite easy, `Create an order` is part of `ECommerce` but sending the order's confirmation email is clearly part of a different BC, something like `Marketing` or `Sales`. *How* you send the email it's an implementation/infrastructure detail and we're not interested in that (at this point).

How do the BCs communicate?

If one BC is interested in data from another BC, it will just subscribe to the relevant events and then it will use that data to maintain its very own **read model** (yes, that's CQRS in action). And since messages are data and not part of a model, boundaries are not violated. All this should be part of the BC's own infrastructure.

But what about invoking a business case which is part of a different BC? When you need that, it means you're dealing with a business process, a sequence of business cases, however, even if a business case belongs to a BC, the process itself doesn't and we're going to have a **process manager** a.k.a saga in charge, which is a technical implementation detail. When we identify a process, we care about which business cases are part of it, the manager itself doesn't exist as part of the domain.

A business case shouldn't invoke directly the next business case i.e it shouldn't act as a part-time project manager; in order to achieve decoupling and to maintain boundaries, business cases don't know anything about other cases, it's the process manager's job to know and to invoke the next case.

In special situations...

..you might want to couple one BC to another, because it's the easiest way. It can be a valid, good enough solution, but you have to be careful. The trick here is to find the right compromise between decoupling and easy implementation.

Sapiens Works

Converting tech into business advantage

Topics About

Sapiens Works

Converting tech into business advantage

Topics About

109 Shares Share Tweet Pin Email Share

« Previous Next »

;

ALSO ON **SAPIENSWORKS**

Using Discriminated Unions In C#

4 years ago • 1 comment

Converting tech into business advantage

Repository vs DAO

3 years ago • 1 comment

Converting tech into business advantage

The Myth of Manual POCO Mapping …

4 years ago • 2 comments

Converting tech into business advantage

It's Time To Let The ORM Go

2 years ago • 1 comment

Converting tech into business advantage

3 Comments **Sapiensworks** **Disqus' Privacy Policy** 1 **Login**

Recommend Tweet Share **Sort by Oldest**

"If one BC is interested in data from another BC, it will just subscribe
to the relevant events and then it will use that data to maintain its
very own read model"
What if one aggregate is interested in data from another aggregate in the same BC. Does it use repository to fetch required aggregate state, or some domain service which read data from BC's read model? And how we must deal with concurrency, locking or queuing everything?

^ | ˅ • Reply • Share ›

Mike Mod ➦ xiety • 4 years ago

An aggregate is a bunch of rules and aggregates don't overlap so data doesn't belong to one, the 'data' is a part of the business state and with CQRS it's part of a read model. How you read that model is an implementation detail, use whatever you want. A domain service represents domain behaviour, it might read from persistence but that's an implementation detail.Certainly it's not used as a DAO.

" And how we must deal with concurrency, locking or queuing everything?" - You're asking implementation details which are very app specific. This tells me that you don't understand yet that DDD has nothing to do with programming. It's a method to gather requirements, to understand the "what". The "how" is programming and there's no recipe involved here.

2 ^ | ˅ • Reply • Share ›

Kamil Kubů ➦ xiety • 2 years ago

What is more scaring of this advice, is that it sounds like a guidance to endless data redudancy. What if you have some really huge domain with central business concept (customer, account, ...) with asteroid belt of BCs dragged to this star of the domain by neccesity to deal with its data in every command they want to accomplish. Would you spread out your data all around to keep your BCs "bounded"?