How to clearly define boundaries of a bounded context

Asked 4 years, 2 months ago Active 2 years, 8 months ago Viewed 1k times

9

2

After a month or so of reading and researching DDD, I decided to start my own project and created DDD with these bounded contexts>

- Clients
- Products
- Orders
- Billing

Each bounded context has rest API as a presentation layer, domain layer, persistent layer.

So far so good, code is running smooth, but coming from a monolithic world, I am still trying to figure out the following:

- when I want to create a new client, issue new invoice, create new order I want to for example access list of countries. Do I:

a) create a list of countries in each BC

b) create a Countries BC -> API and use it to get a list of available countries

- when integrating with 3rd party API using an anti-coruption layer or an adapter layer, what data has to be included in my domain model? For example if I want to integrate a zendesk API with a Client BC. Do I need just a ticketID in my domain, or I have to extract all the data from Zendesk that I want to access and use in a Client BC?

If my MVC app is actually getting data from APIs (presentation layers of my bounded contexts) I find it very difficult to clearly define boundaries of each BC. Does it mean that a properly designed BC would serve a single MVC controller without a need to consume additional APIs?

design-patterns object-oriented-design domain-driven-design

asked Apr 26 '16 at 15:58

Dario Granich
695 5 14

2 Keep in mind that data duplication is not a primary concern in DDD... – John Apr 26 '16 at 17:21

4 Answers

Active | Oldest | Votes

7

If your different bounded contexts understand the meaning/purpose of a country differently, then you need to model it appropriately different in each one. However, if we are speaking simply of reference data of ISO codes and names, then I believe it's pretty fair and standard to stash it wherever is convenient and make it accessible to all interested parties. For example: a database, a configuration file, a web service, etc.

I also wanted to look at your model a little bit. The pieces you have listed could very well be "entities" in one "bounded context", depending on the company's structure. BCs often end up being defined around different areas/departments/teams, since that's frequently the natural boundary between "ubiquitous language"s. So for example, instead of Sales/Products/Orders I'd expect the BCs to be along the lines of Sales/Manufacturing/Warehousing.

Inside those BCs, you don't focus on the nouns. You focus on the use cases, and create models of the nouns that can fulfill the use cases. The methods on an "aggregate root" execute use cases and make the appropriate changes to the related models.

> ... all models are wrong, but some are useful.

Also bear in mind that each BC may use an entirely different system or architecture. A given BC may not merit using "DDD software components" at all, and most of them probably don't. DDD is less about prescriptive software components and more about the process of designing software. The point is to focus on understanding the company's bounded contexts, mapping out each context's ubiquitous languages, and modeling the code for that context using their ubiquitous language. That way when you interact with stake holders and refer to the code, it sounds to them like you are speaking in business terms they understand. And recognizing that the same word has different meanings in different BCs.

your analysis suggests is the most practical thing to do.

edited Apr 26 '16 at 19:15

answered Apr 26 '16 at 19:06

Kasey Speakman

3,831 14 25

3

From your questions, I think you misunderstand bounded context. You may want to reread Chapter 14 of the blue book.

Trying to answer generally - you have to be careful about sharing concepts between two different bounded contexts. After all, part of the reason that the boundary exists is that the ubiquitous language changes. To assume that the same data (and the same representation) of an entity can be used in both contexts is naive -- it may be right, it may be wrong, but there's no good way for those of us on the outside, without access to your domain experts, to judge.

For example, in the client domain, "country" could be related to residence or citizenship. In billing, it might be related to currency exchange rates. In some of those domains, you might need to worry about tariffs and the like.

A second question that you need to raise is which of your models is the book of record for the "shared" data. In the case of "country", the right answer is probably that none of them are! Geopolitical topology is not controlled by your model.

What is supposed to happen in your domain models when a country is occupied by a foreign power?

Keep in mind; a lot of us are accustomed to thinking about data structure; what is the relationship between one piece of data and another. And that's great when you are considering reports, and trying to ensure that all of the data that you need has been collected by your solution. But domain models are not just about structure, but about change. You need to put your attention on that part too, and make sure that you well understand how the data constrains the changes (and how those constraints vary from one bounded context to the next).

answered Apr 26 '16 at 16:57

VoiceOfUnreason

23.6k 1 31 57

0

The concepts you mention (Clients, Products, Orders, Billing) are typically represented in a single Domain Model and hence Bounded Context. I suggest you are understanding these concepts incorrectly.

answered Apr 28 '16 at 9:04

aryeh

241 1 7

---

i don't really agree with you. for example if you have 1M clients generating 5M invoices you would want to

Dario Granich Apr 28 '16 at 11:18

1m clients generating 5m invoices is hardly typical. Small to mid SMEs typically have integrated ERP systems. Mid to large SMEs and Enterprises integrated or Independent (typically suite based) applications. If your circumstances support development of a solution based on 4 complex domain models and you can handle that, kudos to you. – aryeh Apr 28 '16 at 11:53

0

My take on this subject is to define bounded context using a business-capability mapping or other similar techniques like Value-chain analysis. It comes down to following steps:

1. Define your system's higher-level responsibilities, or business-capabilities. The best way to do it I think is to conjure up with steps your enterprise goes through to obtain a business-value. The logical boundaries you come up with are your business-services, or, if you like, bounded contexts.
2. Delve deeper within each service.
3. Identify the communications between your services alongside the first two points.

So the initial focus is on how your business operates.

Couple of practical advises:

1. If one of your contexts/services/etc need some other context's data, most probably your boundaries are wrong.
2. The highly desirable way of context communication is event-based. This is a key to scalability and reliability. If you need synchronous communication, most probably than not you boundaries are wrong, again. Besides that, synchronous communication will kill your system.
3. Your domain is more eventually-consistent than you think. Just like everyone else's. Don't try to make everything 100% consistent. There is no practical sense in that.
4. Contexts don't need to be orchestrated. They are self-contained. Like humans.

With this approach you end up with highly autonomous, maintainable and reliable services. You might want to check an example of defining context boundaries.

answered Oct 11 '17 at 18:09

Vadim Samokhin
1,820 1 10 15