DDD Hamburg Usergroup

#4

~~Want To Contribute?~~

Please To Contribute!

https://github.com/DDD-Hamburg/participate

Employing DDD In Legacy Systems

How To Escape The Big Ball Of Mud

About Me

Hauke Stange
Software Developer

https://github.com/Partyschaum

https://twitter.com/EinJungeAusKiel

https://www.xing.com/profile/Hauke_Stange

How We Got Here

- Attended a reading group for Vernon’s “Red Book”
- Got excited by the ideas of DDD
- Wanted to start implementing DDD
- Got bogged down by reality

How We Got Here

- Googled “start ddd legacy system”
- Found *“Getting Started With DDD When Surrounded By Legacy Systems”*
- Looked for an opportunity to dive into this topic
- Prepared this talk

Requirements for DDD

- Clean Bounded Context
- A model “well suited to the problem at hand”
- A basic set of DDD principles and techniques

What We Have

- Tangled legacy systems
- An implicit but established model
- Changes on the model aren’t possible without effort

What Can We Do?

Just continue…

- Slows development over time
- Changes could (and will) break your system
- Just adding new models leads to conflicting rules and concepts in your code base

What Can We Do?

High risk legacy replacement project...

...without a serious understanding of DDD principles and techniques you'll (very likely) end up with a legacy system again!

But How to Start Then???

“Introducing a difficult new set of development practices and techniques is best done incrementally, as in a pilot project.”

Bubble Context

“A ‘bubble’ is a **small bounded context** established **using an Anticorruption Layer** (ACL) for the purpose of a **particular development** effort”

Bubble Context

“**The bubble isolates** [..] so the team can **evolve a model** that addresses the chosen area, relatively **unconstrained by the concepts of the legacy systems**.”

Getting Started

Developer sees something new. Developer wants to use it!

Technologies

- React / Redux
- ELK Stack
- Apache Kafka
- AWS Lambda

Patterns / Concepts

- CQRS & Event Sourcing
- Reactive Architecture
- Microservices
- DDD

What have all those things in common?

You absolutely need a proper use case to justify their application!

“To get started, you need to choose an important, yet modest-sized, business related problem with some intricacy.”

Defend your domain against all odds!

Isolation is king!

“Business related problem with some intricacy”

LEGACY SYSTEM

Bubble Context

Frontier Guards

Wall Of China

LEGACY SYSTEM

The Translation Agency

ACL in short

English ↔ 英語

Translates concepts from outside the context to the language spoken inside

But...

Don’t we do this all the time?

Mapping things from one to another?

Adapters
Bridges
Facades

Yes… But no...

“...translates concepts
to the language
spoken inside…”

Problem With Mapping

User

1. A person who uses or operates something
2. A person who exploits others
3. The continued use or enjoyment of a right

Translate Concepts

User

- Customer
- Vendor

Shop

- Tenant
- Landlord

Real Estate

- Bidder
- Offerer

Auction

Translate Concepts

Mapping Things

- Concept leakage
- Technical solution

⇨ Yet another layer of abstraction

Translate Concepts

- Intention revealing
- Solution matches context

⇨ Establish a ubiquitous language

Translate Concepts

“An isolating layer to provide clients with functionality in terms of their own domain model. [..] Internally, the layer translates in both directions as necessary between the [..] models.”

Evans, *Domain Driven Design*, p.365

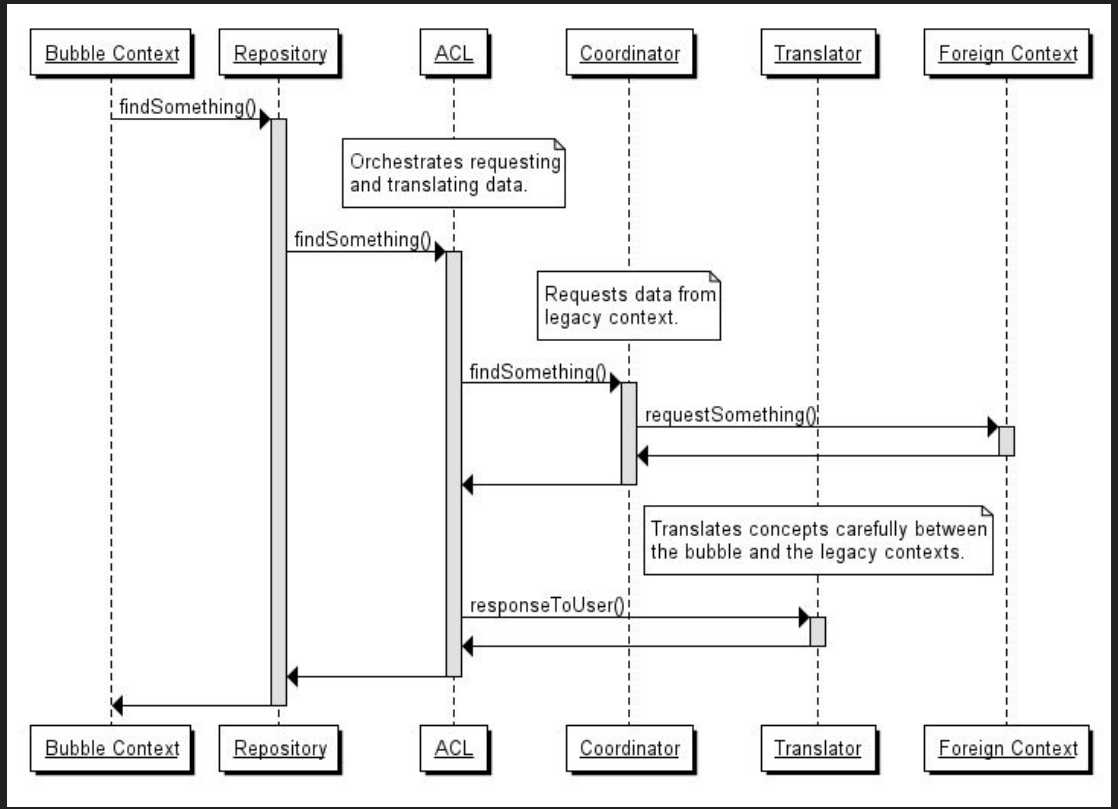
Short Recap

- We need a small business related problem
- Then model it in a bubble context
- Implement an ACL to protect it from the legacy system

Anatomy of an ACL

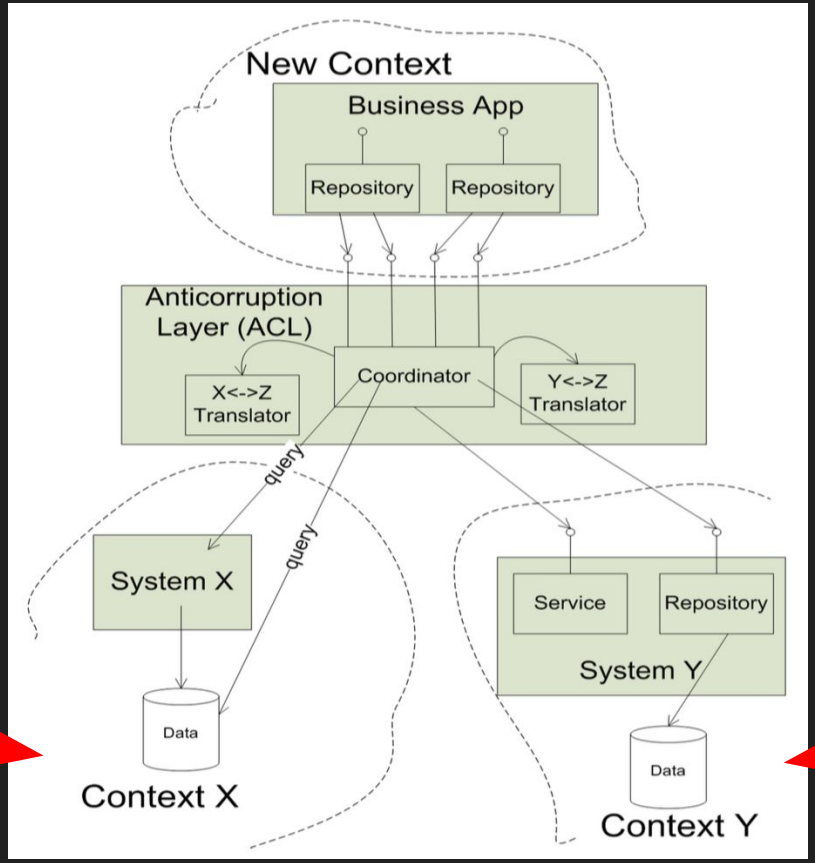
- Requests the API of the legacy system
- Translates concepts between bubble and legacy system
- One ACL may talk to many legacy systems

Anatomy of an ACL



The Repository is the “DDD building block used to **represent access to pre-existing objects**”.

The ACL-backed Repository



The ACL-backed Repository

- Illusion of own datastore via repository interface
- Dependent on upstream context
- Queries the legacy system
- Translates concepts
- **Enables the Bubble Context**

Bubble Context

Key Benefits

- Leverages a new set of DDD development principles and techniques
- Does not require a big commitment
- Can serve as pilot project
- If the Bubble bursts, the functionality remains
- **Design freedom is the point of the Bubble**

The Bubble Context **enables DDD** when surrounded by legacy systems!

Bubble Context

Drawbacks

- Translation can and will be tricky
- ACL is significant piece of software on its own right
- Any additional information needed from legacy context has to be worked out

Bubble Context

Drawbacks

- Allows no data to be used that isn’t available upstream
- New information needs to be added to the legacy system
- New information needs to be translated back to the Bubble increasing the dependencies

Pitfalls

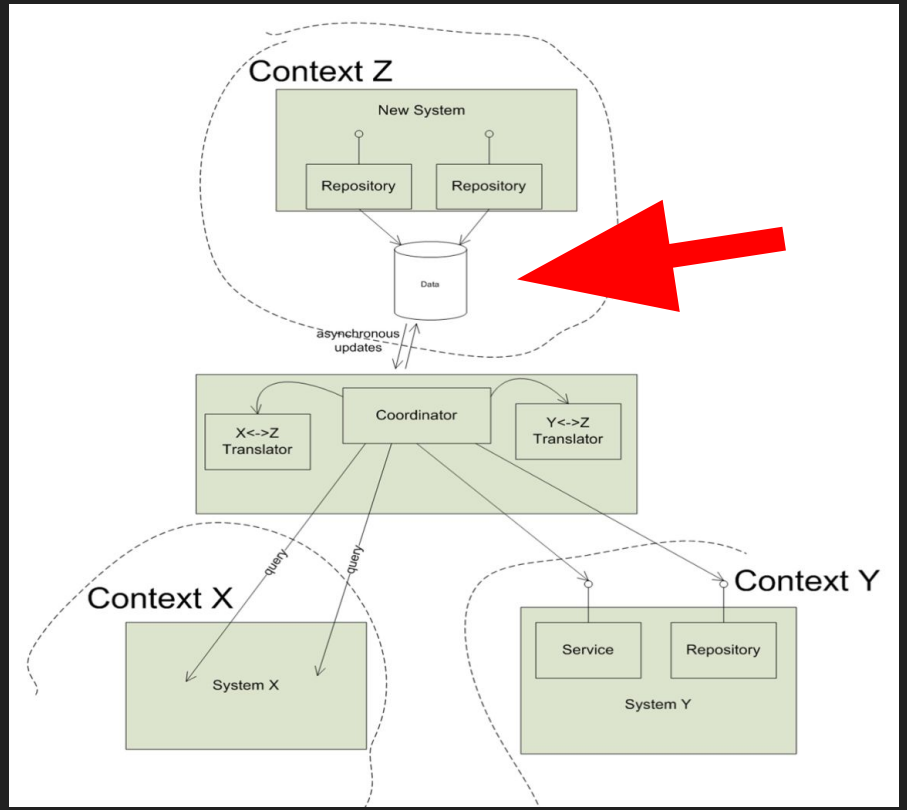
- Weak context boundaries
- Planning and coordination of the development efforts
- Adding data is a modelling job
- Only bring in data you use
- **No business logic in the ACL**

Directions

- Cut the umbilical cord!
- Expose Legacy Assets As Services

“The **Autonomous Bubble** [has] the ability to run its software [..] cut off from other systems.”

Cut The Umbilical



Enables evolution of the model since the context is more loosely coupled.

The **ACL** takes on **responsibility** of **synchronization** between contexts.

Low-Tech Synchronizing ACL

“The Nightly Batch Script”

Low-Tech Synchronizing ACL

- Often implemented as some kind of worker or “Nightly Batch Script”
- Freshness of data depends on how often the worker is run

Low-Tech Synchronizing ACL

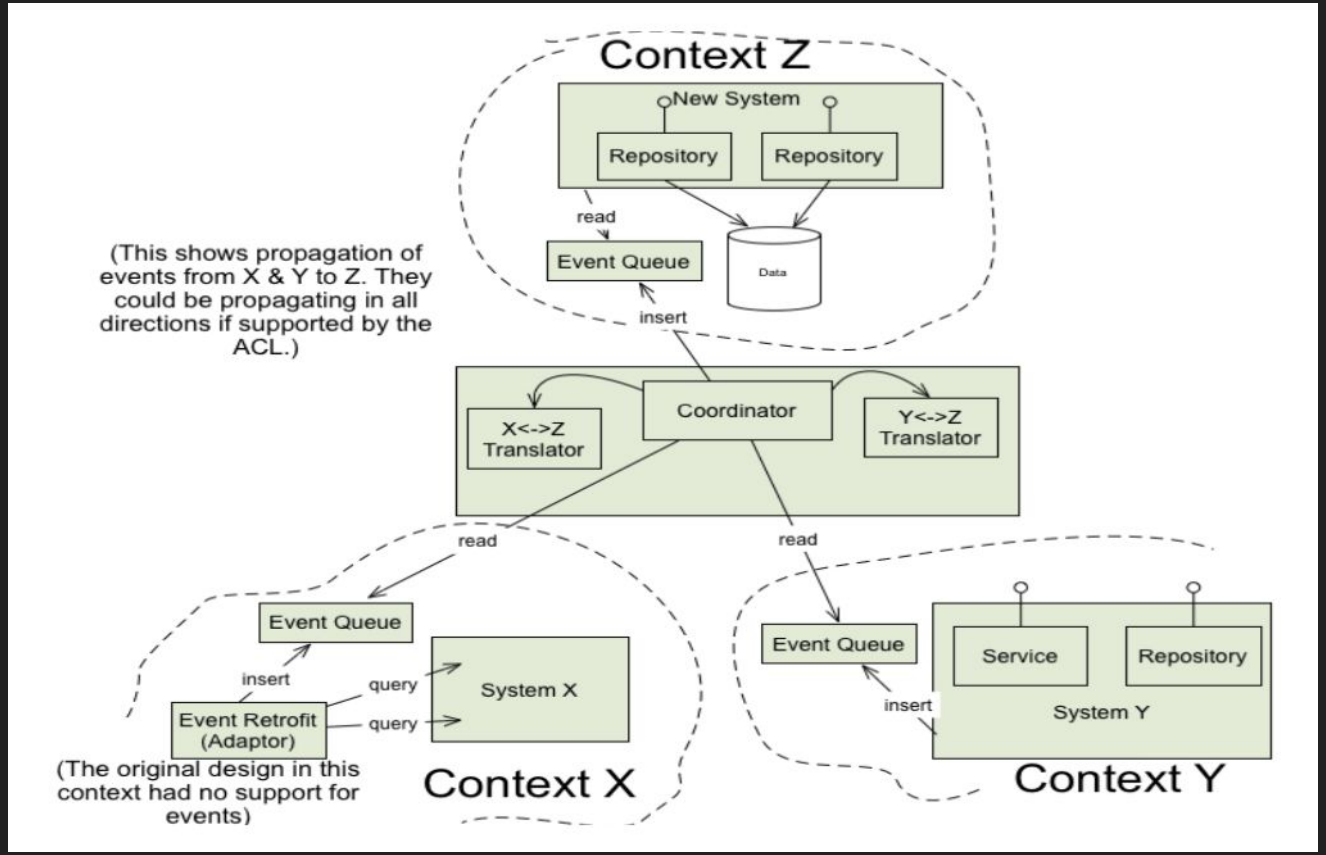
Pitfalls

- Isolate data copying and translation for your context in modules
- Don’t put “once-per-day” business logic into the ACL

Stylish Synchronizing ACL

Messaging And Domain Events

Stylish Synchronizing ACL



Stylish Synchronizing ACL

- Updates the system while it’s being used
- Integrates nicely with event driven architectural patterns

“**Any mechanism that can update a data store** in one context based on data in another [..] and can do this **asynchronously** [..] **could be used to implement a Synchronizing ACL**.”

Things To Remember From This Talk

- Bubble Context enables DDD
- Anticorruption Layer enables Bubble Context
- Translation is not mapping things

References & Related Sources

Getting Started With DDD When Surrounded By Legacy Systems (Eric Evans):
http://domainlanguage.com/wp-content/uploads/2016/04/GettingStartedWithDDDWhenSurroundedByLegacySystemsV1.pdf

Domain Driven Design (Eric Evans):
https://www.safaribooksonline.com/library/view/domain-driven-design-tackling/0321125215/

Implementing Domain Driven Design (Vaughn Vernon):
https://www.safaribooksonline.com/library/view/implementing-domain-driven-design/9780133039900/

Refactoring code that accesses external services (Martin Fowler):
http://www.martinfowler.com/articles/refactoring-external-service.html is an nice example how to refactor service access into an ACL

Architecture - The Lost Years (Uncle Bob):
https://www.youtube.com/watch?v=WpkDN78P884 explains why context matters (and the framework must not be important)