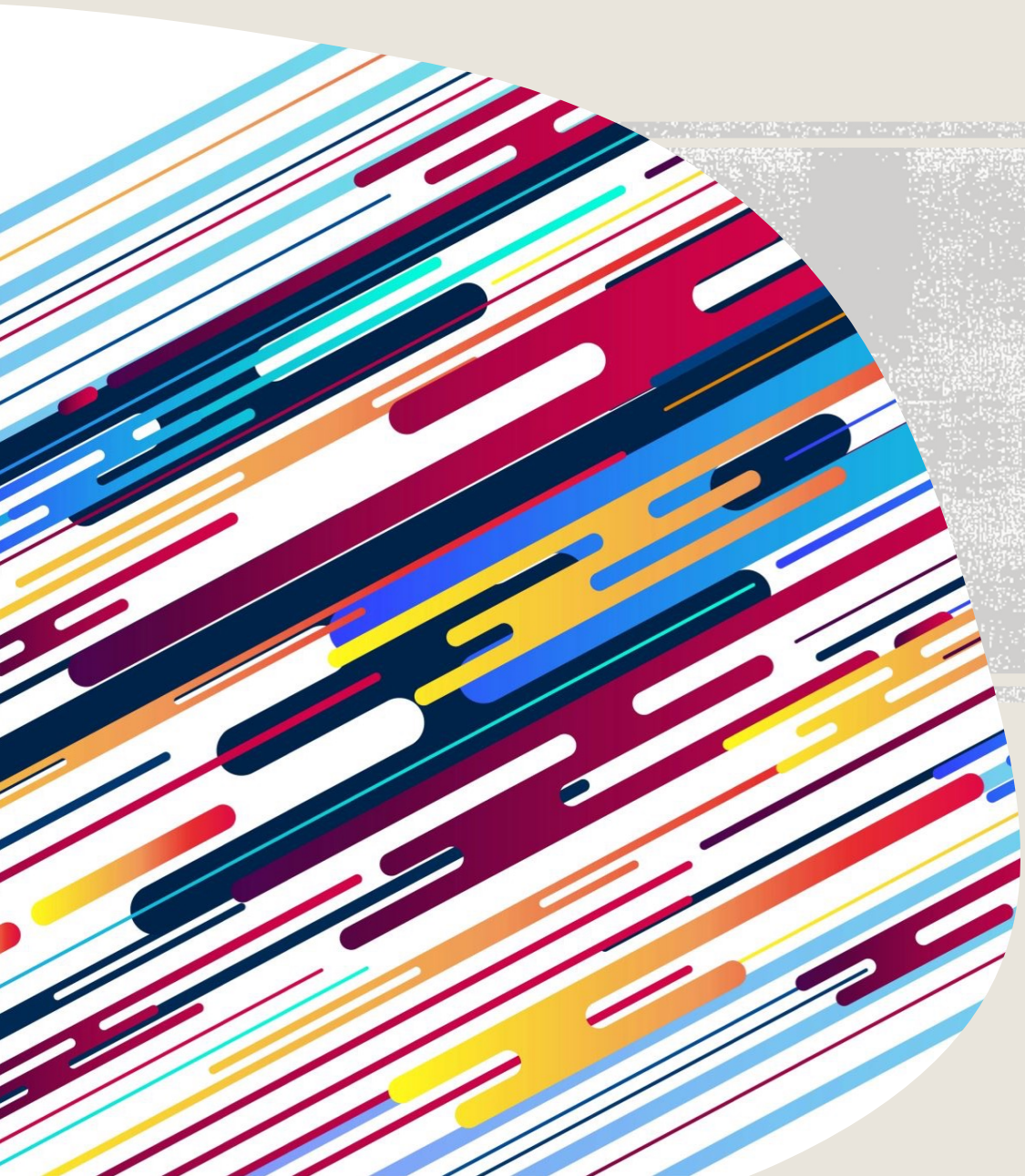
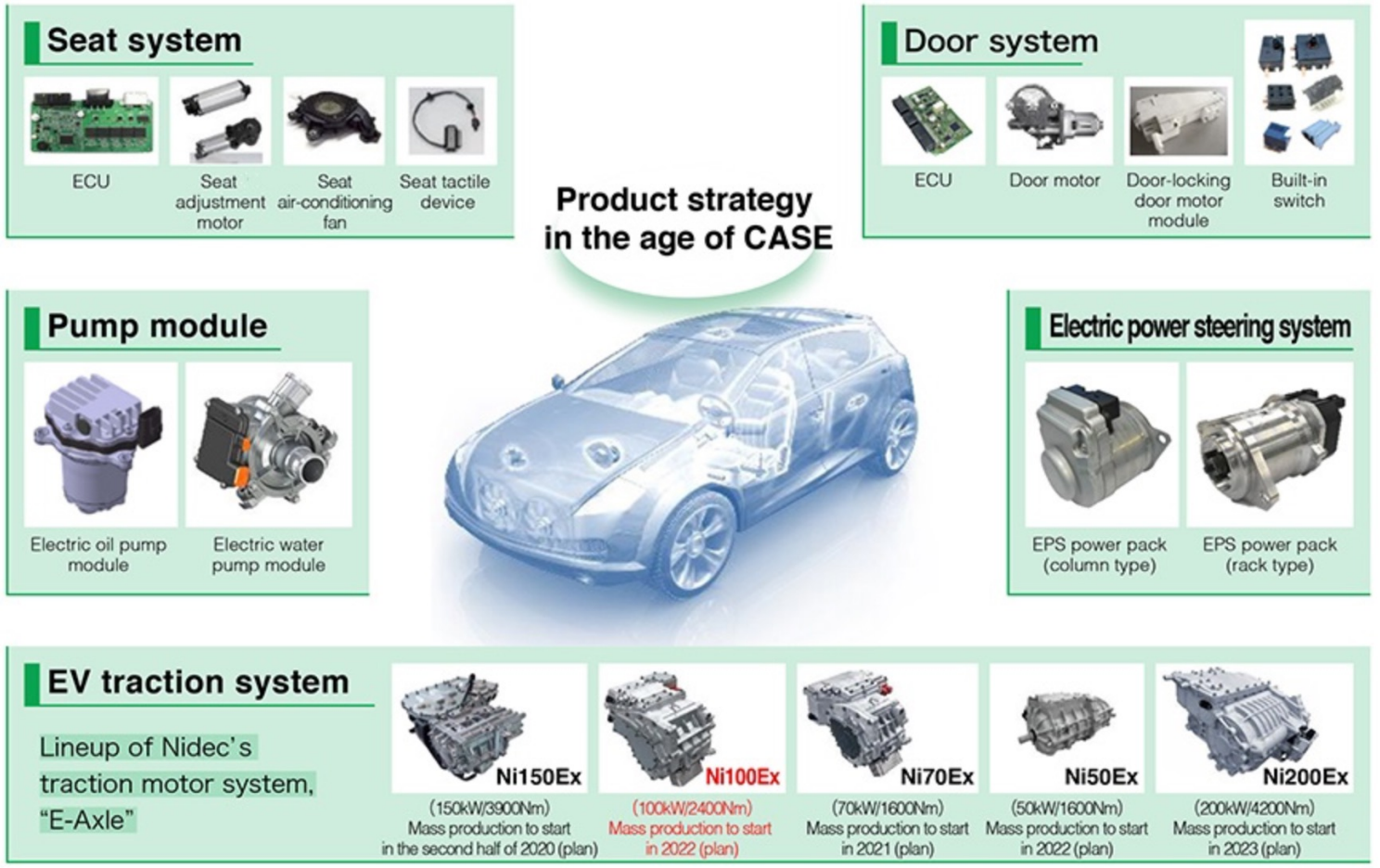


DOMAIN DRIVEN DESIGN (DDD)

A great car starts withj a good vision



Not only a functional car, but also a great car

Creafully write spec

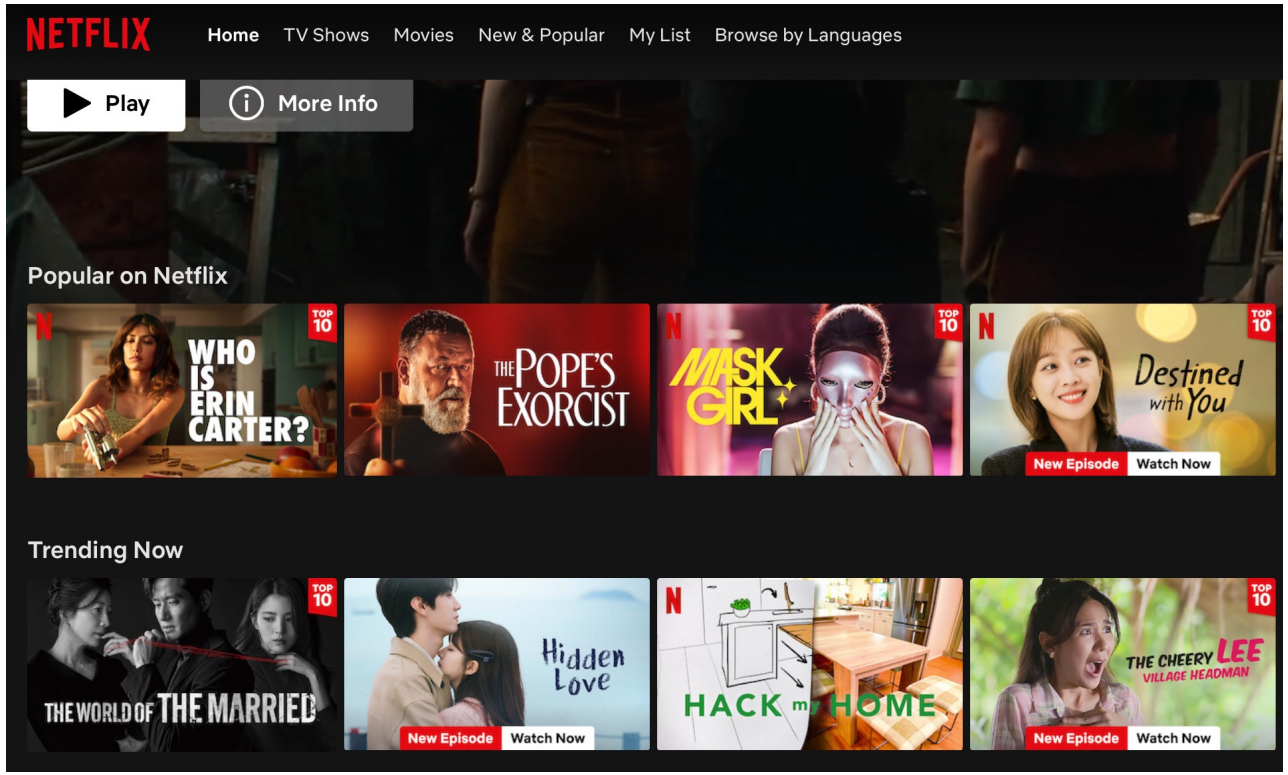
Lost of designs

https://www.nidec.com/en/technology/new_field/ev-platform/

WHAT DDD?

DDD focuses on modelling the software that match the business's core domain

The software design will correspond to the input from domain experts



What is NETFLIX' core domain?

SOFTWARE DEVELOPMENT PROCESS

- In order to create a good software, you have to know what that software is all about.
- You cannot create a banking software system unless you have a good understanding of what banking is all about.
- One must understand the *domain* of banking.
- Software needs to incorporate the core concepts and elements of the domain, and to precisely realize the relationships between them.
- Software which does not have its roots planted deeply into the domain will not react well to change over time.

ต้องเข้าใจระบบนั้น ๆ

ยิ่งเข้าไจมาก ก็ยิ่งออกแบบได้ดีและต่อยอดได้ง่าย

How about designing a software for figth-control system?

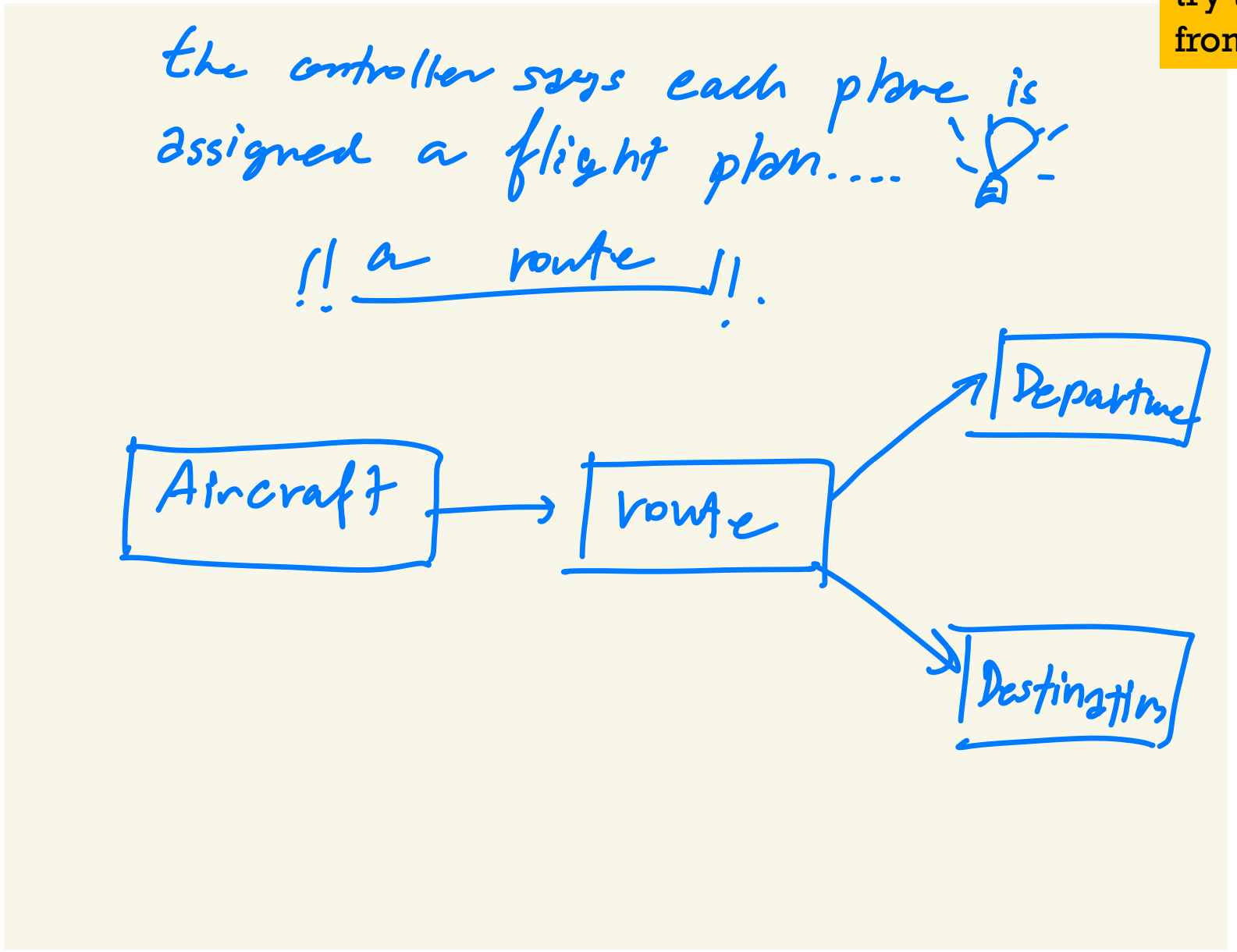
What do we need to know?

Develop ideas and
try to extract knowledge
from domain experts

An airplane flight control system

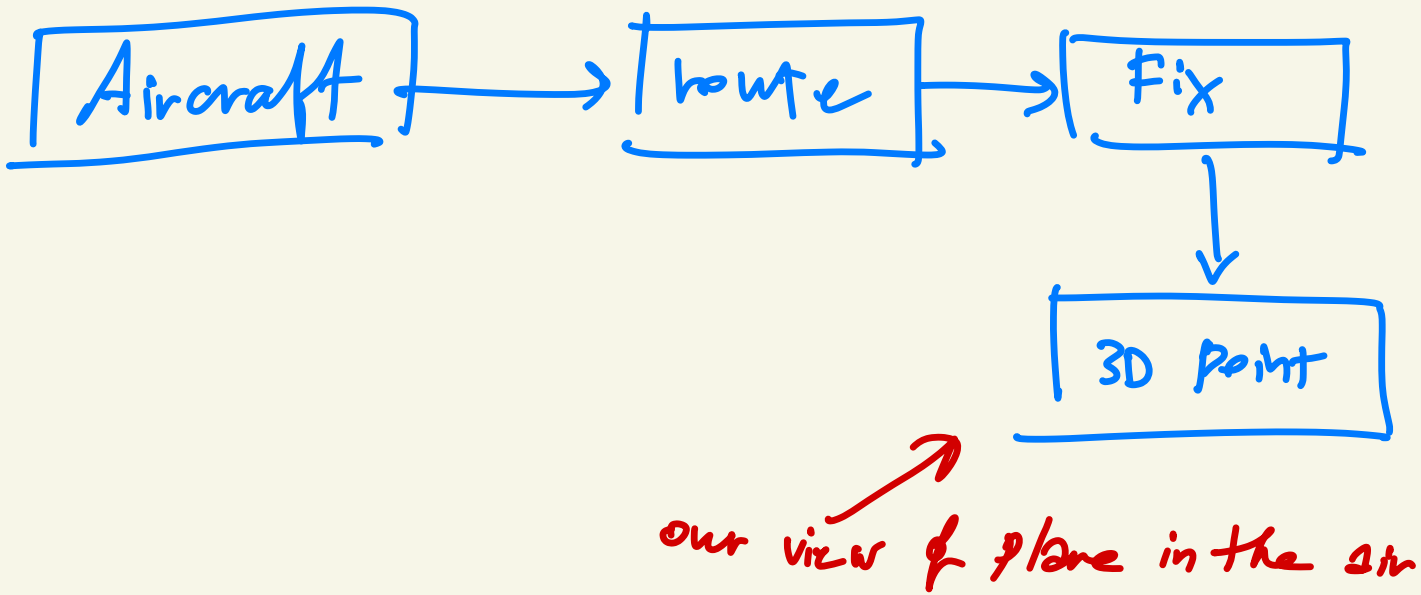
Vocabs: taking off, landing, mid air, danger of collision, etc....

Develop ideas and
try to extract knowledge
from domain experts



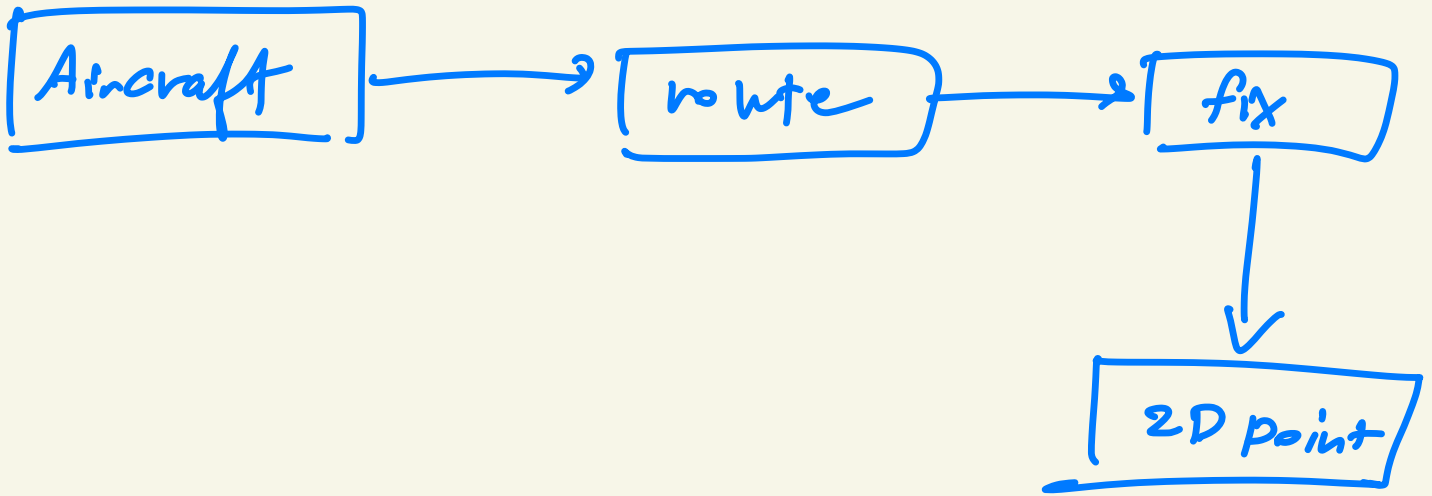
Develop ideas and
try to extract knowledge
from domain experts

the route is made up of small segments, which put together constitute some sort of crooked line from departure to destination.



Develop ideas and
try to extract knowledge
from domain experts

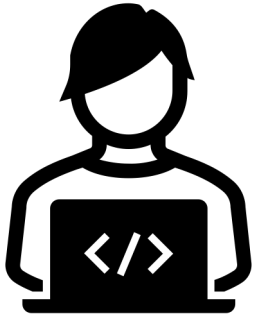
So, revise the diagram ....



Controllers see points as projection
on earth ...

Ubiquitous languages

The need for a common language



Classes, methods, algorithms, full stack, front-end, backend, Patterns, SQL, NoSQL, cloud, Token, ….

Planes, routes, altitudes, latitude, longitude, deviances from the normal route, plane trajectories….

UBIQUITOUS LANGUAGE CONCEPTS

- A core principle of domain-driven design is to use a language based on the model.
- Language does not appear overnight.
- We need to find key concepts which define the domain and design and find corresponding words for them and start using them consistently.

UBIQUITOUS LANGUAGE IN FLIGHT CONTROL SYSTEM

What is the ubiquitous language in insurance?

Deductible,
High deductible health plan,
Health saving account,
Premium,
Copayment,
Coinsurance,
Out-of-pocket maximum,
HMO (Health Maintenance Organization),
POS (Point of service),
PPO (preferred provider organization)

UBIQUITOUS LANGUAGE IN FLIGHT CONTROL SYSTEM

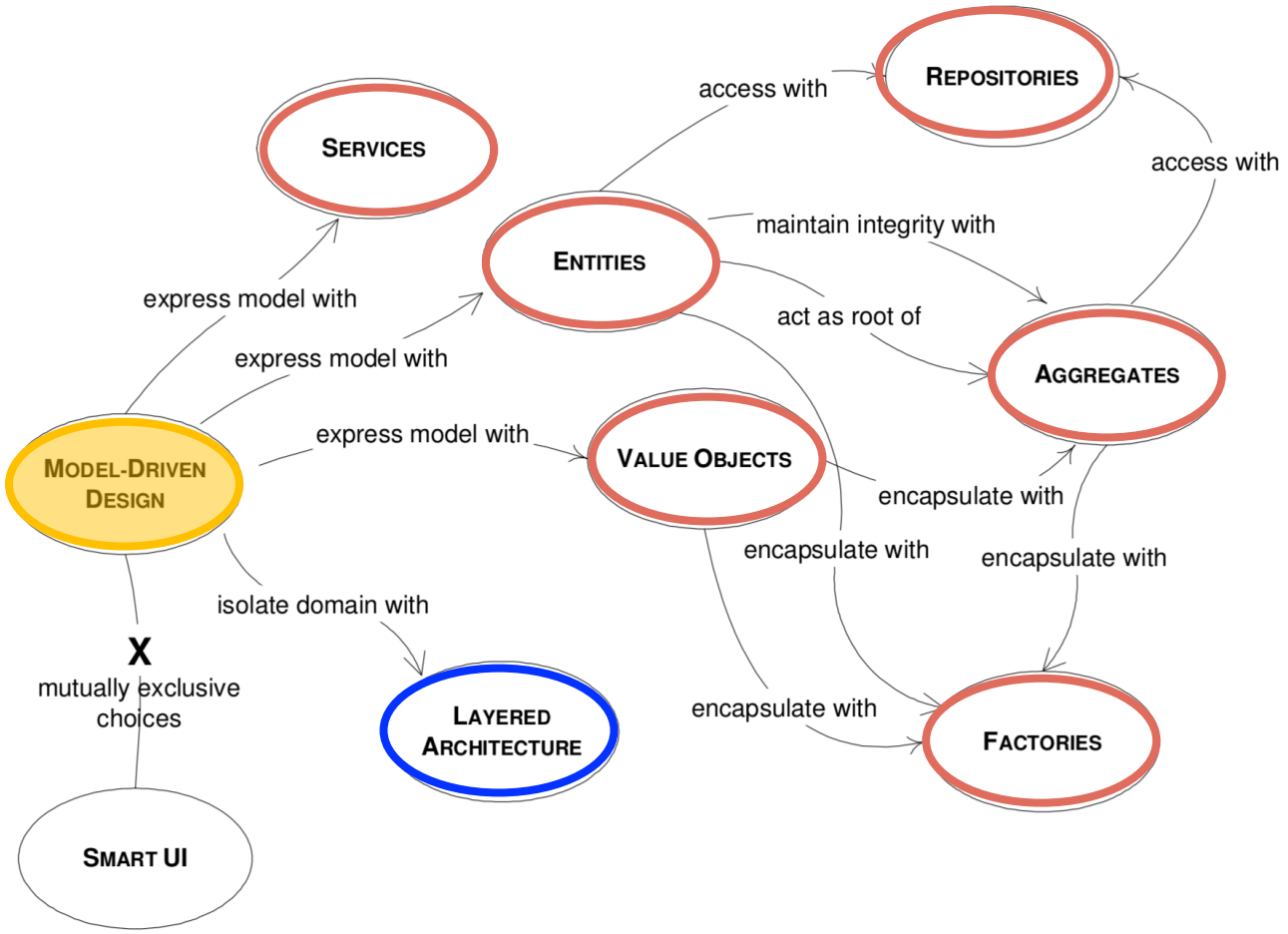
What is the ubiquitous language your term project?

Model-Driven Design

The goal of domain-driven design is to create better software by focusing on a model of the domain rather than the technology.

Avram A., Marinescu F., Domain-Driven Design Quickly: A summary of Eric Evans' Domain Driven Design
https://www.lulu.com/content/325231?page=1&pageSize=4

Eric Evans. Domain Driven Design. 2003



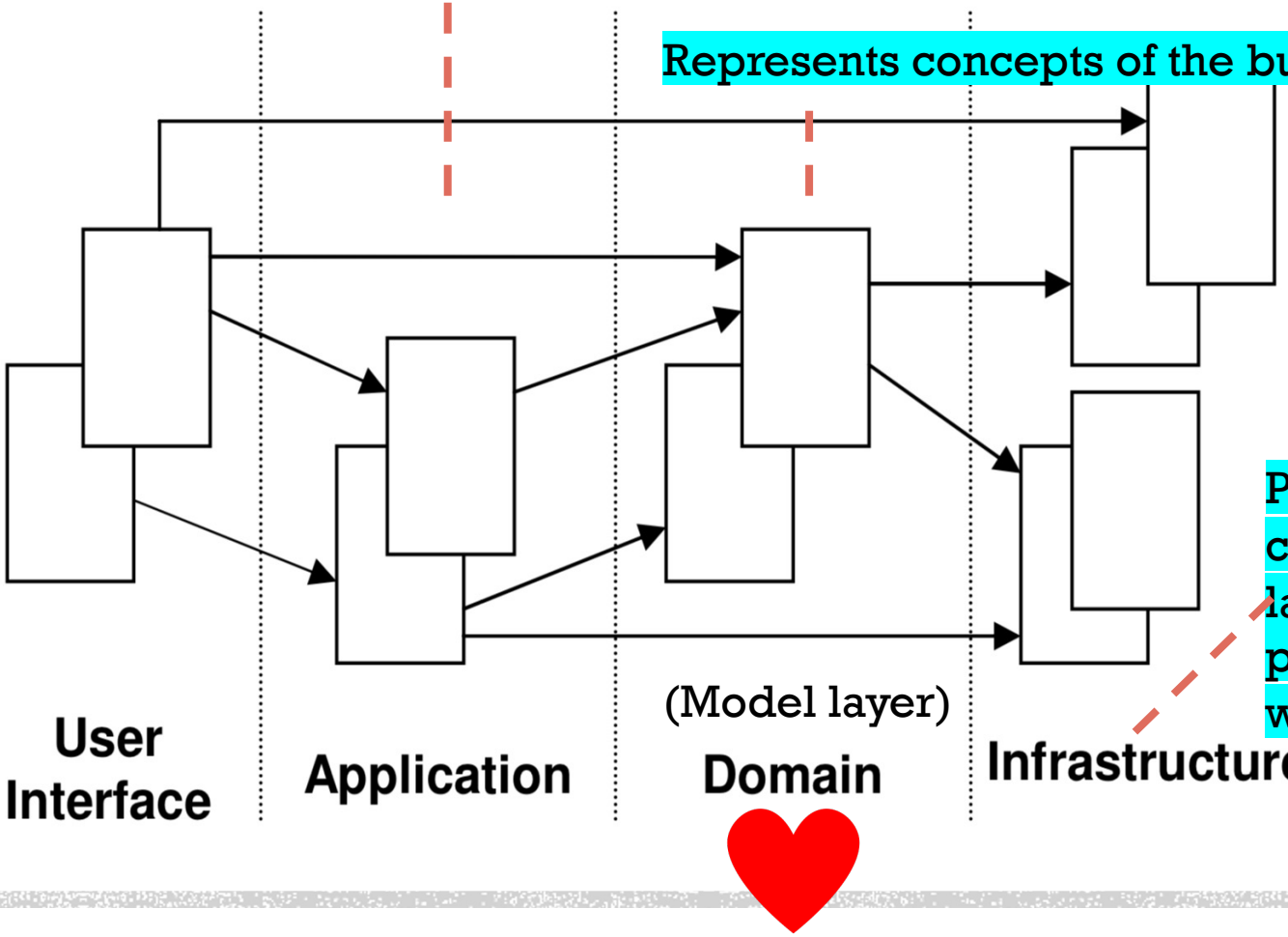
THE BUILDING BLOCKS OF A MODEL-DRIVEN DESIGN

Avram A., Marinescu F., Domain-Driven Design Quickly: A summary of Eric Evans' Domain Driven Design
https://www.lulu.com/content/325231?page=1&pageSize=4

A common architectural solution for domain-driven designs contain four conceptual layers:

Coordinates tasks and delegates work to collaboration of domain objects in the domain layer

Represents concepts of the business



LAYERED ARCHITECTURE

Provides generic technical capabilities that support higher layers, e.g., message sending, persistence for the domain, drawing widgets

Avram A., Marinescu F., Domain-Driven Design Quickly: A summary of Eric Evans' Domain Driven Design
https://www.lulu.com/content/325231?page=1&pageSize=4

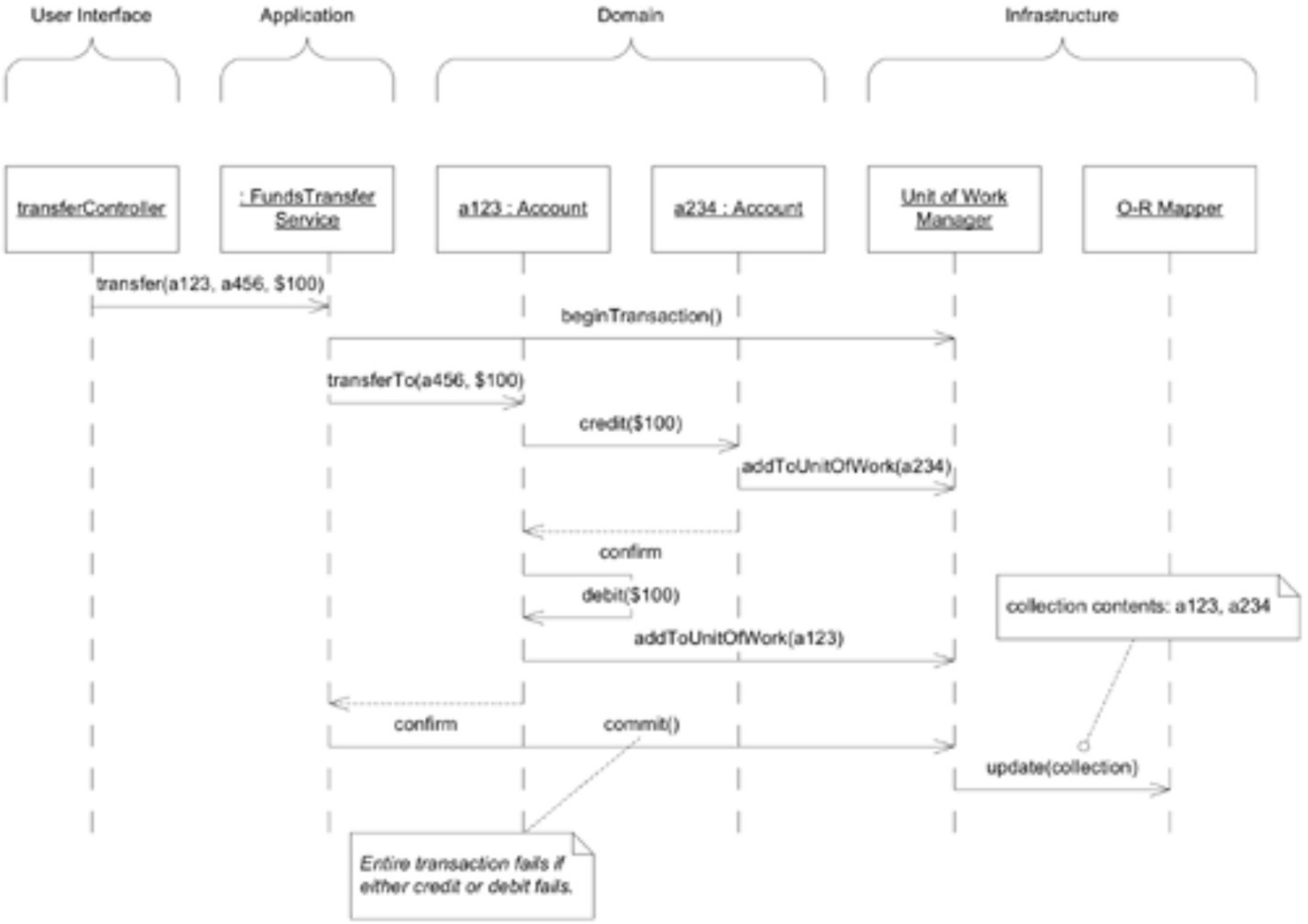


Figure 4.1. Objects carry out responsibilities consistent with their layer and are more coupled to other objects in their layer.

Eric Evans. Domain Driven Design. 2003

ENTITIES

- A category of objects which seem to have an identity, which remains the same throughout the states of the software.
- For these objects it is not attributes which matters but a thread of continuity and identity, which spans the life of a system.
- Examples: Plane, Flight plan, Route, Fix, Airport

What is entities in RegChula system?

What is entities in a match maker system?

Analysis Activities: Identifying Entity Objects

- Entity object represents persistent information tracked by the system
- Participating objects are found by examining each use case and identifying candidate objects
- Use natural language analysis to identify objects, attributes, and associations from a requirements specification
- Identify objects' attributes and their responsibilities
- Heuristics for identifying entity objects
  - terms that developers or users need to clarify in order to understand the use case
  - recurring nouns in the use case (e.g., Incident)
  - real-world entities that the system needs to track (e.g., FieldOfficer, Dispatcher, Resource)
  - real-world activities that the system needs to track (e.g., EmergencyOperationsPlan)
  - data sources or sinks (e.g., Printer)

Entity → Persistent Class (stored in DB)

Role(subject), (Object)

Thing, Place,

Transaction, Event, ..

Etc.

True or False

1. **Entities** ปรากฏได้มากกว่า **1 use cases in BCE**
2. DDD พิจารณา **entities** จากมุมมอง **system decomposition into bounded contexts**
3. BCE เป็นมุมมองจาก **use cases**

VALUE OBJECTS

- An object used to describe certain aspects of a domain, and which does not have identity.
- It is recommended to select as entities only those objects conform to the entity definition and make the rest of the objects to be Value Objects.
- It is highly recommended that value objects are immutable.
- Examples: Air travel booking system create object for each flight, the flight code can be shared among the customers, who book the same flight.

Is “Address” a V*ALUE OBJECT*? Who's Asking?

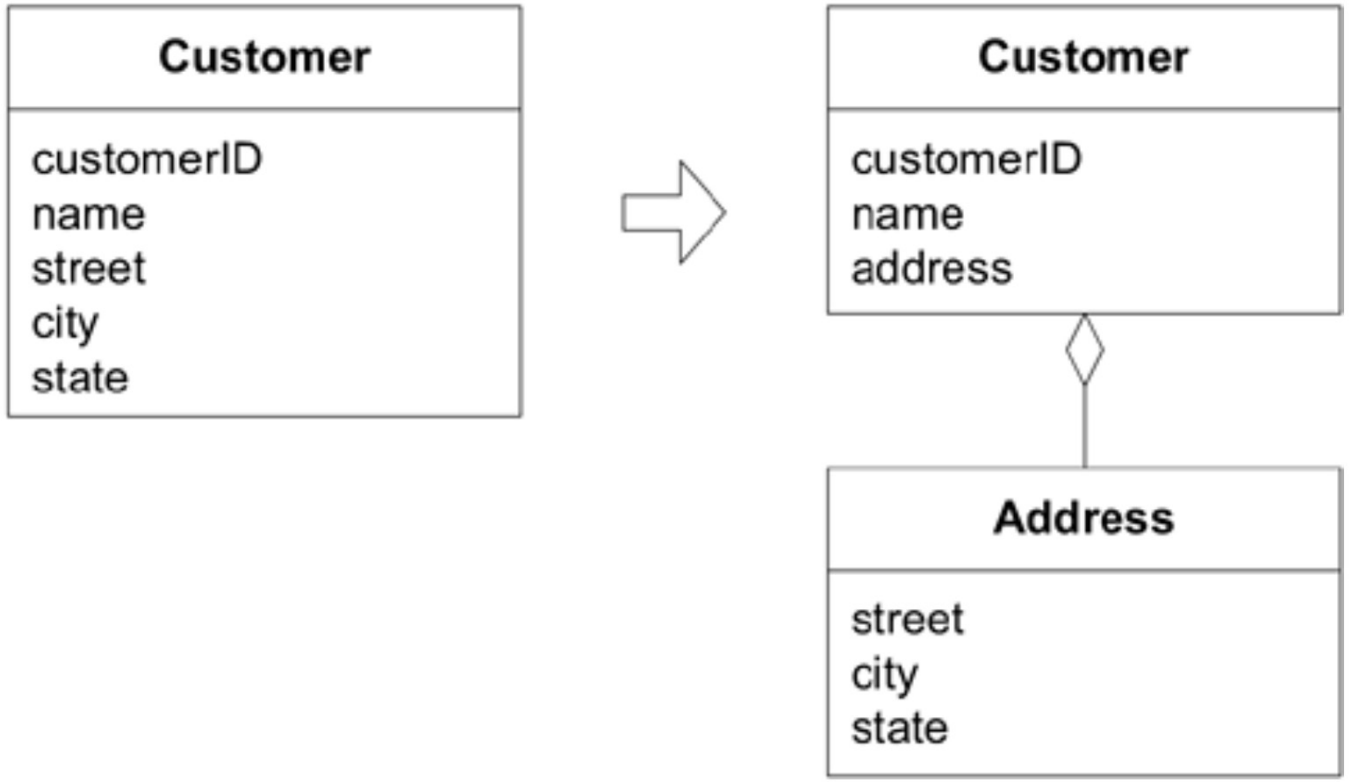
In software for a mail-order company, an address is needed to confirm the credit card, and to address the parcel. But if a roommate also orders from the same company, it is not important to realize they are in the same location. Address is a *VALUE OBJECT*.

In software for the postal service, intended to organize delivery routes, the country could be formed into a hierarchy of regions, cities, postal zones, and blocks, terminating in individual addresses. These address objects would derive their zip code from their parent in the hierarchy, and if the postal service decided to reassign postal zones, all the addresses within would go along for the ride. Here, Address is an *ENTITY*.

In software for an electric utility company, an address corresponds to a destination for the company's lines and service. If roommates each called to order electrical service, the company would need to realize it. Address is an *ENTITY*. Alternatively, the model could associate utility service with a “dwelling,” an *ENTITY* with an attribute of address. Then Address would be a *VALUE OBJECT*.

Colors are an example of VALUE OBJECTS in base libraries in many modern development systems.

Eric Evans. Domain Driven Design. 2003



ENTITY CAN CONTAIN VALUE OBJECTS

Eric Evans. Domain Driven Design. 2003

AGGREGATES

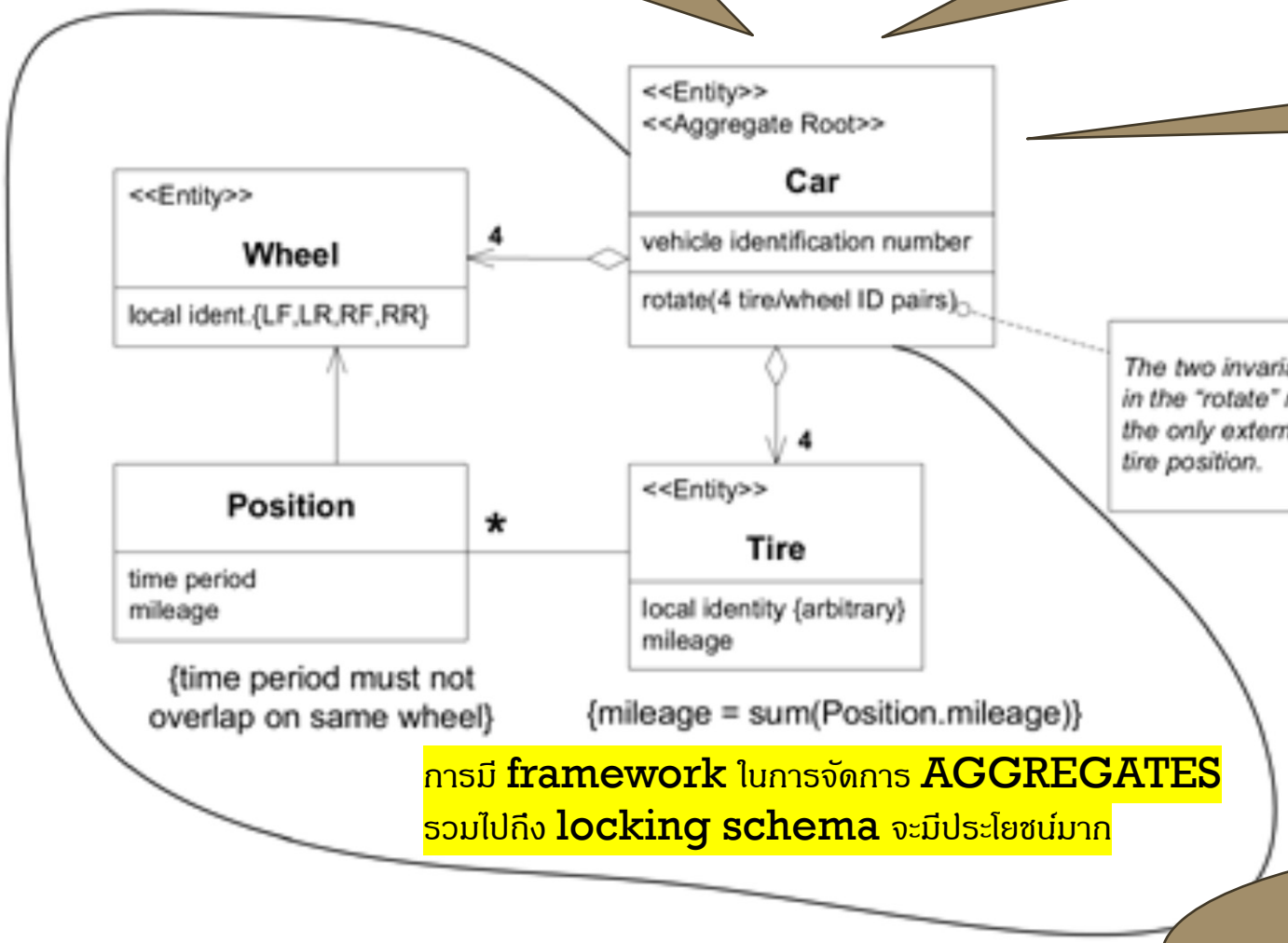
- A group of associated objects which are considered as one unit regarding data changes.
- The Aggregate is demarcated by a boundary which separates the objects inside from outside.
- Each Aggregate has one root, an Entity, the only accessible object from outside.

Root ENTITY has global identity

Root ENTITY is responsible to check invariants

A change within the AGGREGATE boundary need to satisfy invariants

EXAMPLE OF AN AGGREGATE



การมี **framework** ในการจัดการ **AGGREGATES** รวมไปถึง **locking schema** จะมีประโยชน์มาก

Figure 6.3. AGGREGATE invariants

A delete operation must remove everything within the AGGREGATE boundary

Eric Evans. Domain Driven Design. 2003

EXAMPLE OF AN AGGREGATE

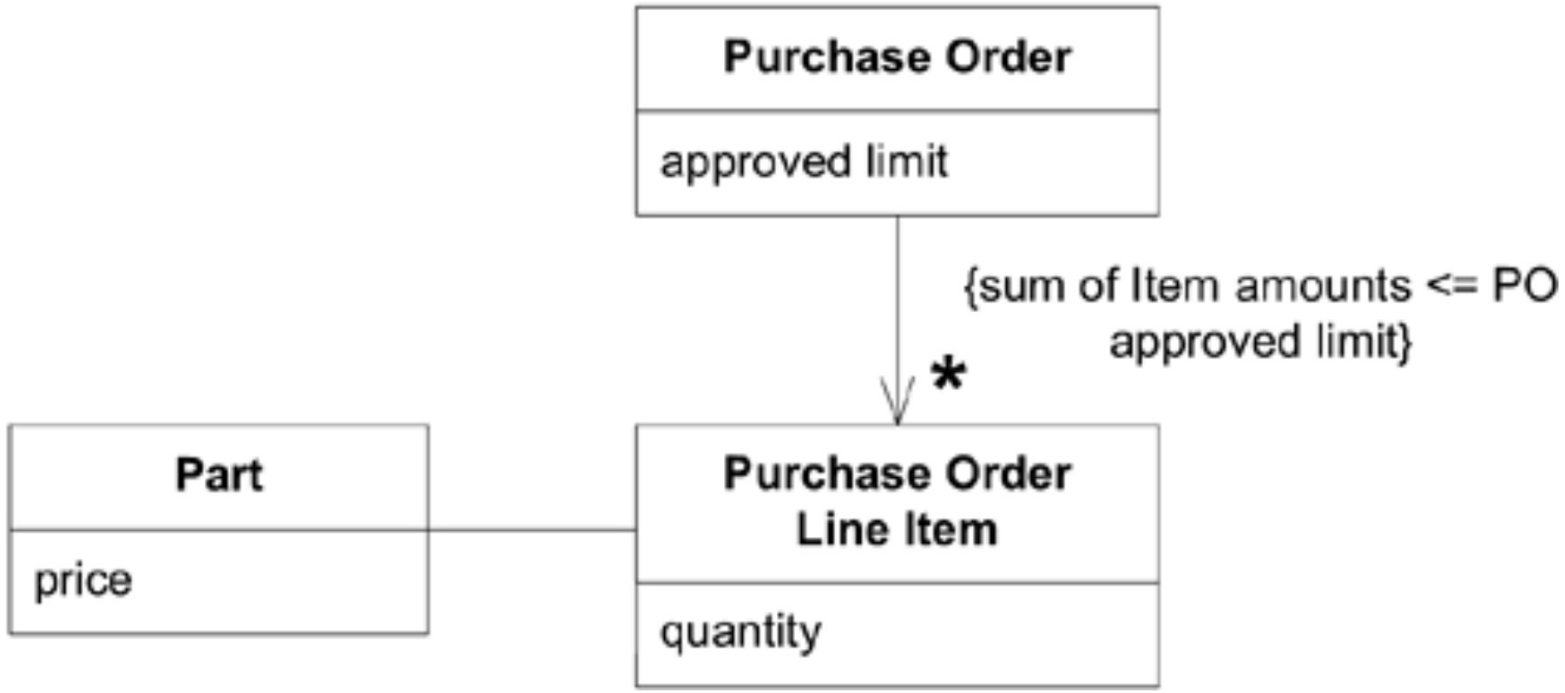


Figure 6.4. A model for a purchase order system

Eric Evans. Domain Driven Design. 2003

AGGREGATES

PO #0012946 Approved Limit: $1,000.00

Item #	Quantity	Part	Price	Amount
001	3	Guitars	@ 100.00	300.00
002	2	Trombones	@ 200.00	400.00
			Total:	700.00

Original PO

These individual edits cause no errors

Figure 6.5. The initial condition of the PO stored in the database

Each item editedsimultaneously

George adds guitars in his view

PO #0012946 Approved Limit: $1000.00

Item #	Quantity	Part	Price	Amount
001	**5**	Guitars	@ 100.00	**500.00**
002	2	Trombones	@ 200.00	400.00
			Total:	900.00

Amanda adds a trombone in her view

PO #0012946 Approved Limit: $1,000.00

Item #	Quantity	Part	Price	Amount
001	3	Guitars	@ 100.00	300.00
002	**3**	Trombones	@ 200.00	**600.00**
			Total:	900.00

Figure 6.6. Simultaneous edits in distinct transactions

Eric Evans. Domain Driven Design. 2003

AGGREGATES

PO #0012946 Approved Limit: **$1,000.00**

Item #	Quantity	Part	Price	Amount
001	5	Guitars	@ 100.00	500.00
002	3	Trombones	@ 200.00	600.00
			Total:	**1,100.00**

Figure 6.7. The resulting PO violates the approval limit (broken invariant).

How can we solve this?

The merged of these edits however violates the limit!!!.

Eric Evans. Domain Driven Design. 2003

AGGREGATES

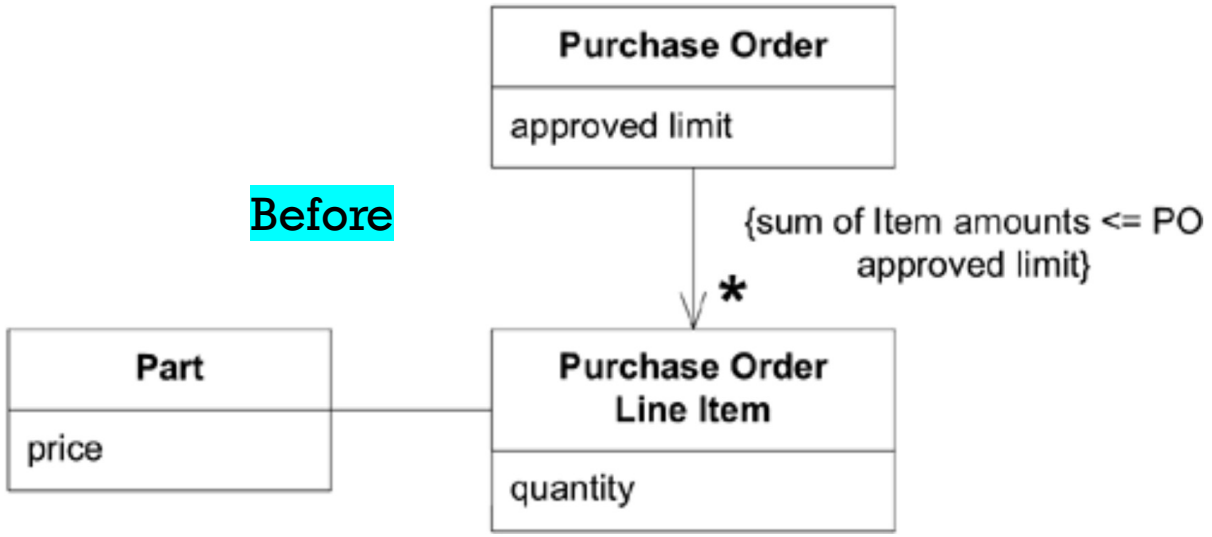


Figure 6.4. A model for a purchase order system

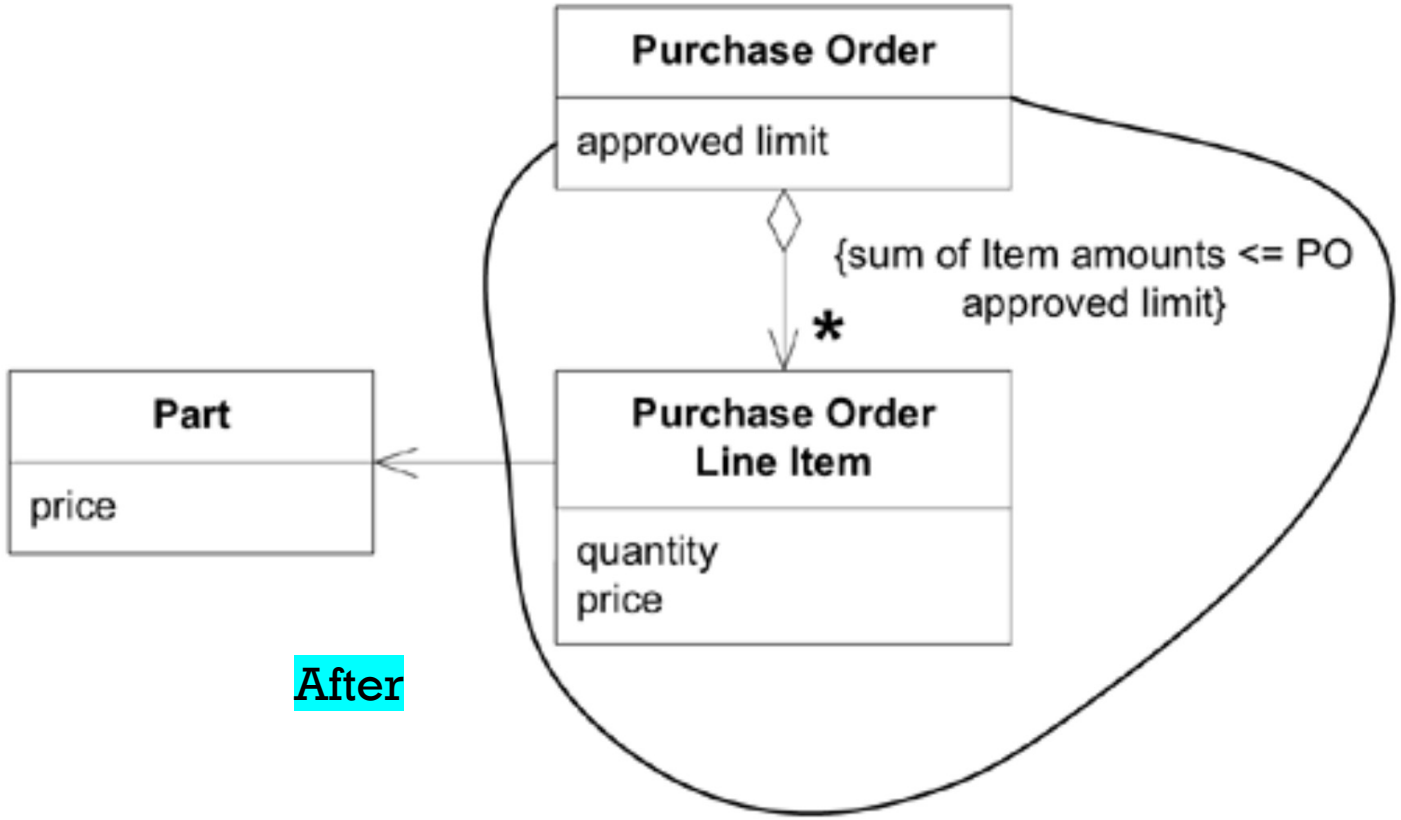


*Figure 6.11. Price is copied into **Line Item**. AGGREGATE invariant can now be enforced.*

https://levelup.gitconnected.com/practical-ddd-in-golang-aggregate-de13f561e629

Eric Evans. Domain Driven Design. 2003

SERVICES

- Service is used to represent behavior within a domain.
- It can group related functionalities which serves the Entities and Value objects.
- It is much better to declare service explicitly.
- Services act as interfaces which provide operations, which should be stateless.
- Services provide a point of connection for many objects.

Eric Evans. Domain Driven Design. 2003

FACTORIES

- Factories are used to encapsulate the knowledge necessary for object creation, and they are especially useful to create Aggregates.
- Factories is responsible to create other objects.
- For immutable Value Objects, all attributes are initialized to their valid state.

Eric Evans. Domain Driven Design. 2003

Add elements inside a preexisting AGGREGATE FACTORY METHOD on the root of the AGGREGATE.

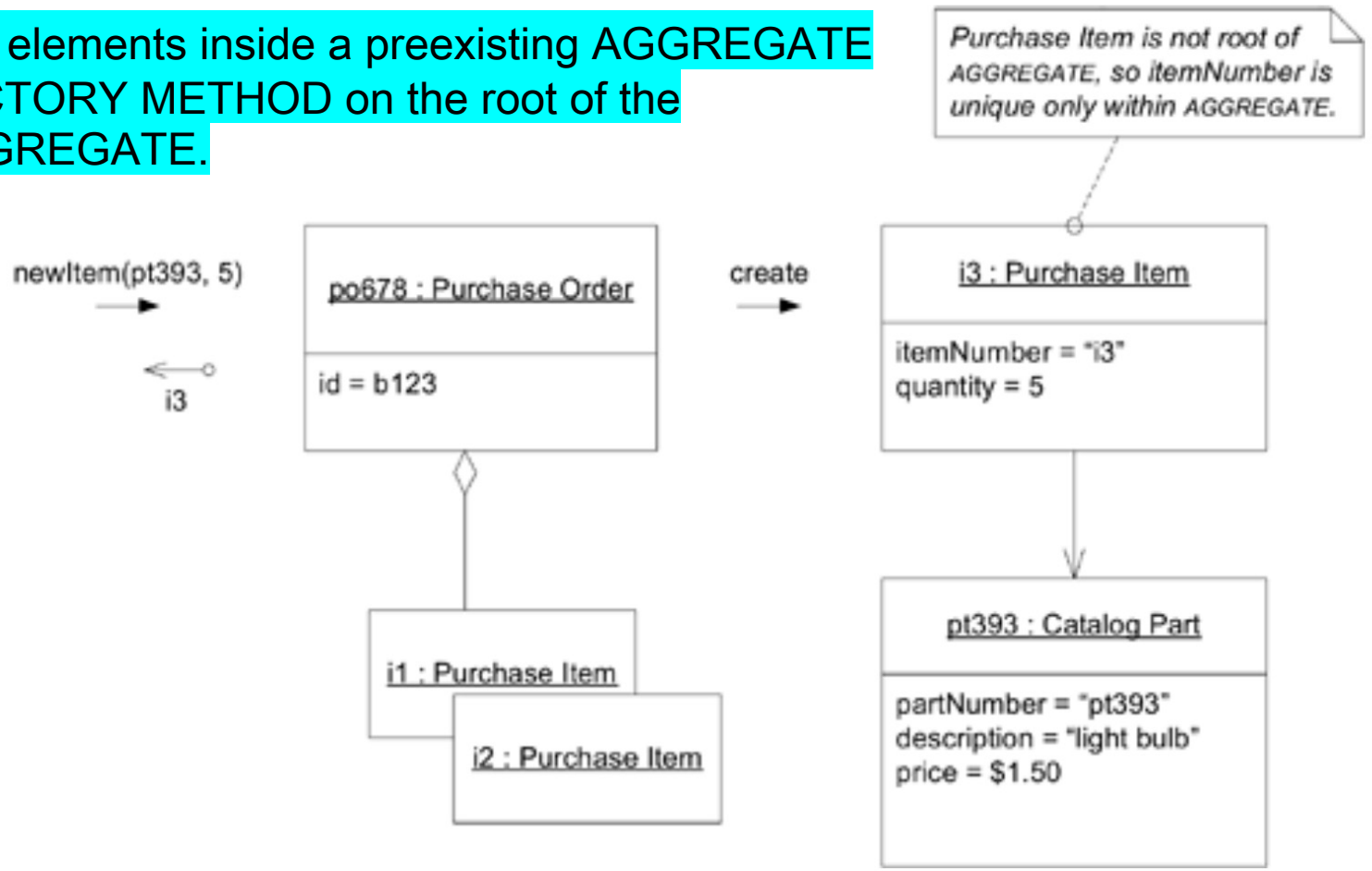


Figure 6.13. A FACTORY METHOD encapsulates expansion of an AGGREGATE.

FACTORIES AND THEIR SITES

Eric Evans. Domain Driven Design. 2003

FACTORIES AND THEIR SITES

Allow an AGGREGATE (Brokerage account to create TradeOrders)

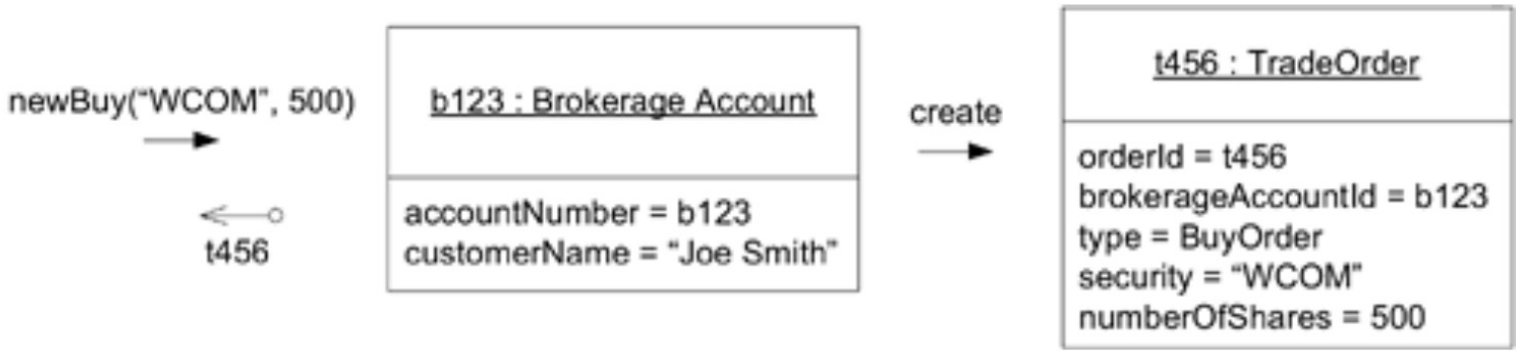


Figure 6.14. A FACTORY METHOD spawns an ENTITY that is not part of the same AGGREGATE.

Eric Evans. Domain Driven Design. 2003

FACTORIES AND THEIR SITES

A standalone FACTORY to build AGGREGATE

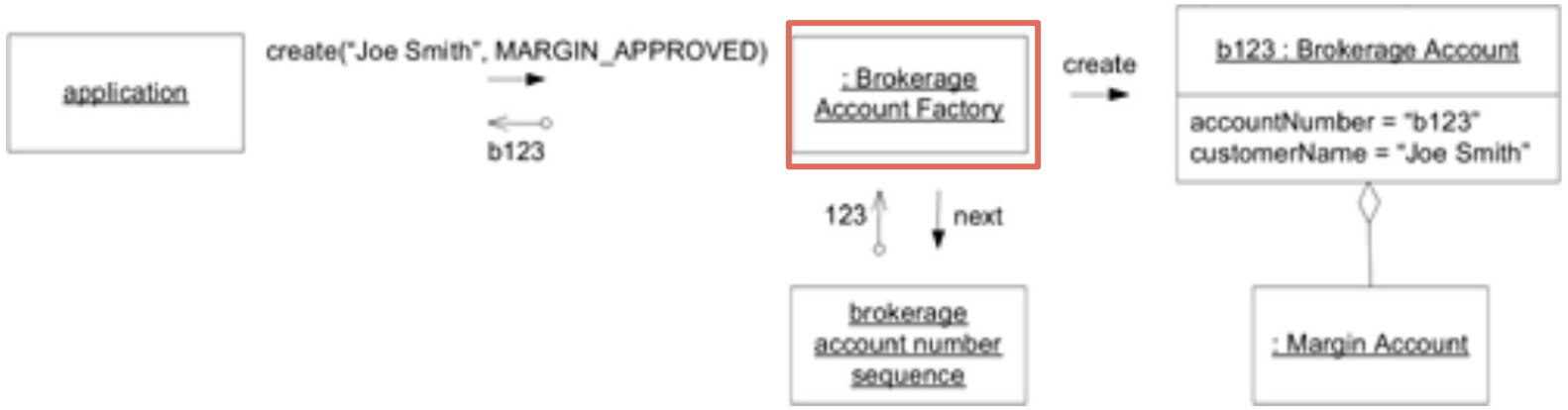


Figure 6.15. A standalone FACTORY builds AGGREGATE.

Eric Evans. Domain Driven Design. 2003

REPOSITORIES

"Exposure of technical infrastructure and database access mechanisms complicates the client and obscures the MODEL-DRIVEN DESIGN."

Repository pattern is a simple conceptual framework to capsulate those solutions and bring back our model focus…

Eric Evans. Domain Driven Design. 2003

REPOSITORIES

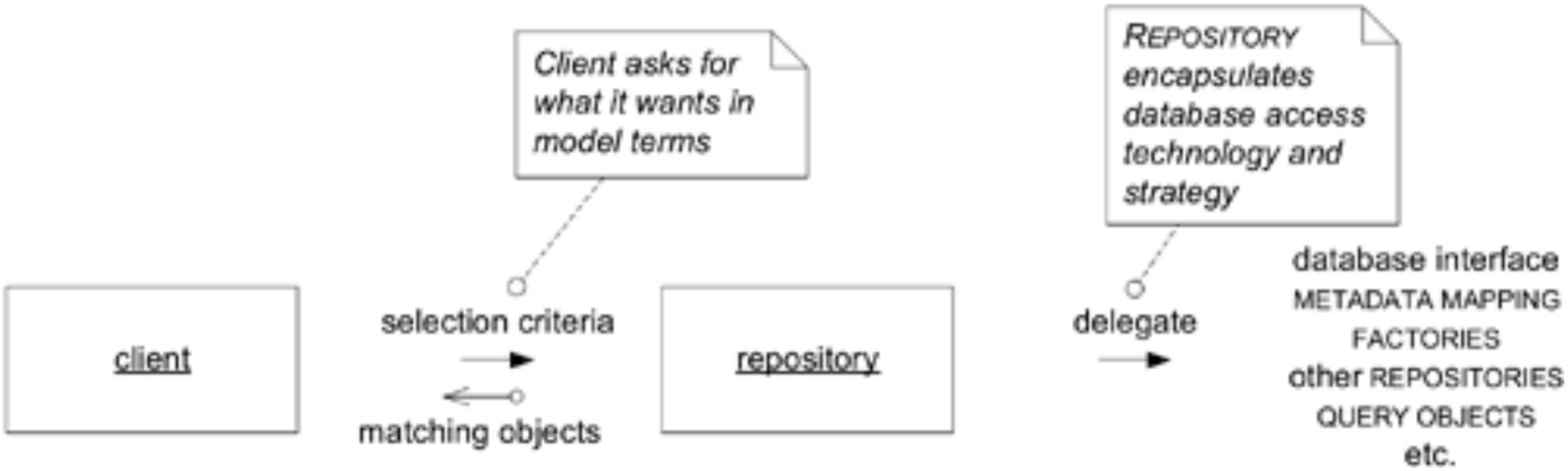


Figure 6.18. A REPOSITORY doing a search for a client

Eric Evans. Domain Driven Design. 2003

REPOSITORIES ADVANTAGES

- They present clients with a simple model for obtaining persistent objects and maintaining their life cycle.
- They decouple application and domain design from persistence technology, multiple database strategies, or even multiple data sources.
- They communicate design decisions about object access.
- They allow easy substitutions of a dummy implementation, for use in testing.

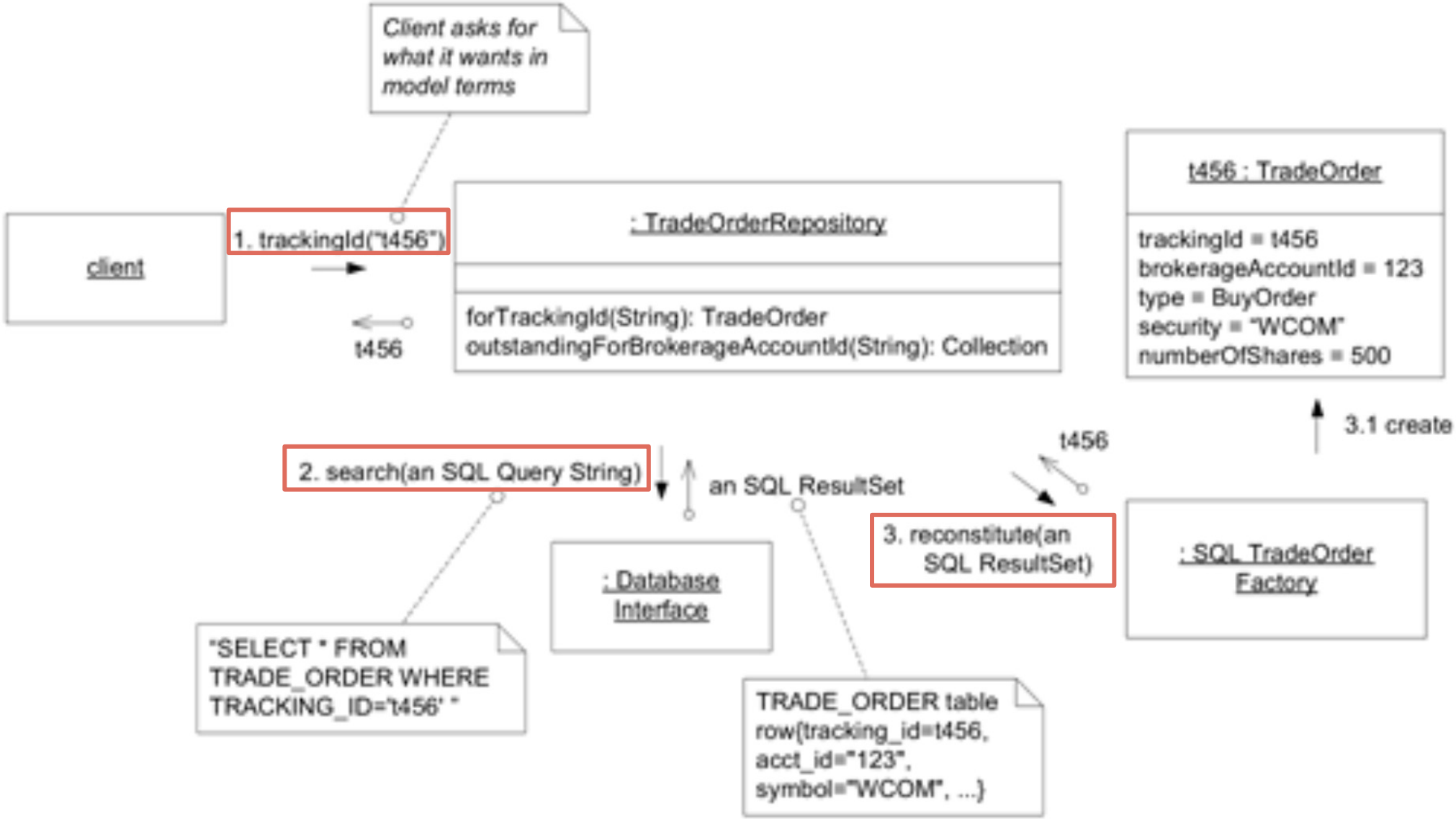
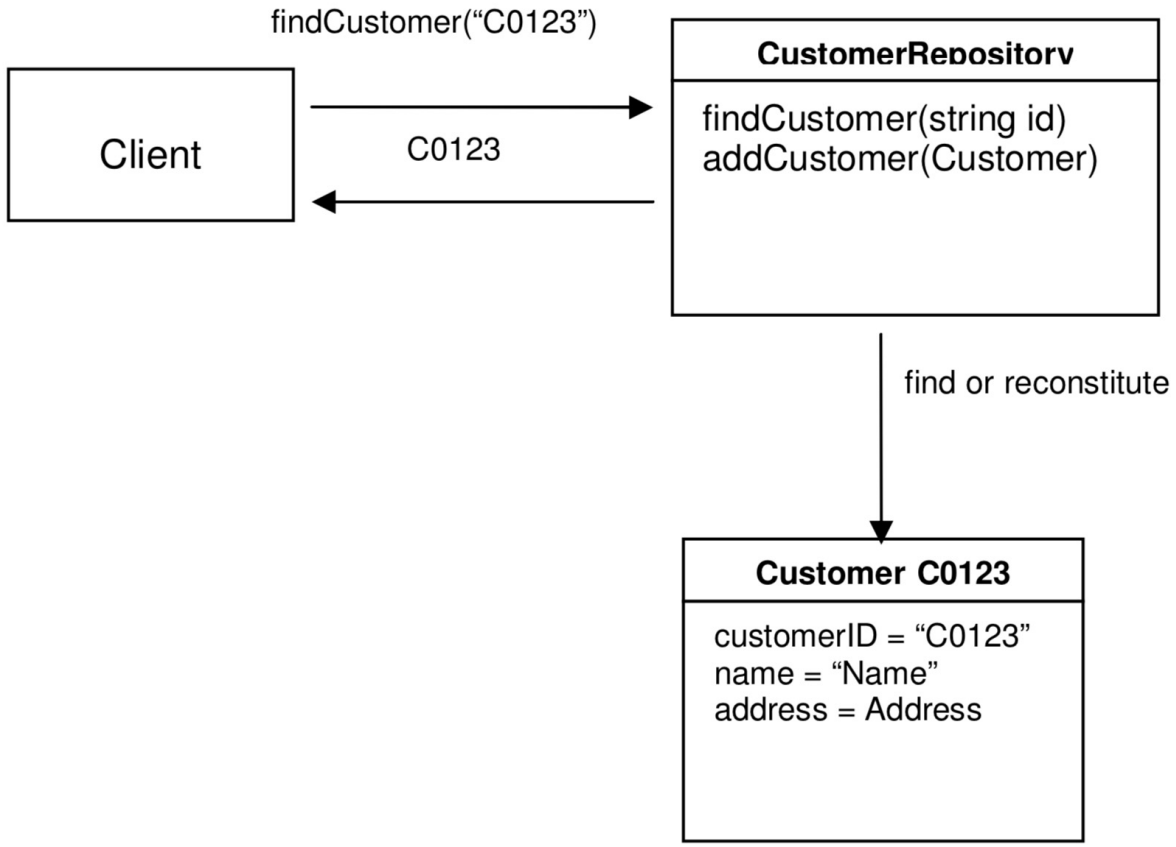


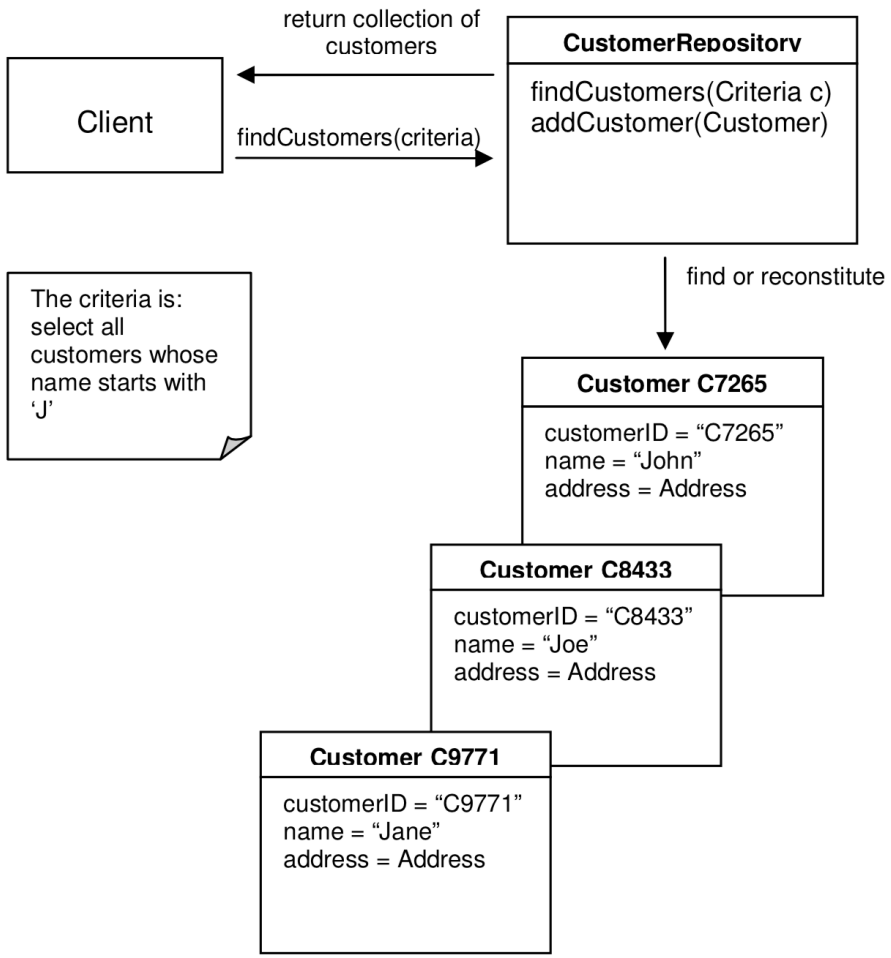
Figure 6.21. The REPOSITORY encapsulates the underlying data store.



REPOSITORIES

A repository encapsulates all the logic needed to obtain object references.

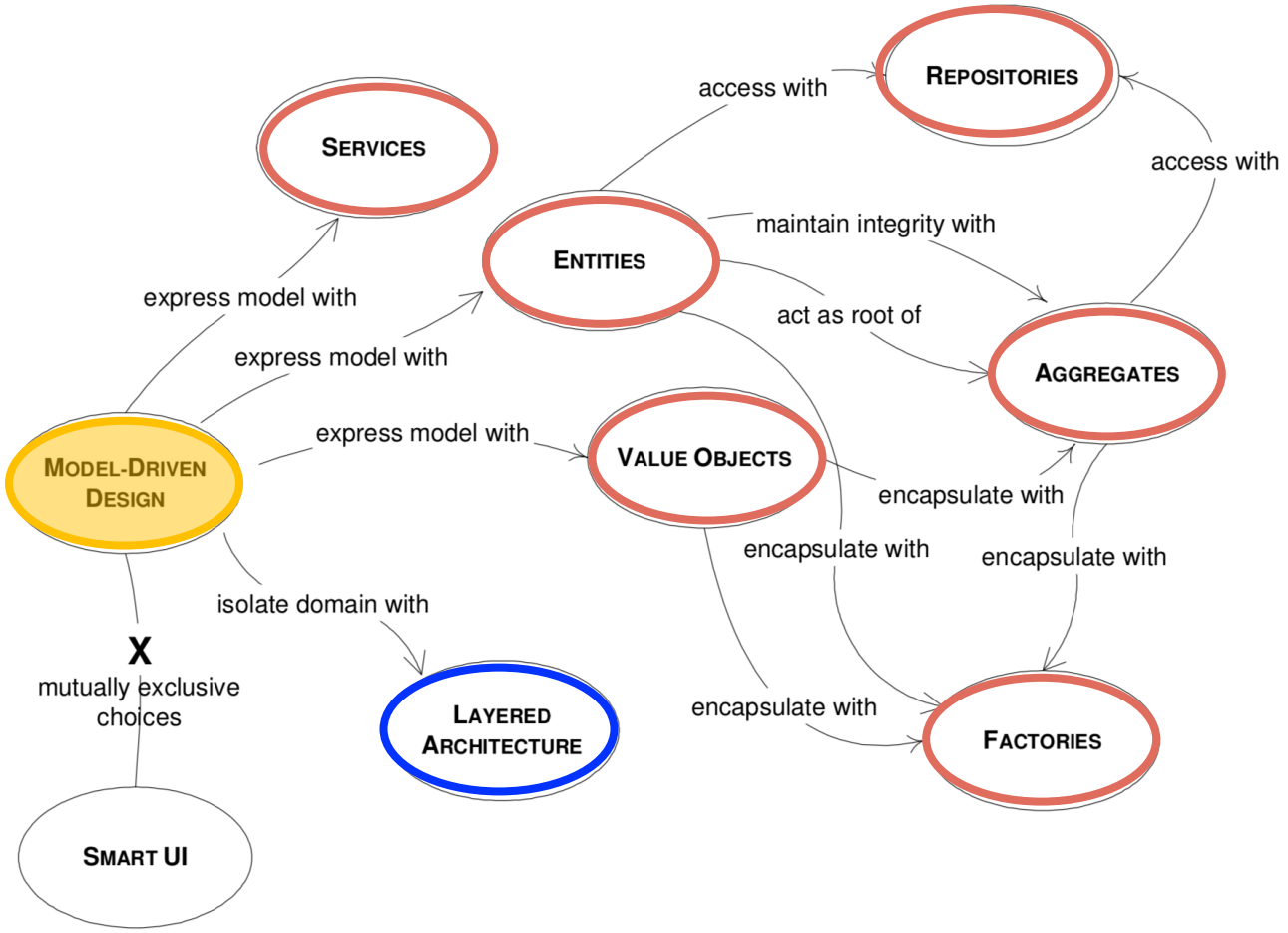
Avram A., Marinescu F., Domain-Driven Design Quickly: A summary of Eric Evans' Domain Driven Design
https://www.lulu.com/content/325231?page=1&pageSize=4



REPOSITORIES

A repository encapsulates all the logic needed to obtain object references.

Avram A., Marinescu F., Domain-Driven Design Quickly: A summary of Eric Evans' Domain Driven Design
https://www.lulu.com/content/325231?page=1&pageSize=4



THE BUILDING BLOCKS OF A MODEL-DRIVEN DESIGN

Avram A., Marinescu F., Domain-Driven Design Quickly: A summary of Eric Evans' Domain Driven Design
https://www.lulu.com/content/325231?page=1&pageSize=4

DOMAIN EVENTS (INTRODUCED SINCE 2004)

- A domain event is representation of something that happened in the domain.
- Something happened that domain experts care about.
- Each event is represented as a domain object.

Preserving model integrity

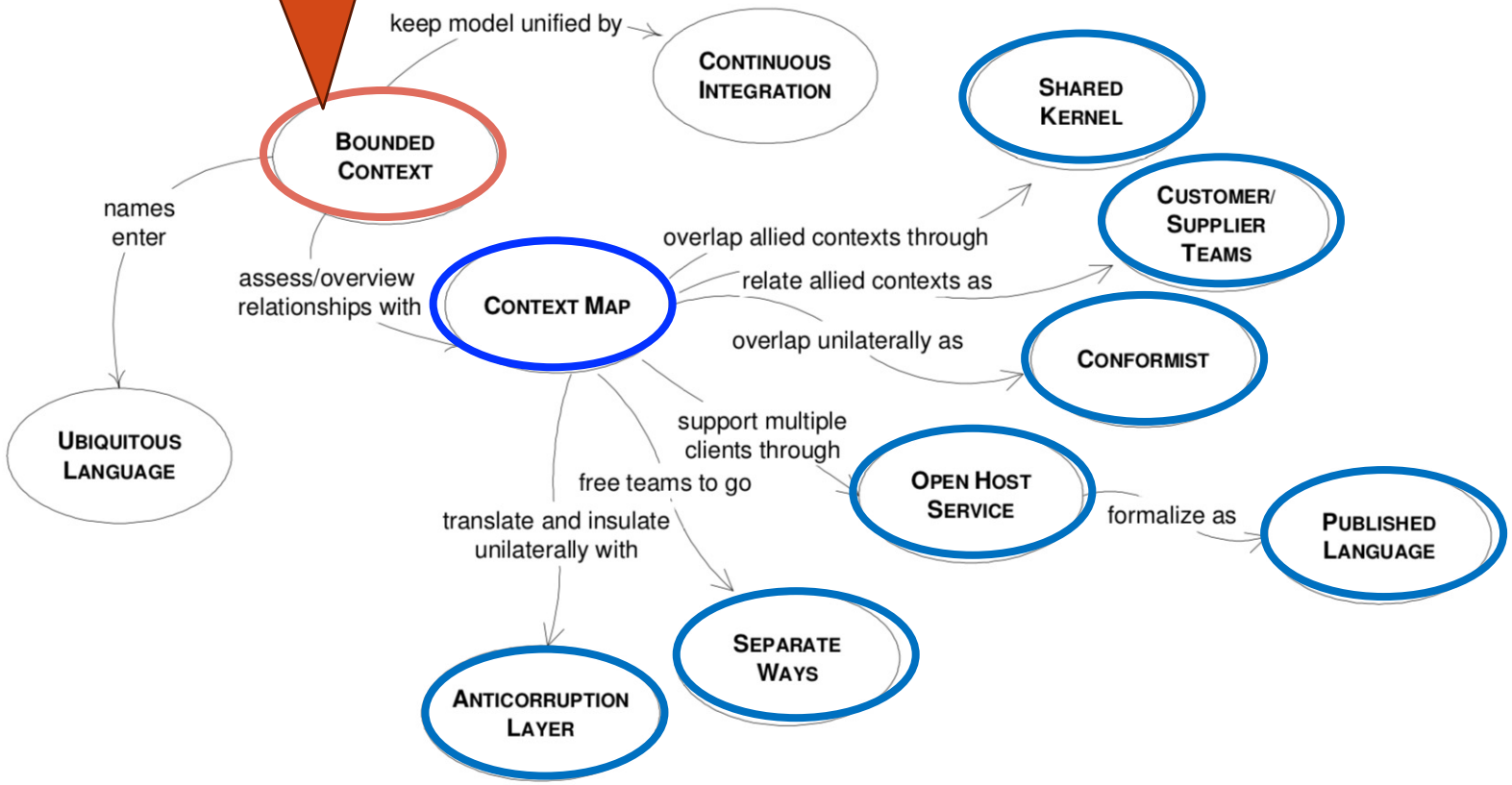
Chapter 14 in Evan's book

PRESERVING MODEL INTEGRITY

We need ways of keeping crucial parts of the model tightly unified.

Total unification of the domain model for a large system will not be feasible or cost-effective.

Defines the range of applicability of each model



A SET OF TECHNIQUES USED TO MAINTAIN MODEL INTEGRITY

Avram A., Marinescu F., Domain-Driven Design Quickly: A summary of Eric Evans' Domain Driven Design
https://www.lulu.com/content/325231?page=1&pageSize=4

Bounded Context

PROBLEM IN A LARGE PROJECT

- Multiple models are in play on any large project.
- When code based on distinct models is combined, software becomes buggy, unreliable, and difficult to understand.
- Communication among team members becomes confused.
- It is often unclear in what context a model should not be applied.

TO MANAGE A LARGE PROJECT

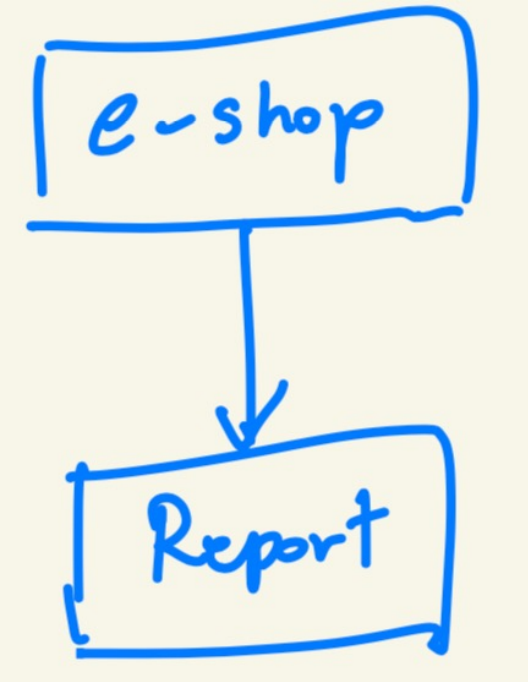
- Explicitly define the context within which a model applies.
- Explicitly set boundaries in terms of team organization, usage within specific parts of the application, and physical manifestations such as code bases and database schemas.
- Keep the model strictly consistent within these bounds, but don't be distracted or confused by issues outside.

BOUNDED CONTEXT

- A Bounded Context defines the range of applicability of each model.
- A Bounded Context provides the logical frame inside of which the model evolves.

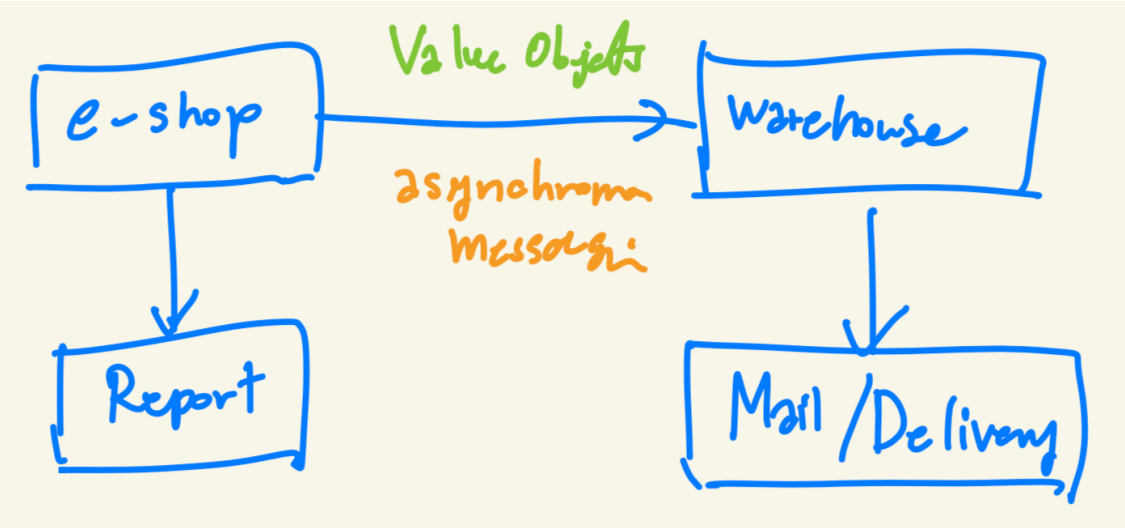
HOW ABOUT AN E-COMMERCE APPLICATION

- Sell stuff on the Internet
- Customers can register
- We collect their personal data
- Customers need to browse the site looking for merchandise, and place orders
- The application needs to send notify to the merchandise when an order has been placed.
- The application needs to generate reports, so we can monitor the status of goods, what the customers like, don't like.



THE RECOMMENDED APPROACH

- To create a separate model for each of the domains; one for the e-commerce, and one for the reporting.
- So, they can evolve freely without much concern about each other and even become separate applications.



THE RECOMMENDED APPROACH

- A messaging system is needed to inform the warehouse personnel about the order placed, so they can mail the purchased merchandise.
- The mail personnel will use an application which give them detailed information about the item purchased, the quantity, the customer address, and delivery requirements.
- The e-shop should just send Value Objects containing purchase information to the warehouse using asynchronous messaging.

Continuous Integration

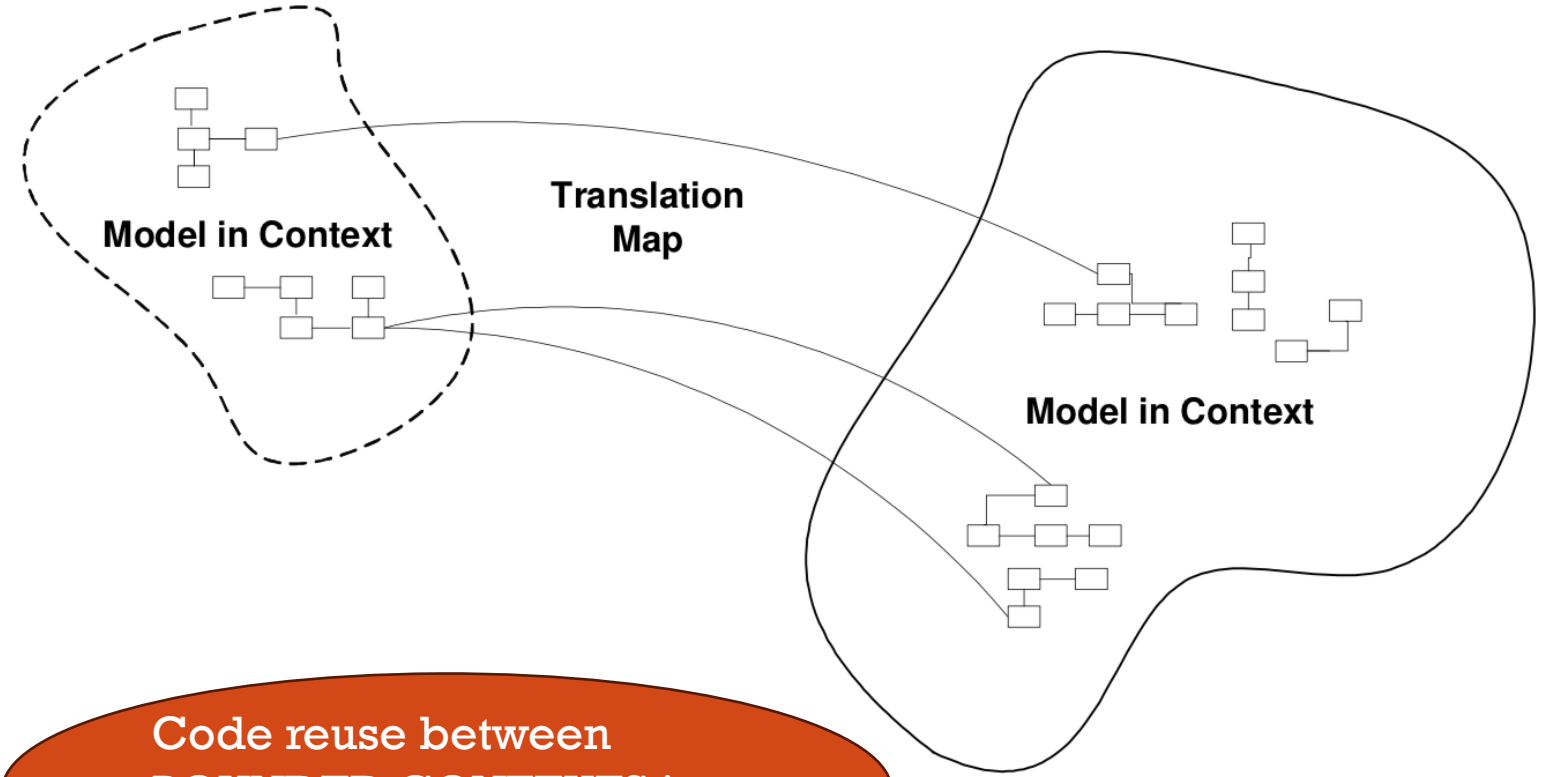
CONTINUOS INTEGRATION

- All work within the context is being merged and made consistent frequently enough that t when splinters happen they are caught and corrected quickly.
- Continuous Integration separates at two levels:
  - the integration of model concepts
  - the integration of the implementation.

Context Map

A CONTEXT MAP is in the overlap between project management & software design.

Context Map



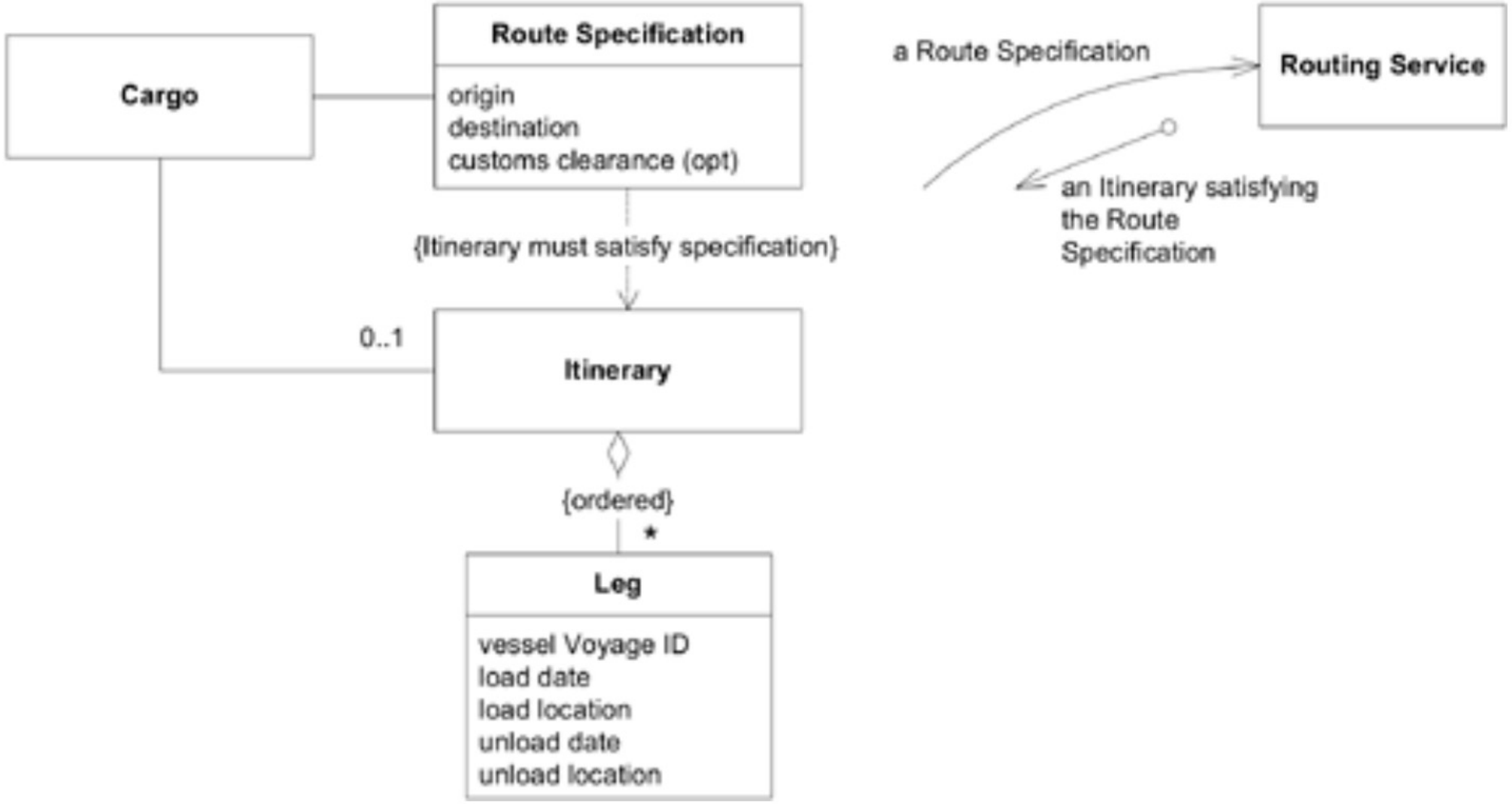
Code reuse between BOUNDED CONTEXTS is a hazard to be avoided.

WHY?

- A Context Map is a document which outlines the different Bounded Contexts and relationships between them.

Eric Evans. Domain Driven Design. 2003

TWO CONTEXTS IN A SHIPPING APPLICATION



TWO CONTEXTS IN A SHIPPING APPLICATION (CONT.)

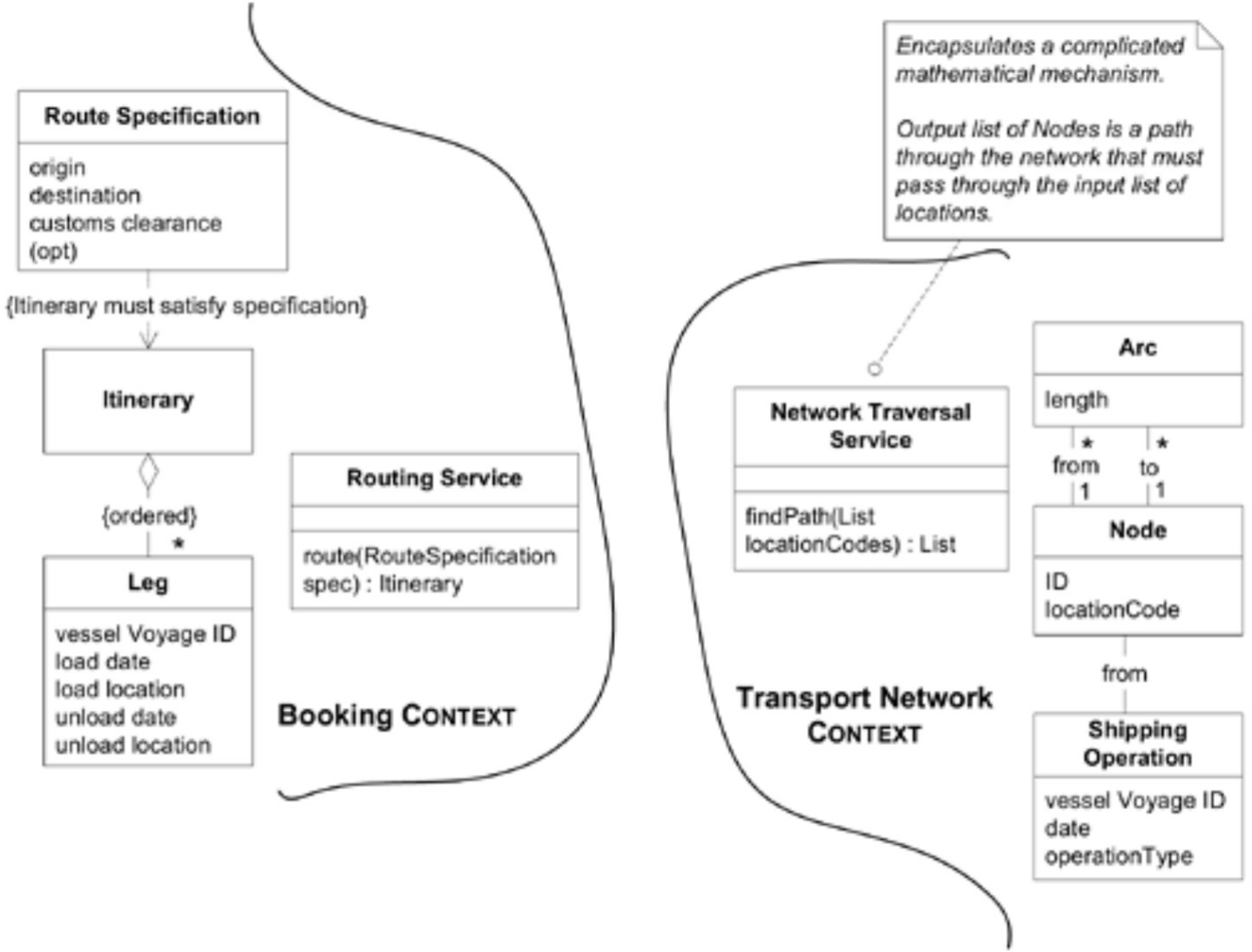
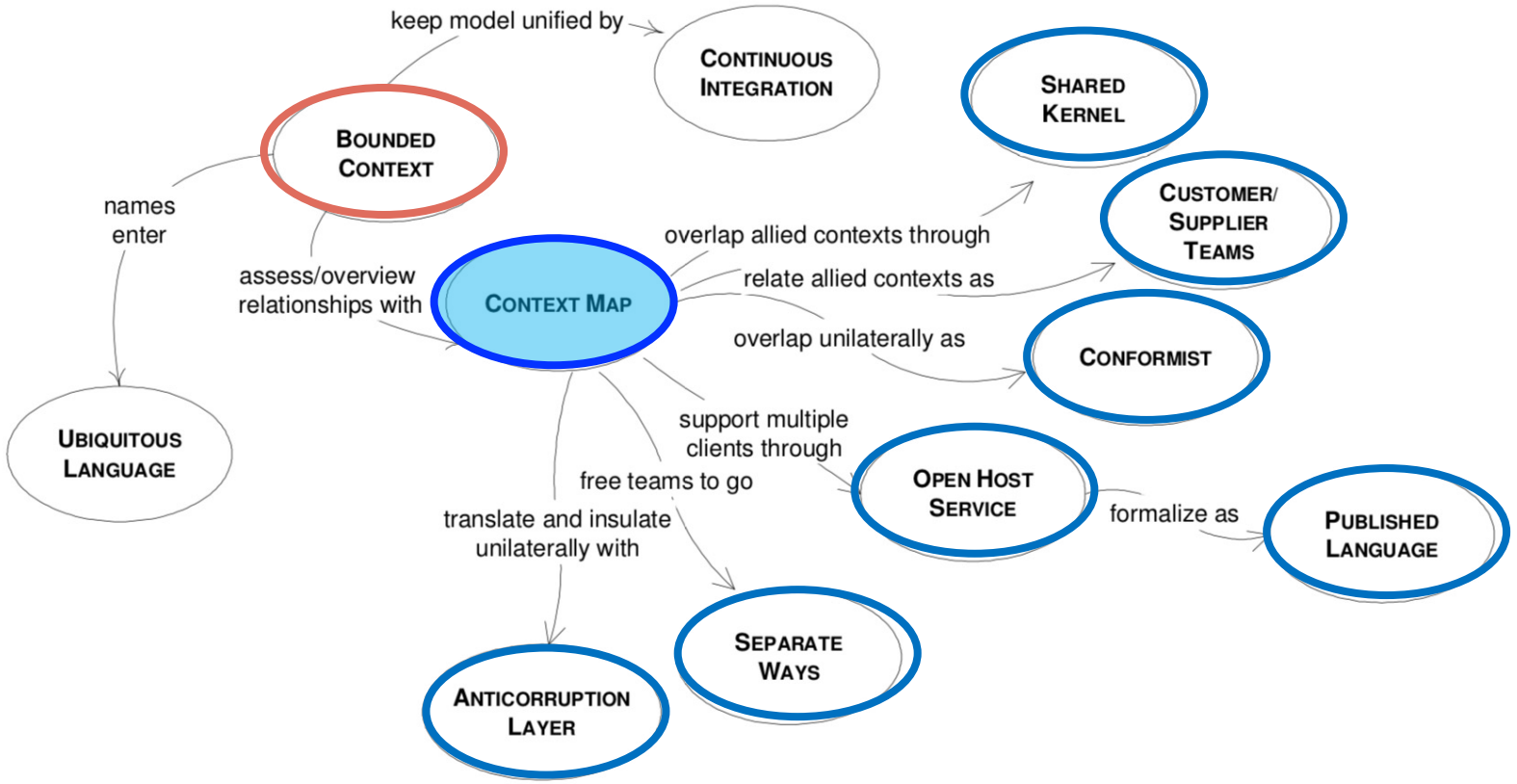


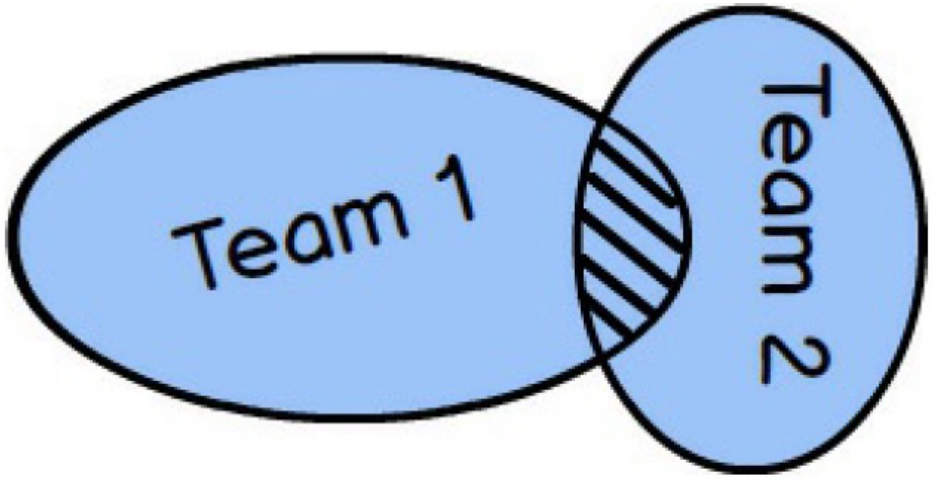
Figure 14.3. Two BOUNDED CONTEXTS formed to allow efficient routing algorithms to be applied



A SET OF TECHNIQUES USED TO MAINTAIN MODEL INTEGRITY

Avram A., Marinescu F., Domain-Driven Design Quickly: A summary of Eric Evans' Domain Driven Design
https://www.lulu.com/content/325231?page=1&pageSize=4

SHARED KERNEL

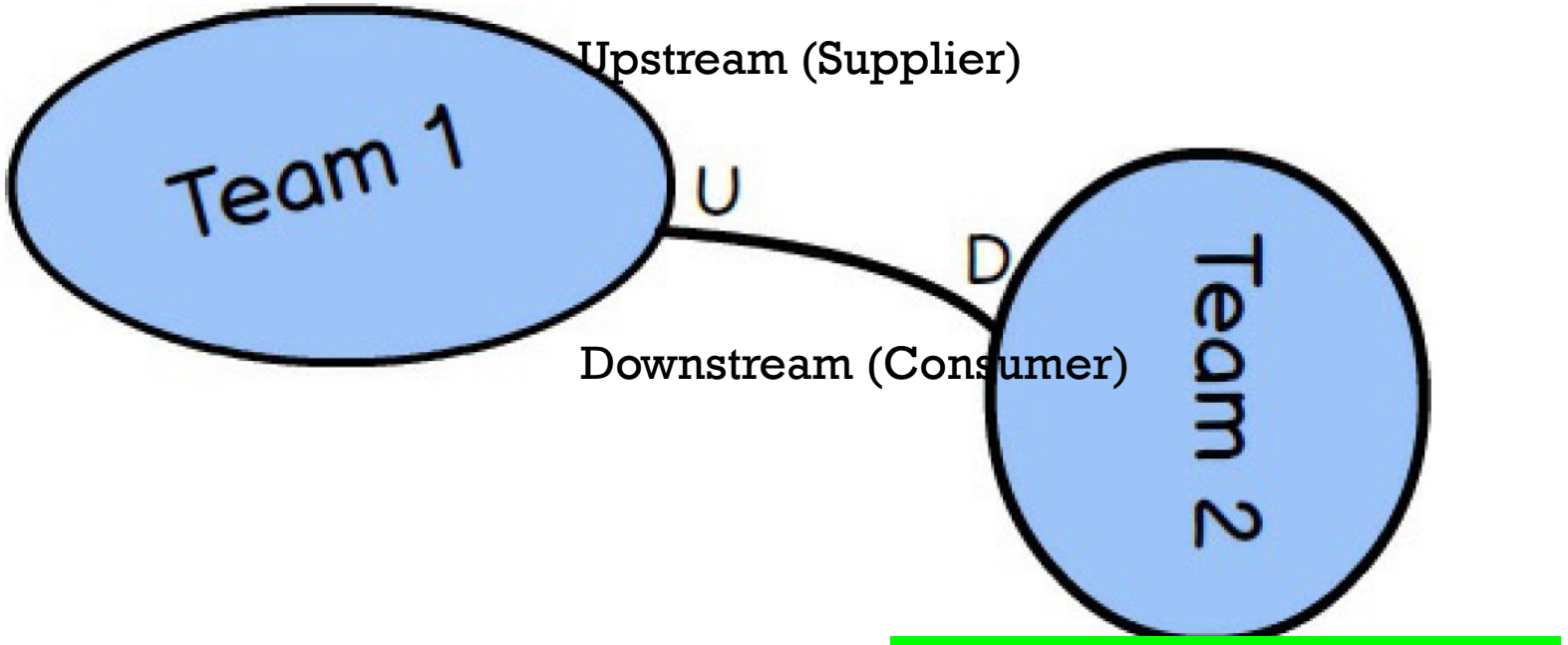


The teams must agree on what model elements they are to share.

Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc

CUSTOMER-SUPPLIER

How could we handle the team management?

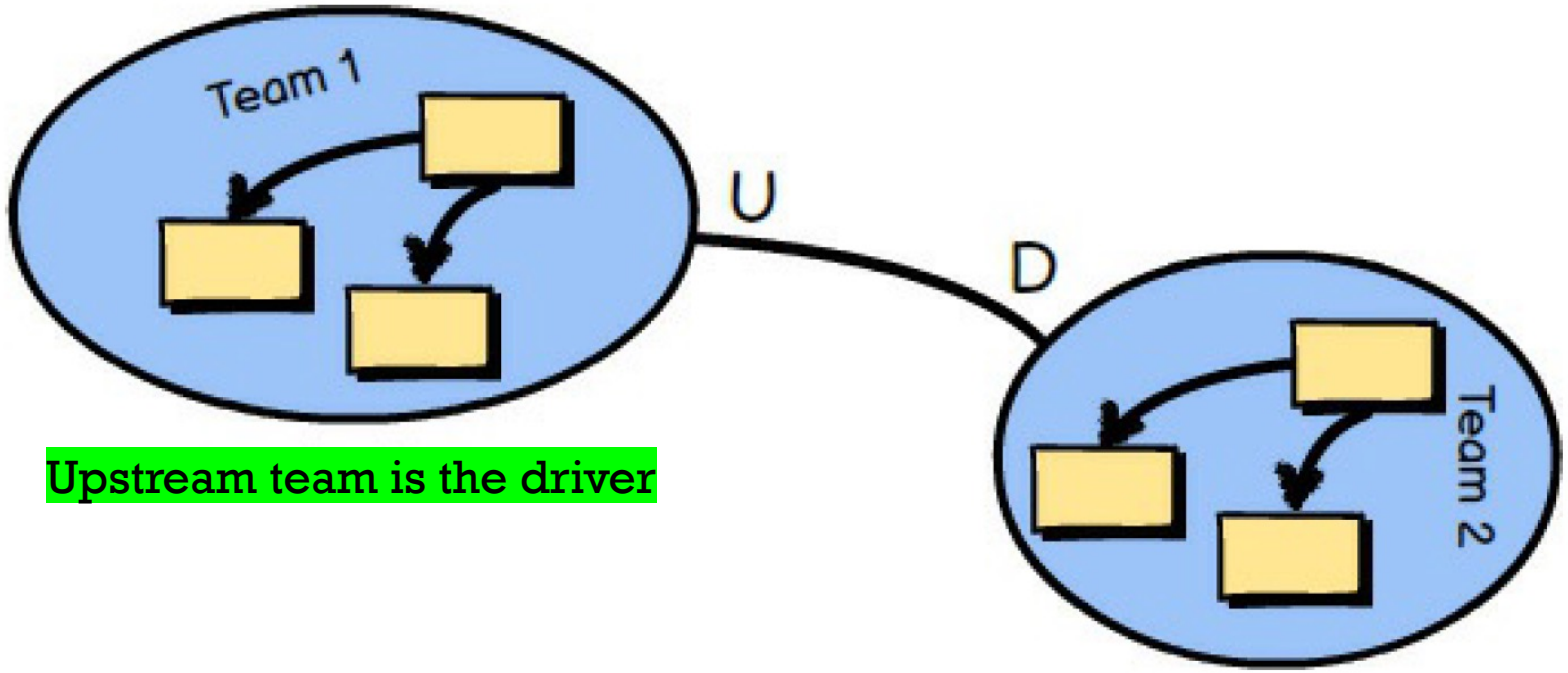


Downstream team is the driver

The *Supplier* must provide what the *Customer* needs.

Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc

CONFORMIST

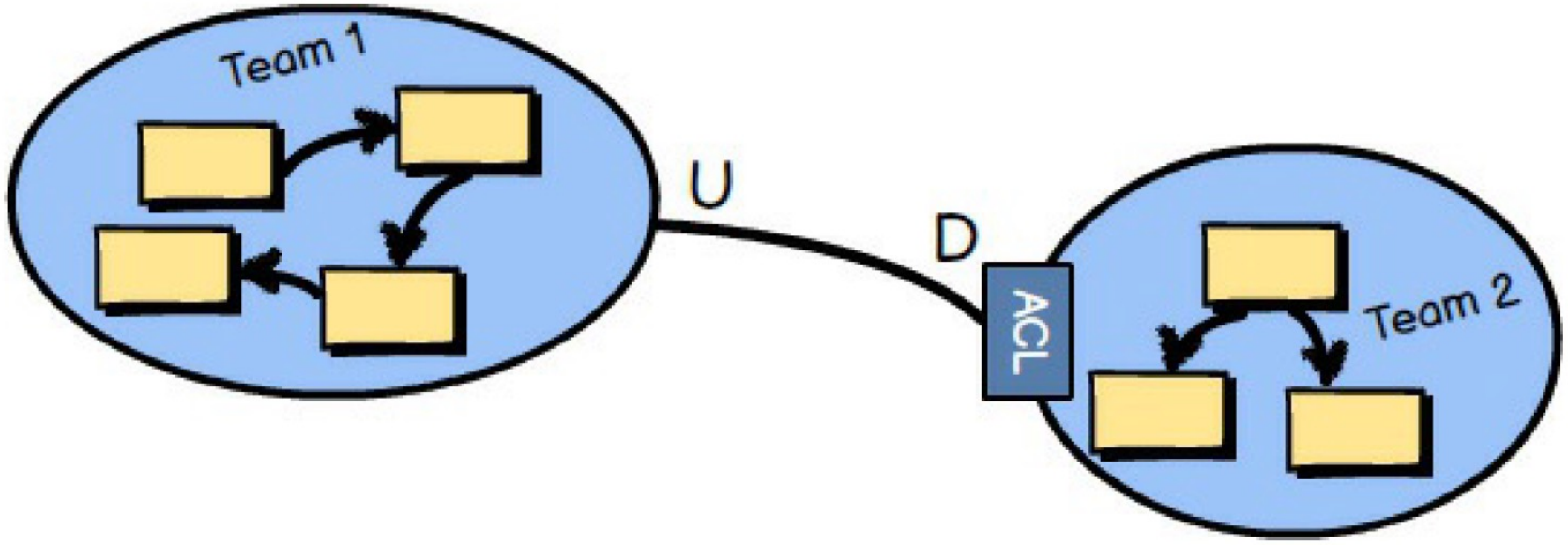


Upstream team is the driver

For various reasons the downstream team cannot sustain an effort to translate the Ubiquitous Language of the upstream model to fit its specific needs, so the team conforms to the upstream model as is.

Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc

ANTICORRUPTION LAYER (ACL)



An Anticorruption Layer is the most defensive Context Mapping relationship. The layer isolates the downstream model from the upstream model and translates between the two.

Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc

ANTICORRUPTION LAYER (ACL) (CONT.)

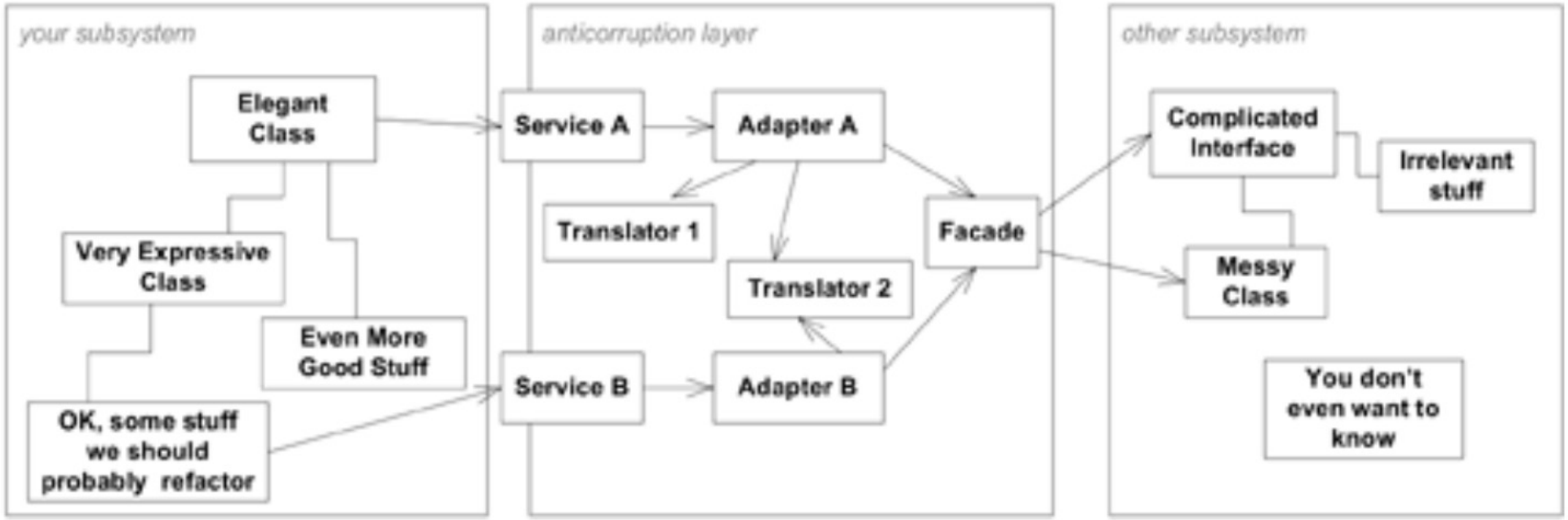
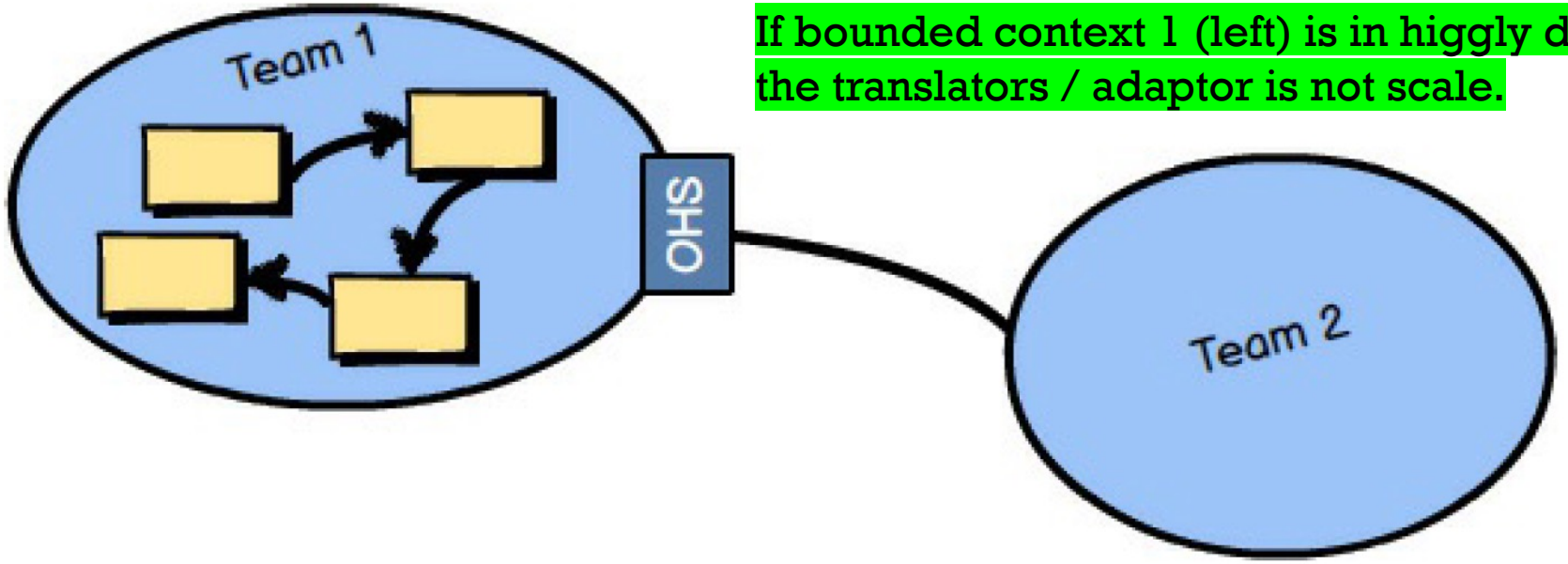


Figure 14.8. The structure of an ANTICORRUPTION LAYER

Bounded context 2 (right) needs to understand the domain language used by bounded context 1

OPEN HOST SERVICE

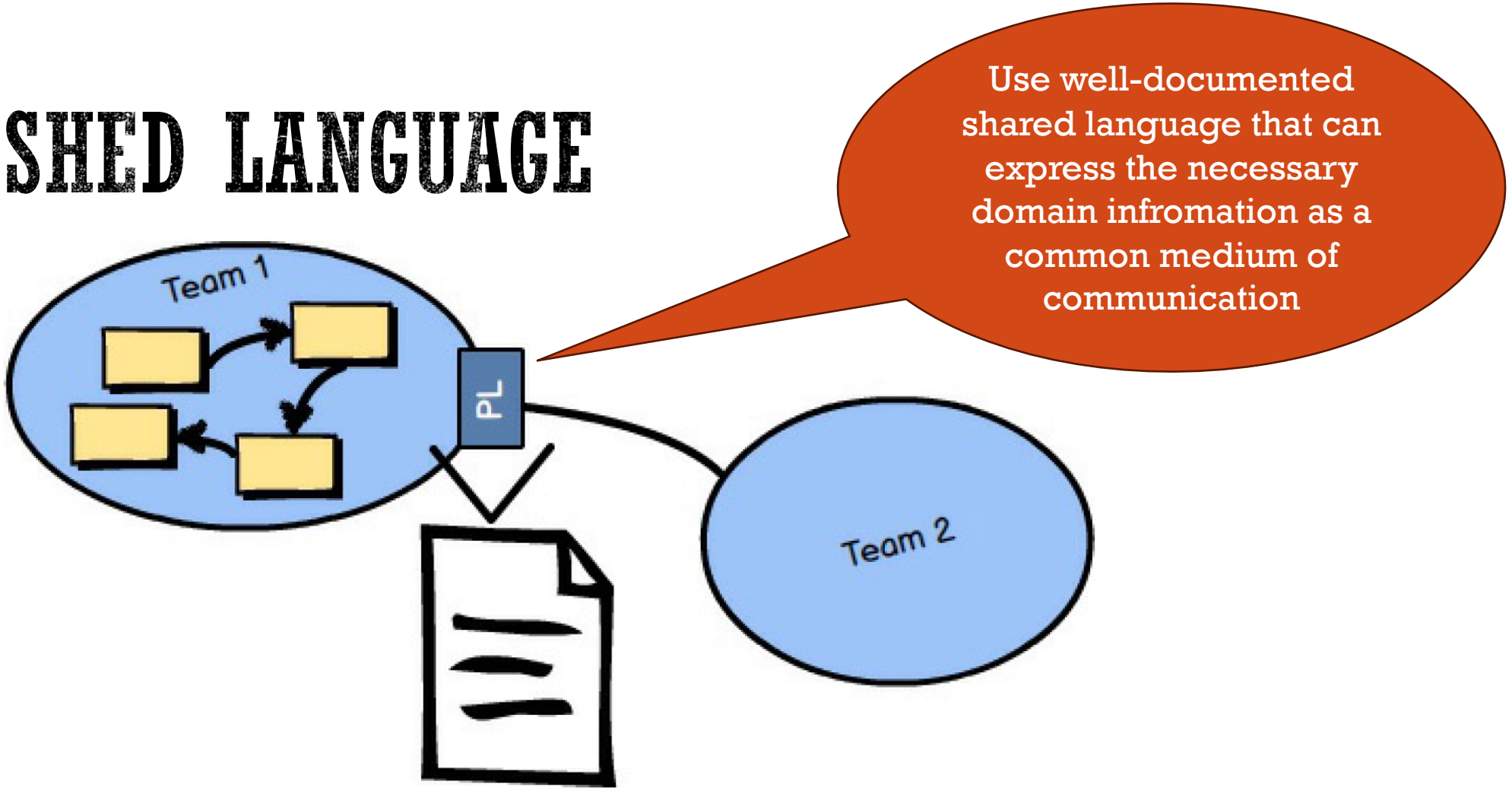


If bounded context 1 (left) is in higgly demanded, the translators / adaptor is not scale.

An Open Host Service defines a standardized protocol or interface that gives access to your Bounded Context as a set of services.

Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc

PUBLISHED LANGUAGE



Such a Published Language can be defined with XML Schema, JSON Schema, or a more optimal wire format, such as Protobuf or Avro.

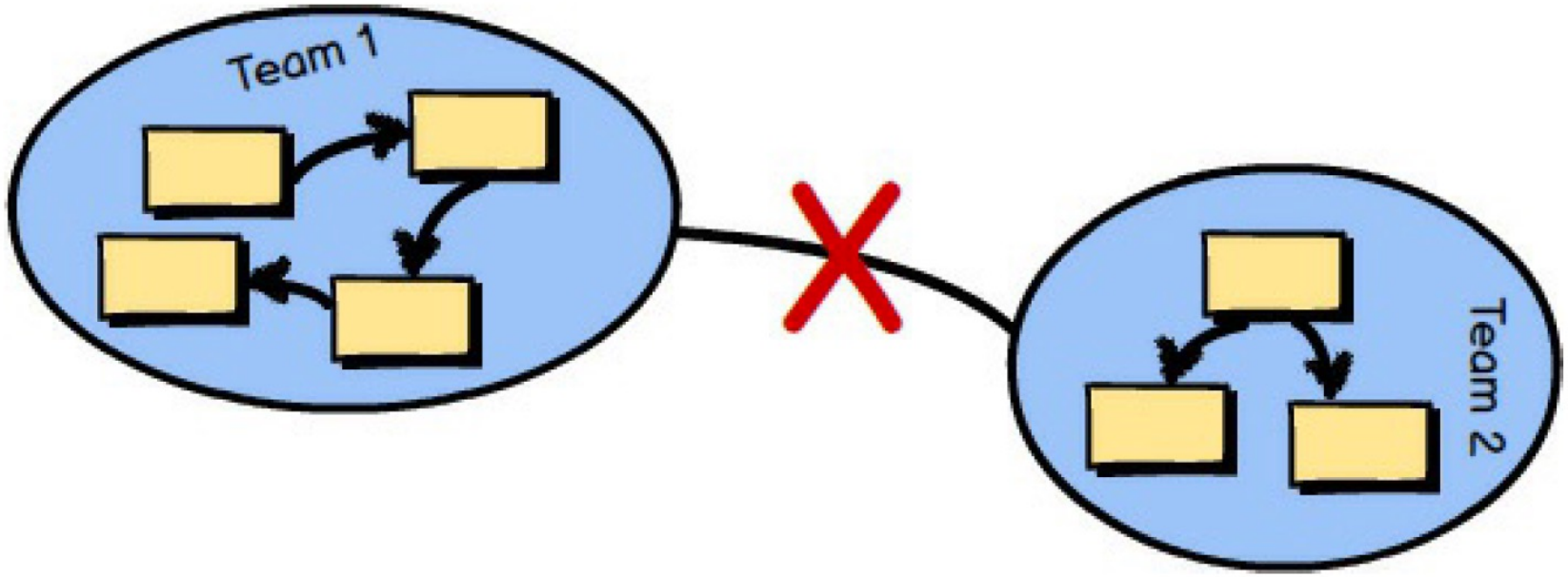
Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc

Example: Chemical Markup Language (CML)

```
<CML.ARR ID="array3" EL.TYPE=FLOAT NAME="ATOMIC ORBITAL ELECTRON POPULATIONS" SIZE=30 ↩
GLO.ENT=CML.THE.AOEPOPS>
   1.17947   0.95091   0.97175   1.00000   1.17947   0.95090   0.97174   1.00000
   1.17946   0.98215   0.94049   1.00000   1.17946   0.95091   0.97174   1.00000
   1.17946   0.95091   0.97174   1.00000   1.17946   0.98215   0.94049   1.00000
   0.89789   0.89790   0.89789   0.89789   0.89790   0.89788
</CML.ARR>
```

SEPARATE WAYS

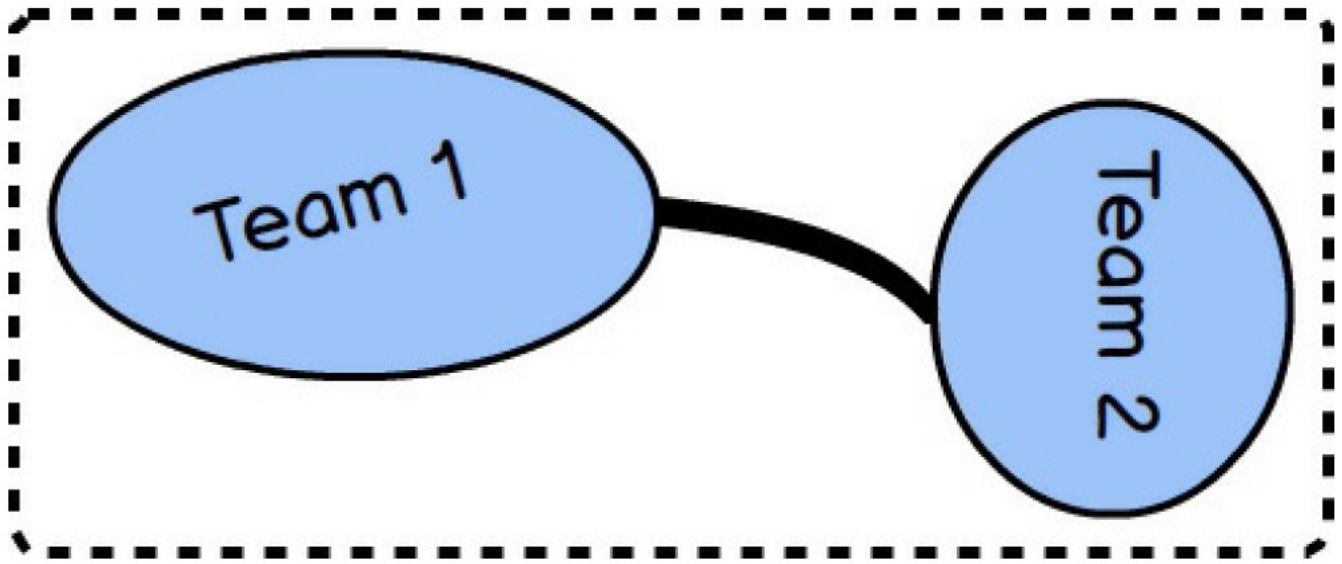
Allowing developers to find simple, specialized solutions within this small scope.



In this case produce your own specialized solution in your Bounded Context and forget integrating for this special case.

Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc

PARTNERSHIP (INTRODUCED SINCE 2004)

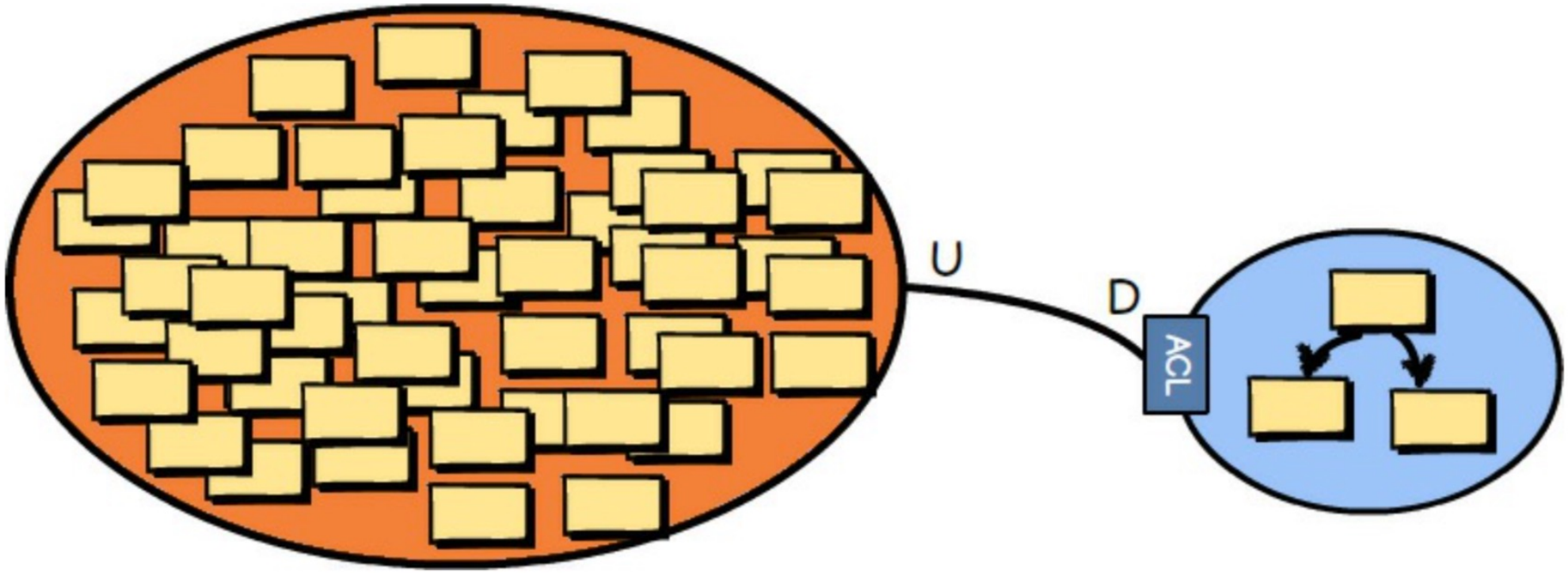


Each team is responsible for one Bounded Context.

When teams in two contexts will succeed or fail together, a cooperative relationship often emerges.

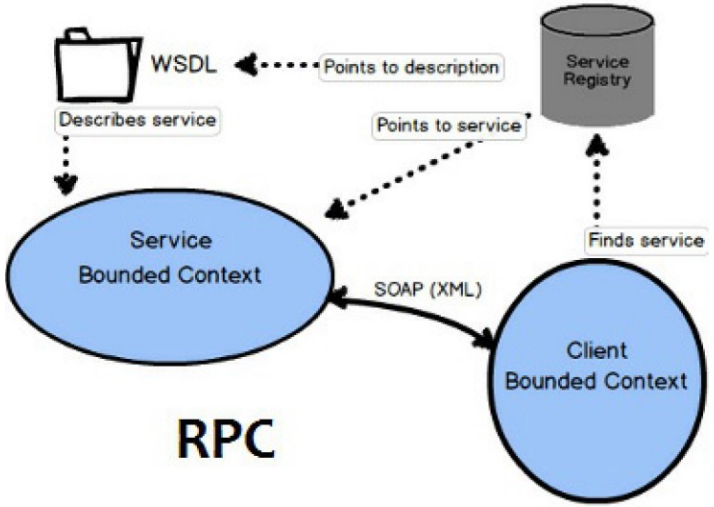
Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc

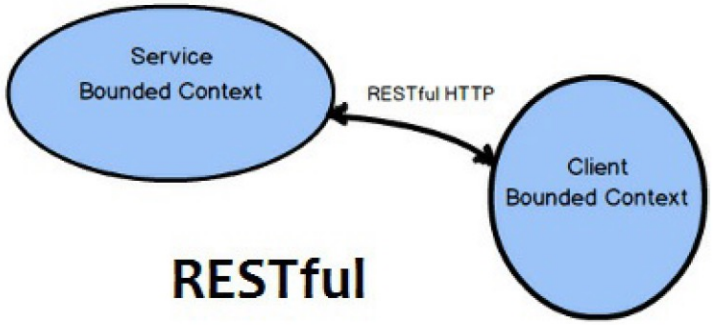
BIG BALL OF MUD (INTRODUCED SINCE 2004)



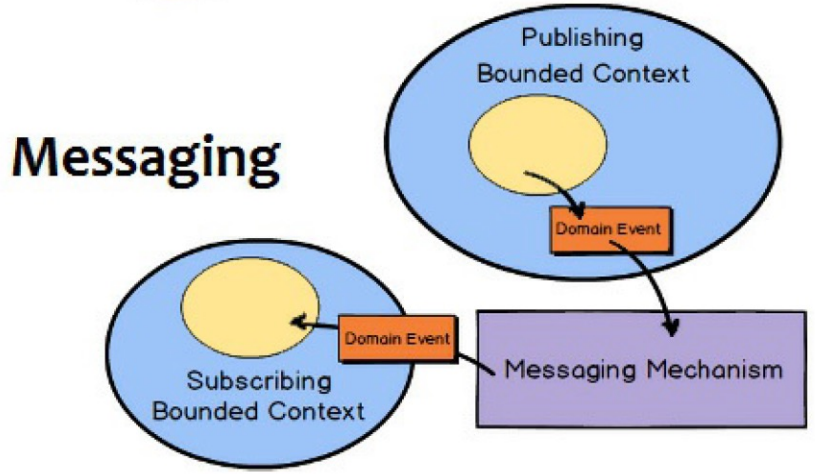
Avoid to create big ball of mud, and if need to use it, try ACL.

MAKING GOOD USE OF CONTEXT MAPPING



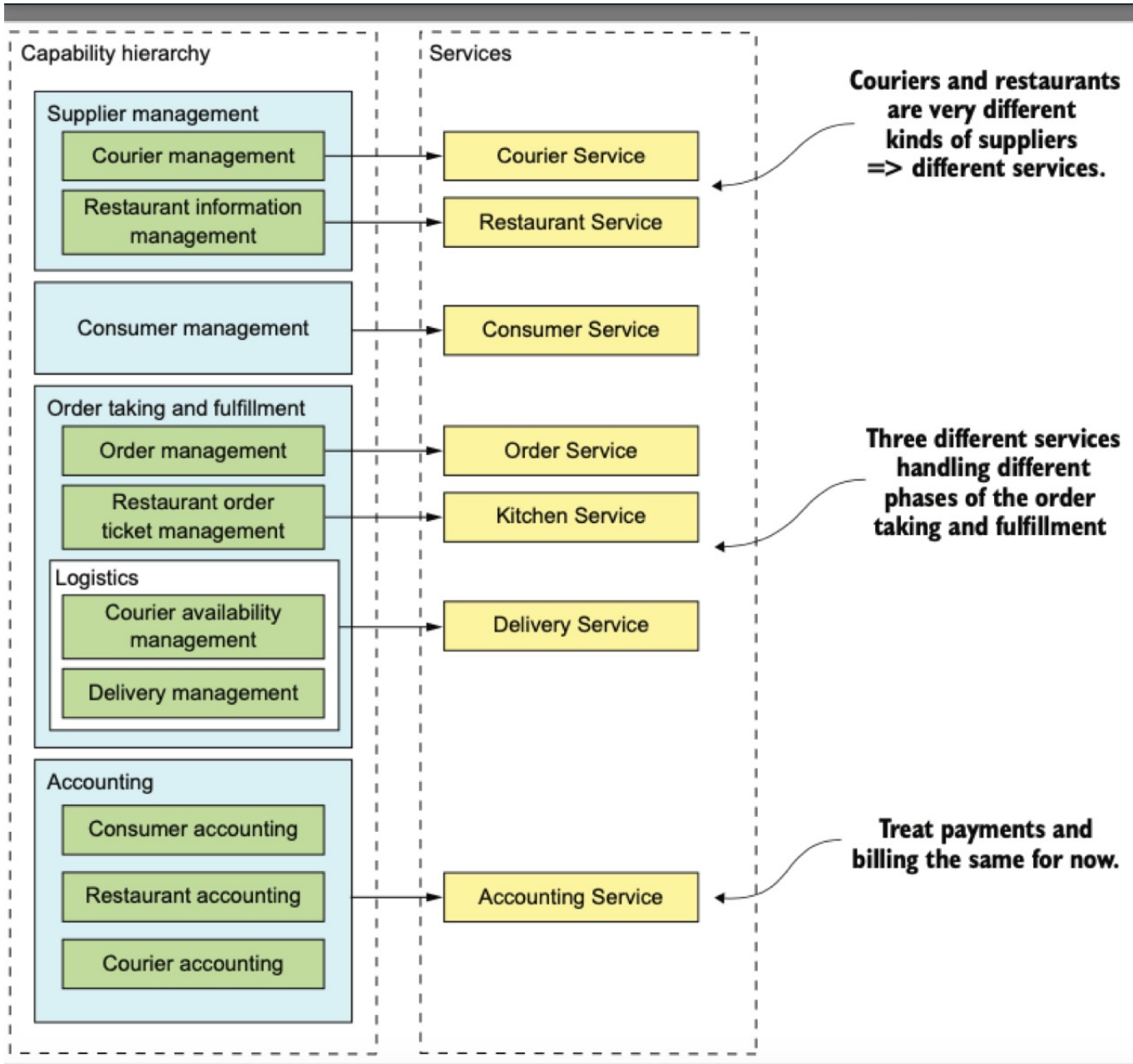


What a specific kind of interface would be supplied to allow you to integrate with a given *Bounded Context?*



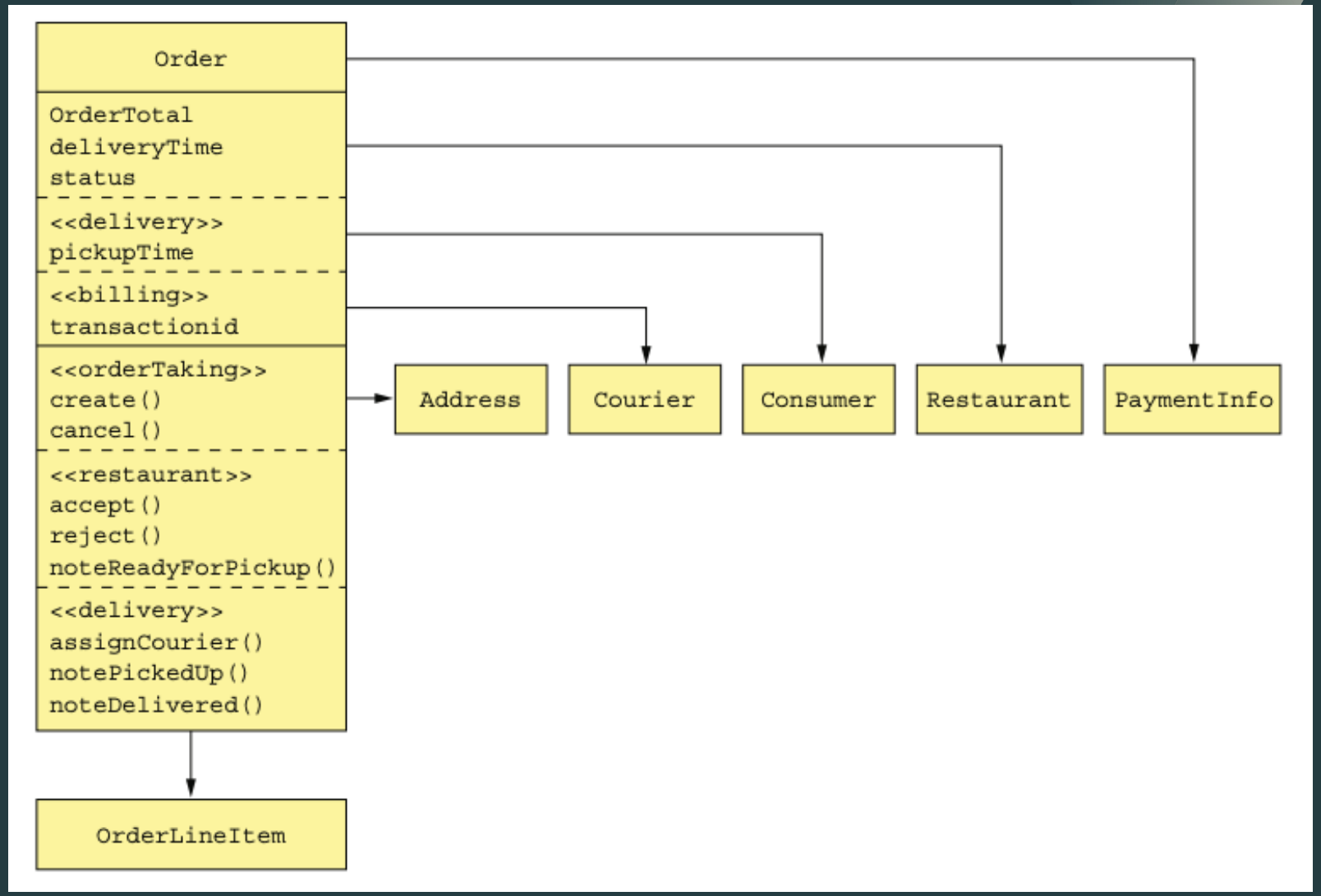
Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc

Mapping FTGO business capabilities to services



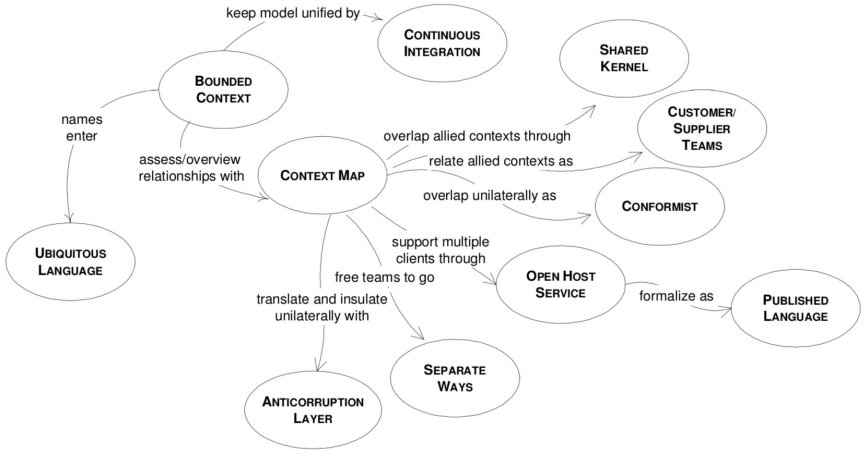
The Order, a god class in FTGO

- The purpose of FTGO is to deliver food orders to customers.
- If the FTGO application had a single domain model, the Order class would be a very large class

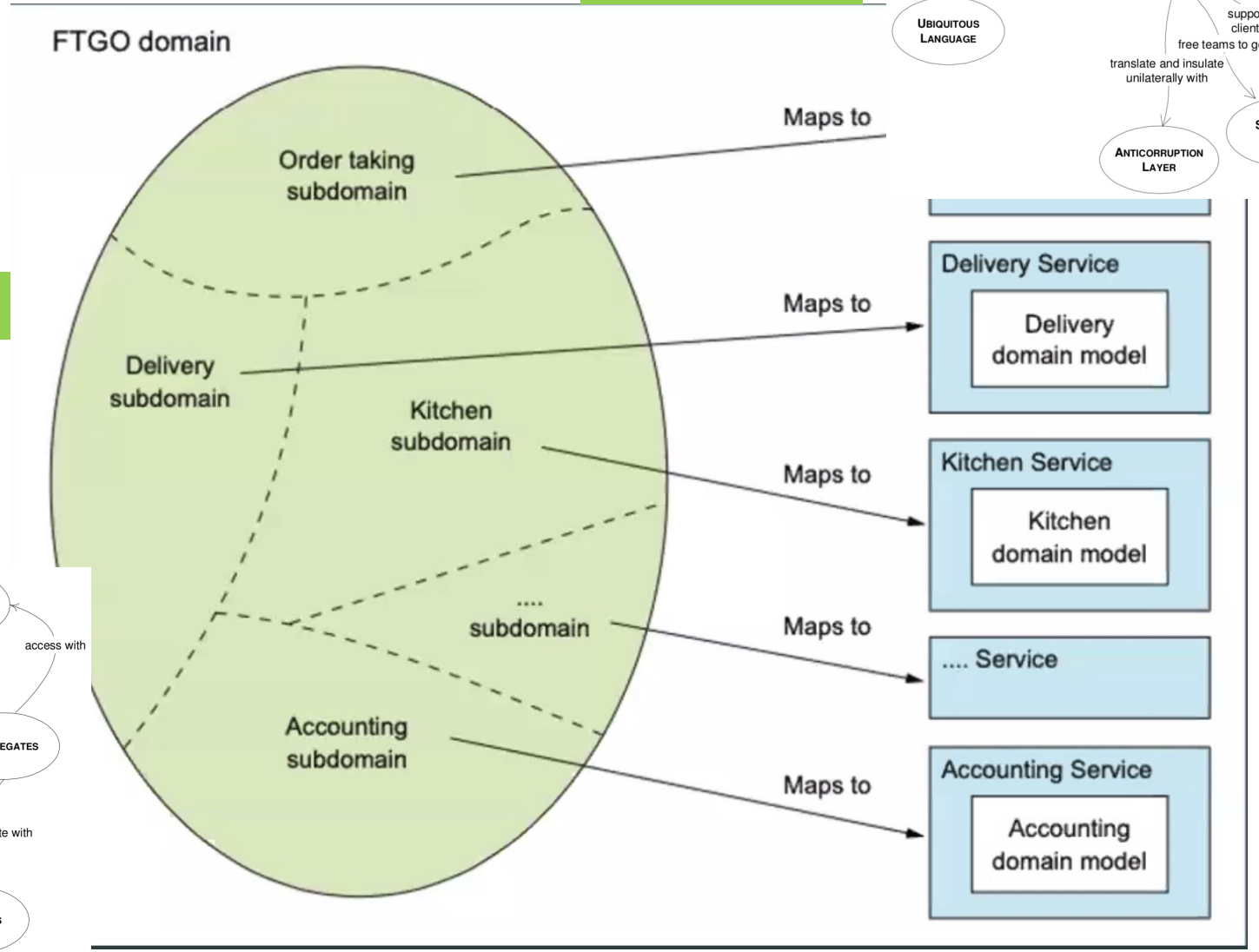


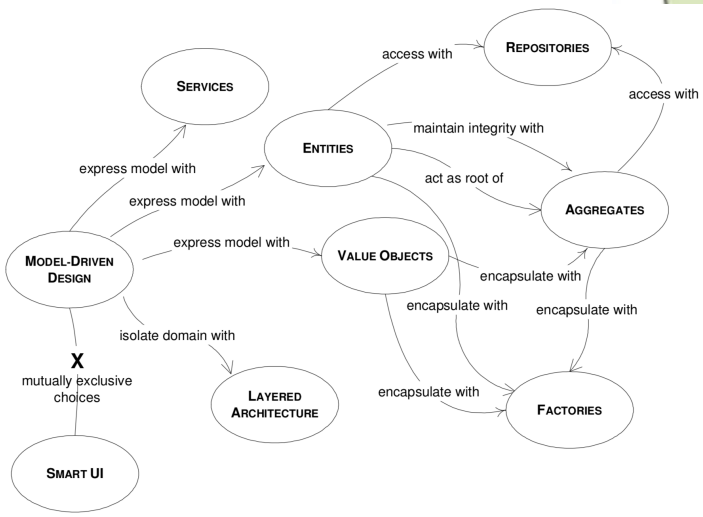
The Order god class is bloated with numerous responsibilities

Context map



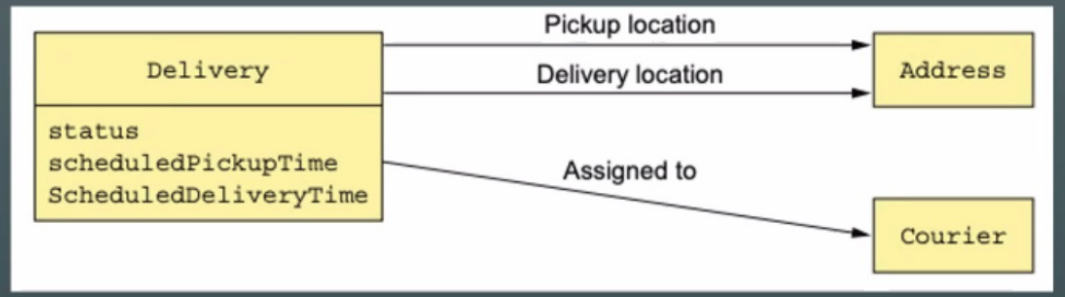
Bounded context



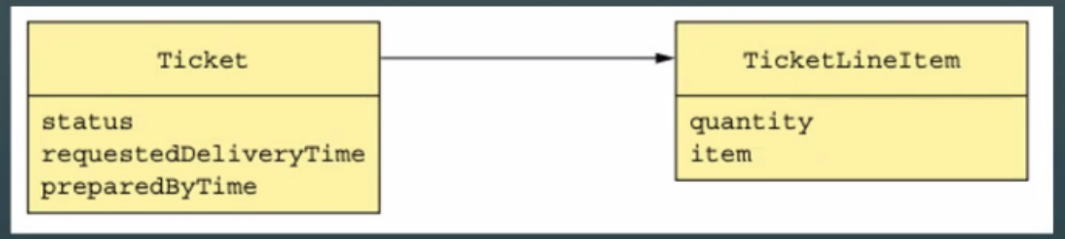


Applying DDD is a solution.

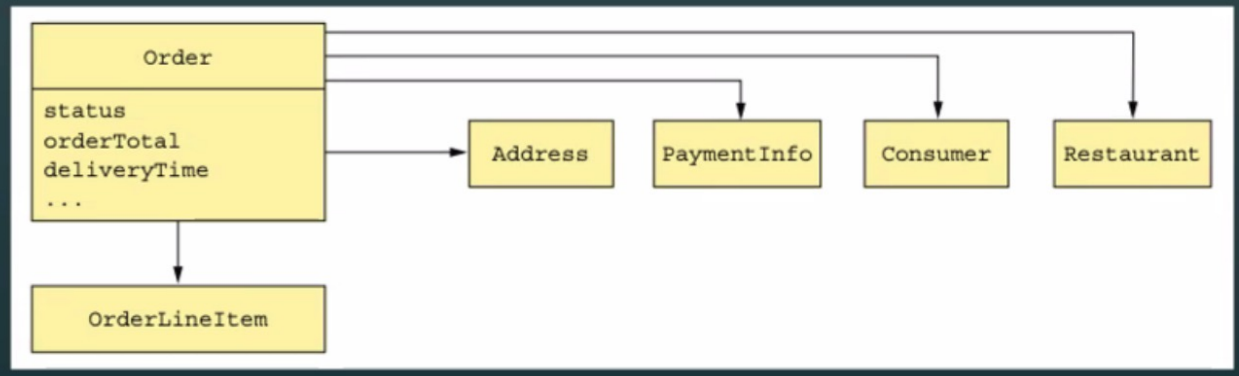
- A much better approach is to apply DDD.
- Treat each service as a separate sub- domain with its own domain model.
- The Order class in each domain model represents different aspects of the same Order business entity



The *Delivery Service* domain model



The *Kitchen Service* domain model



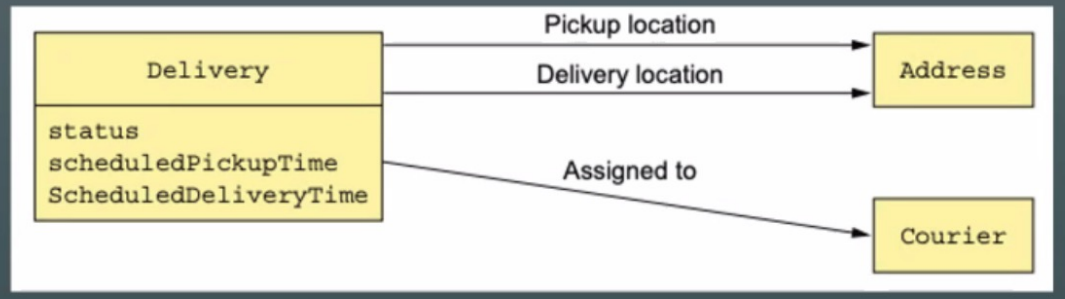
The *Order Service* domain model

1) What is bounded context?
2) Within a bounded context, what is Entities/ Value Object?
3) What is Aggregates?
4) What is repositories?
5) What is Services?
6) What is Factories?

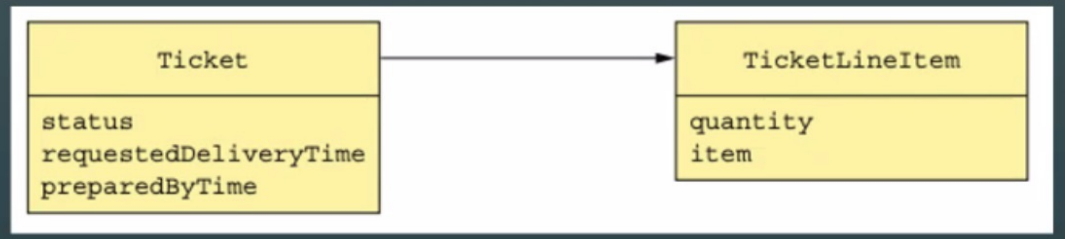
- A much better approach is to apply DDD.

1) What is context map? What types of context map will be?

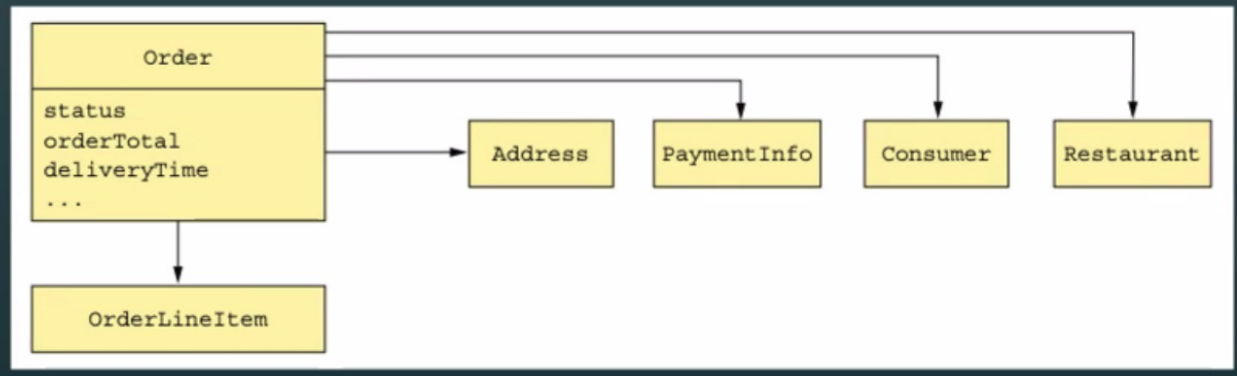
Order business entity



The *Delivery Service* domain model



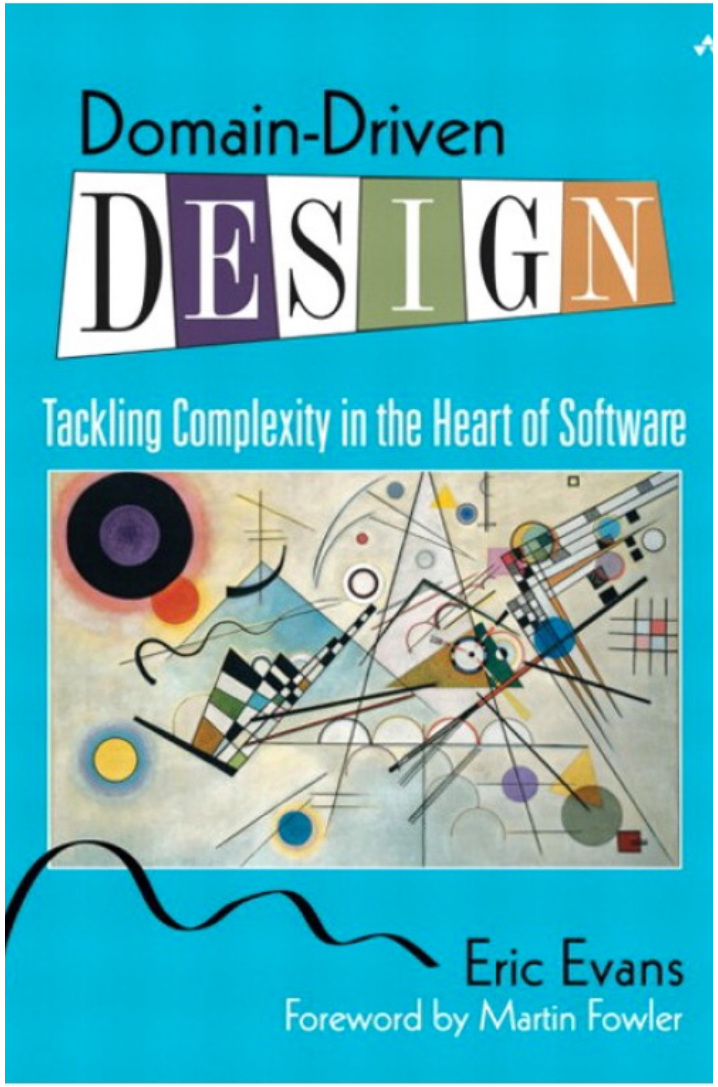
The *Kitchen Service* domain model



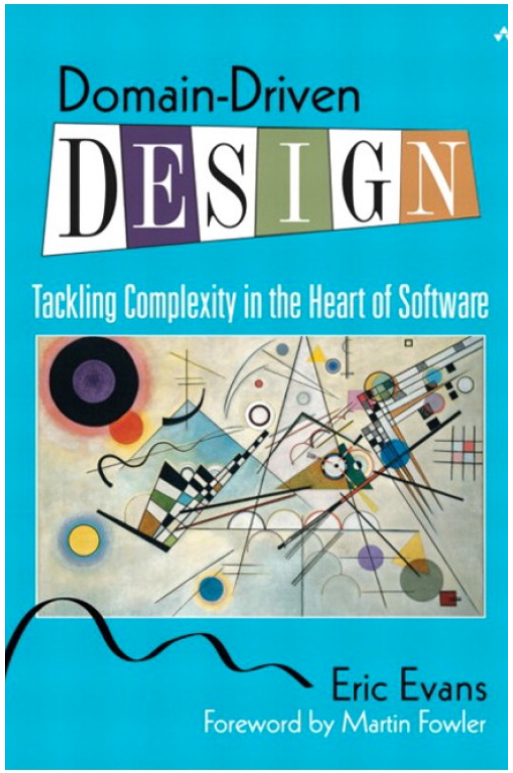
The *Order Service* domain model

REFERENCES

- Avram A., Marinescu F., Domain-Driven Design Quickly: A summary of Eric Evans' Domain Driven Design https://www.lulu.com/content/325231?page=1&pageSize=4
- Vernon, V. Domain-Driven Design Distilled (2016). Chapter 4: Strategic Design with Context Mapping. Pearson Education, Inc
- https://github.com/ernesen/DDD
- Evans, E. Domain-Driven Design: Tacking Complexity in the Heart of Software, 2003.



TAKE HOME YOUTUBE VIDEO



Bounded Contexts - Eric Evans - DDD Europe 2020

20,878 views • Oct 2, 2020

LIKE DISLIKE SHARE SAVE ...

https://www.youtube.com/watch?v=am-HXycfalo

TAKE HOME YOUTUBE VIDEO

#EventStorming
Event Storming - Alberto Brandolini - DDD Europe 2019

https://www.youtube.com/watch?v=mLXQIYEwK24

Introducing EventStorming

An act of deliberate collective learning

The book is available on Leanpub (PDF, EPub and Mobi).
It's not finished, but there's already enough content to help explorers and practitioners.

Read the book ›

Paper version will follow

https://www.eventstorming.com/book/

IN CLASS GROUP ASSIGNMENT

ทำโปรเจคอะไร

- มี domain อะไรบ้าง / มี Event อะไรบ้าง (1-2 events)
- Ubiquitous language
- ยกตัวอย่าง Entity / value objects / service ใน 1 domain
- ยกตัวอย่าง context map มา 1 context map พร้อมเหตุผล

ส่ง 11:59am

TAKE HOME MESSAGE

- “DDD is less about software design patterns and more about problem solving through collaboration.
- Evans presents techniques to use software design patterns to enable models created by the development team and business experts to be implemented using the UL.
- However, without the practices of analysis, and collaboration, the coding implementation really means very little on its own
- DDD is not code centric; its purpose is not to make elegant code. Software is merely an artifact of DDD.

“

Scott Millett with Nick Tune. Patterns, Princioles and Practices of Domain-Driven Design. 2015

TAKE HOME MESSAGE

- Strategic design focuses on defining the bounded contexts, ubuquitous languages, and context maps
- Tactical design is a set of technical resources used in constructing domain model in a bounded context.