# Domain Events and Anti-Corruption Layers

## Decoupling the Domain Model's Communications

Julie Lerman
TheDataFarm.com
@julielerman

Steve Smith
Ardalis.com
@ardalis

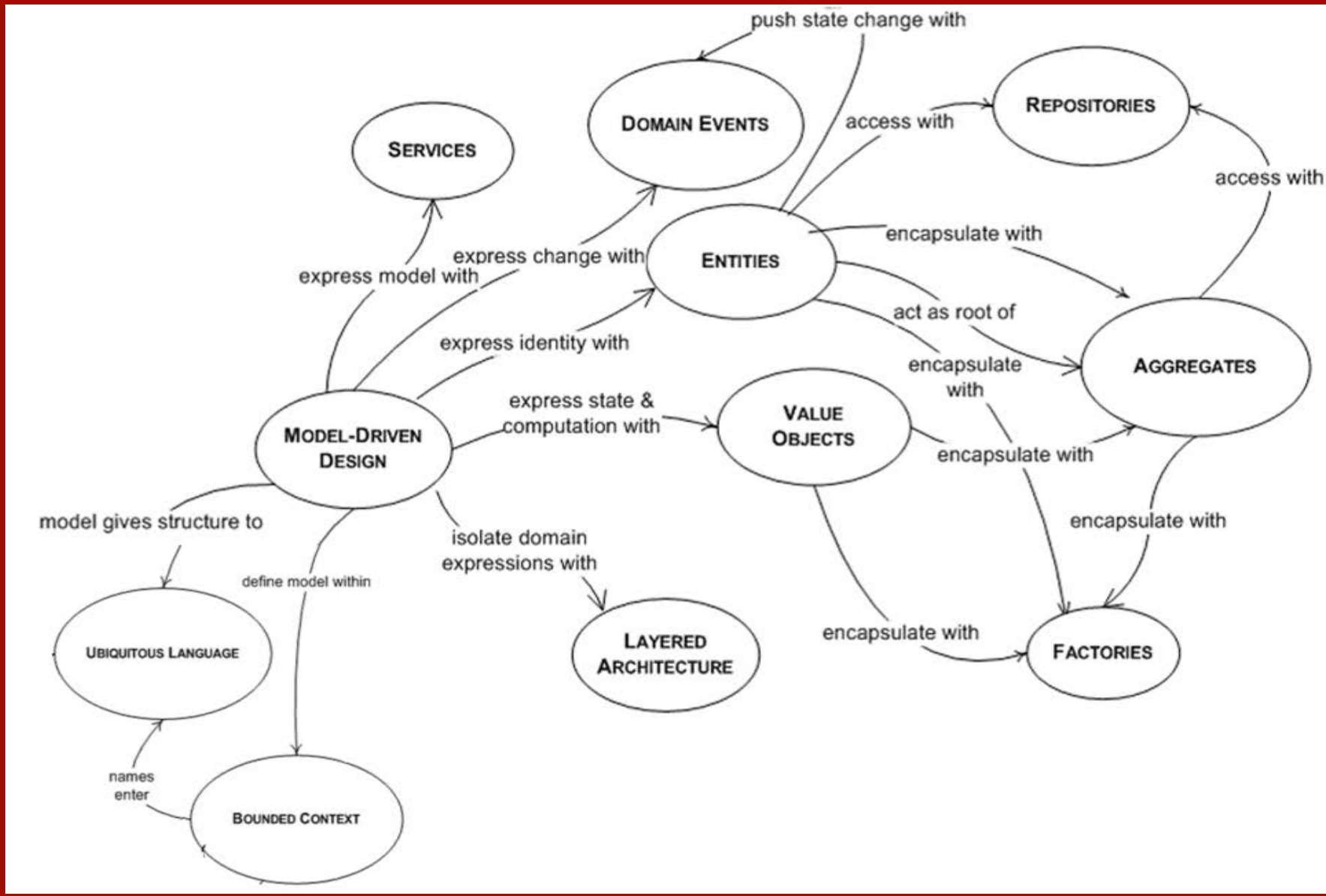
pluralsight
hardcore dev and IT training

# In This Module

- Domain Events
- Anti-Corruption Layers

# Domain Events

# Domain Events



“

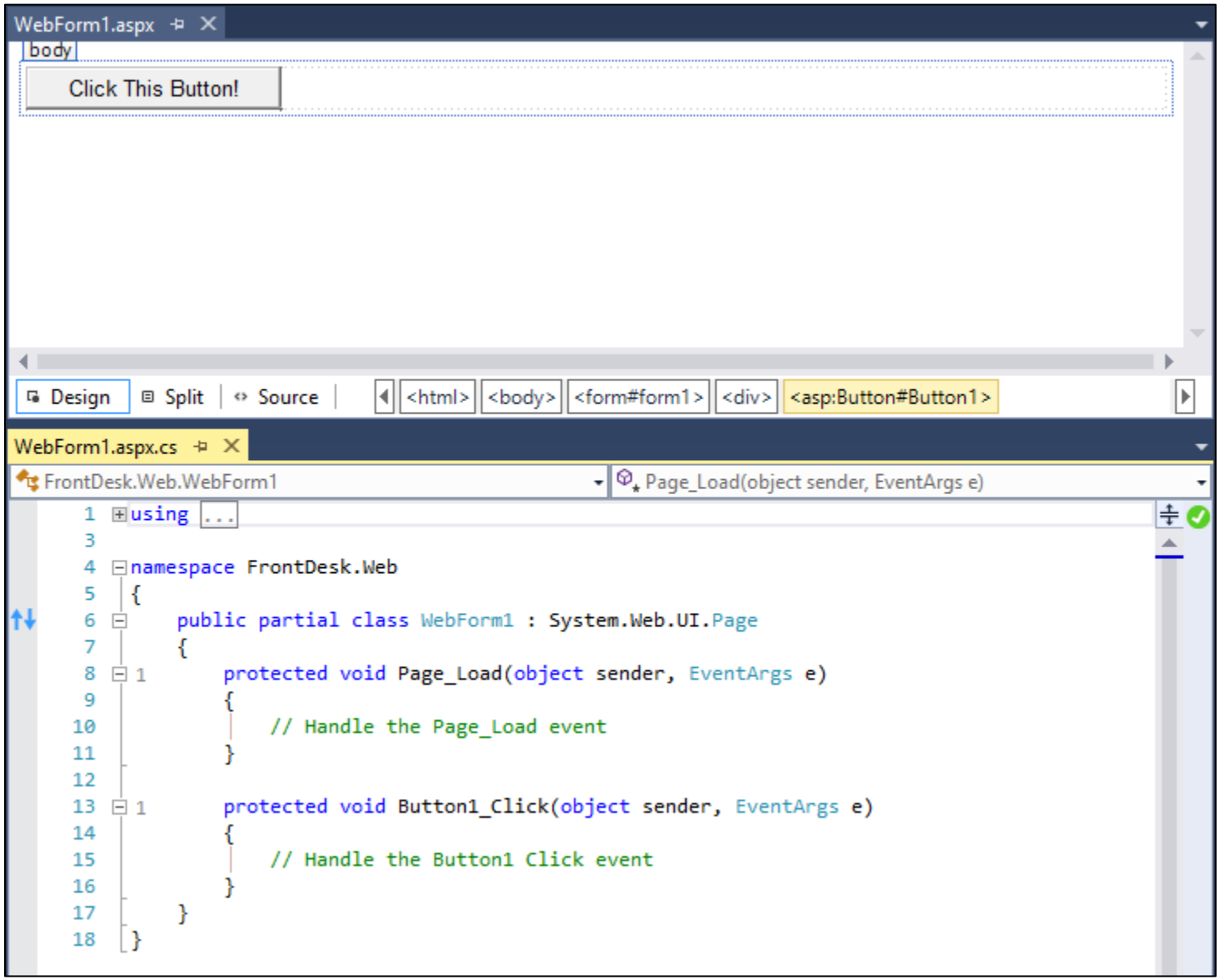
Use a **Domain Event** to capture an occurrence of something that happened in the domain.

”

— Vaughn Vernon

Implementing Domain-Driven Design

# Familiar Events



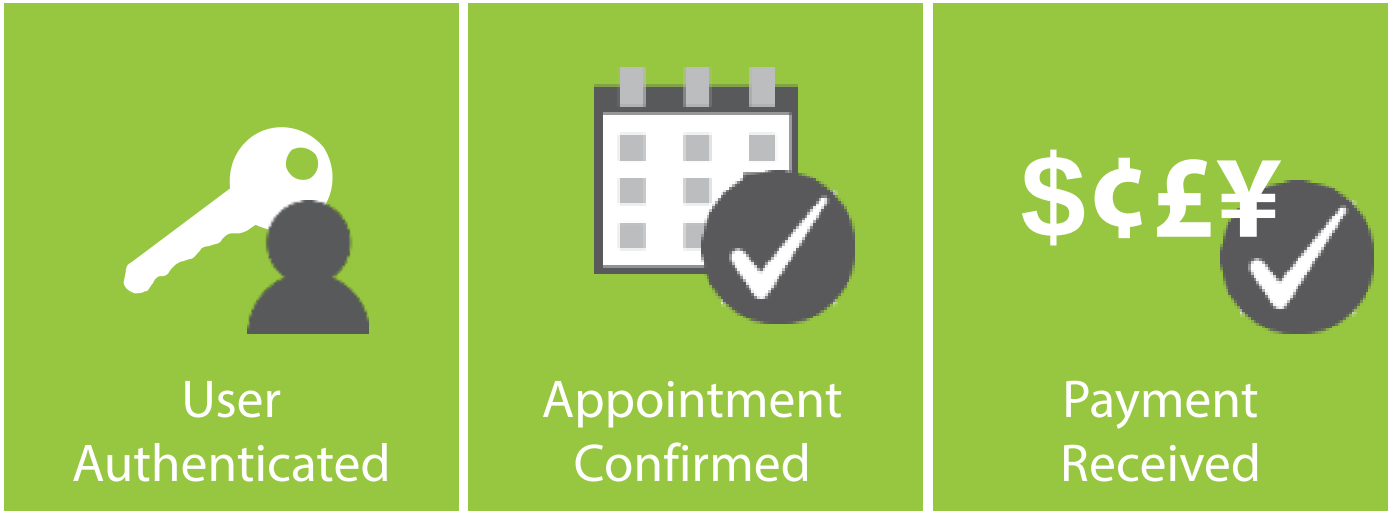
# Domain Event Pointers

- **"When this happens, then something else should happen."**
  - Also listen for:
    - "If that happens…"
    - "Notify the user when…" or "Inform the user if…"

- **Domain events represent the past**

- **Typically they are immutable**

# Domain Event Examples



# Designing Domain Events

- Each event is its own class
- Include when the event took place

```
public interface IDomainEvent
{
  DateTime DateTimeOccurred {get;}
}
```

- Capture event-specific details
  - What would you need to know to trigger this event again?
  - Consider including the identities of any aggregates involved in the event
- Event fields are initialized in constructor
- No behavior or side effects

## Domain Events: First Look

# Demo

# Domain Events in our Application

# Domain Event Boundaries

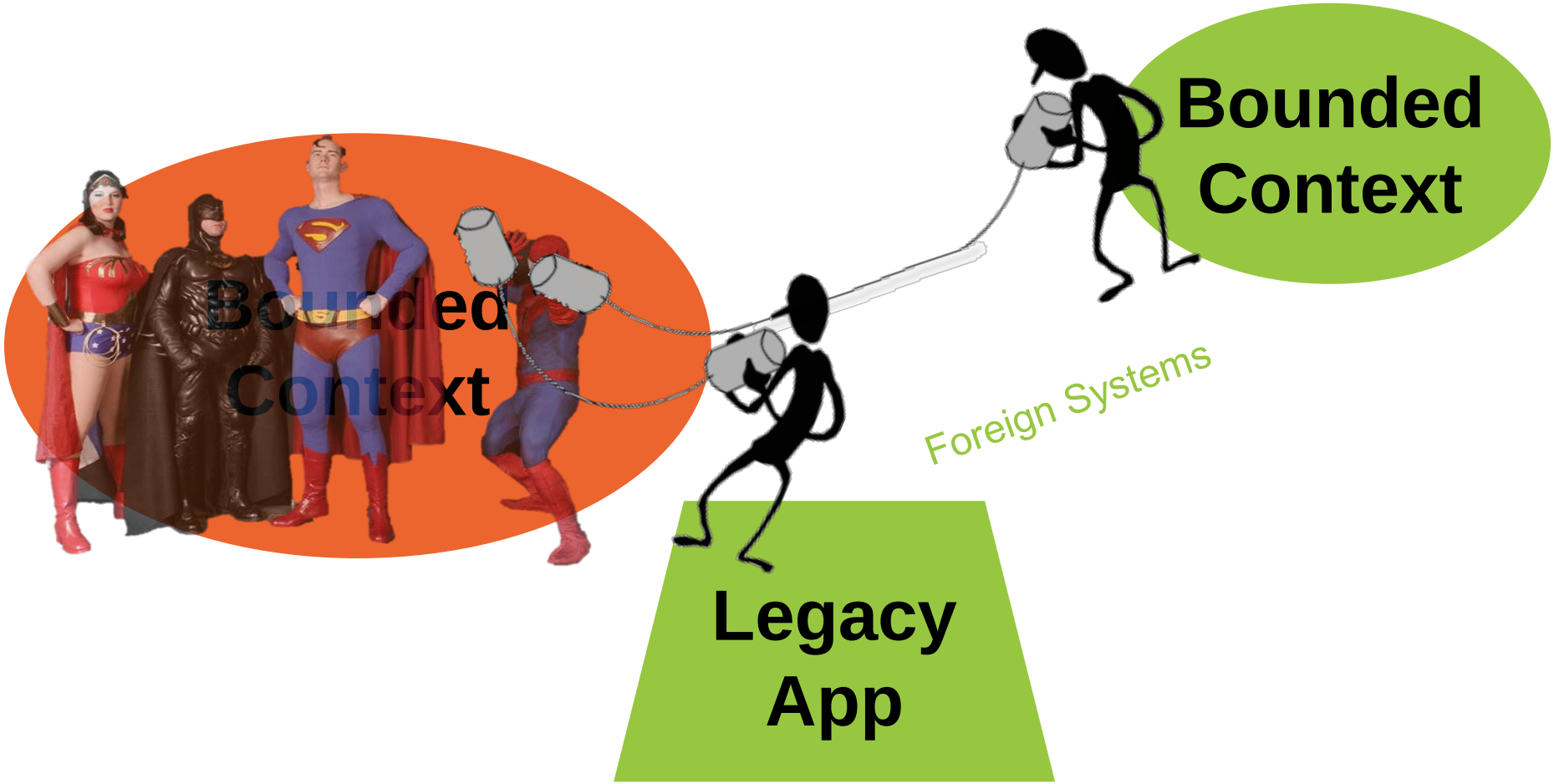
```
public class RelayAppointmentScheduledService : IHandle<AppointmentScheduledEvent>
{
    public void Handle(AppointmentScheduledEvent domainEvent)
    {
        dynamic newEvent = new
        {
            Start = domainEvent.AppointmentScheduled.TimeRange.Start,
            End = domainEvent.AppointmentScheduled.TimeRange.End,
            Client = "",
            Patient = "",
            AppointmentType = "",
            Doctor = ""
        };

        Fetch the Client name, Patient name, Appointment Type, and Doctor information

        Publish the newEvent to a message bus for other systems to consume
    }
}
```

# Anti-Corruption Layers


push state change with
DOMAIN EVENTS
access with
REPOSITORIES
access with
SERVICES
express model with
express change with
ENTITIES
encapsulate with
act as root of
express identity with
encapsulate with
AGGREGATES
MODEL-DRIVEN DESIGN
express state & computation with
VALUE OBJECTS
encapsulate with
model gives structure to
isolate domain expressions with
encapsulate with
LAYERED ARCHITECTURE
define model within
encapsulate with
FACTORIES
UBIQUITOUS LANGUAGE
cultivate rich model with
keep model unified by
CONTINUOUS INTEGRATION
names enter
BOUNDED CONTEXT
SHARED KERNEL
SHARED KERNEL
CORE DOMAIN
interdependent contexts form
overlap allied contexts through
CUSTOMER /SUPPLIER
relate allied contexts as
work in autonomous, clean
assess/overview relationships with
CONTEXT MAP
minimize translation
CONFORMIST
avoid overinvesting in
GENERIC SUBDOMAINS
segregate the conceptual messes
free teams to go
support multiple clients through
OPEN HOST SERVICE
translate and insulate unilaterally with
loosely couple contexts through
formalize
BIG BALL OF MUD
SEPARATE WAYS
PUBLISHED LANGUAGE
ANTI-CORRUPTION LAYER

# Anti-Corruption Layers

## Insulate Bounded Contexts



# Translate

between

# foreign systems' models & our own

using

## Design patterns, e.g. : Façade, Adapter

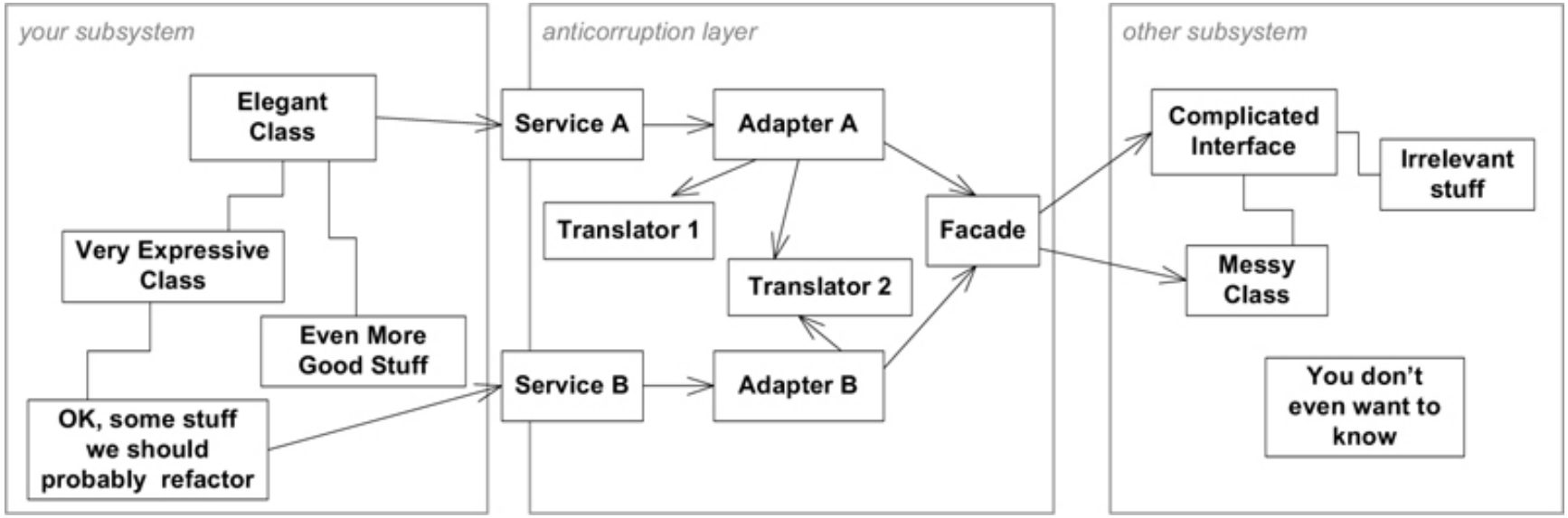
## or custom translation classes or services

> Even when the other system is well designed, it is not based on the *same* model as the client.
>
> And often the other system is not well designed.

— Eric Evans

Domain-Driven Design

# The structure of an Anti-Corruption Layer




**Source: Evans, Domain-Driven Design, p. 367**

# Glossary of Terms from this Module

## Domain Event

A class that captures the occurrence of an event in a domain object

## Hollywood Principle

"Don't call us, we'll call you"

## Inversion of Control (IOC)

A pattern for loosely coupling a dependent object with an object it will need at runtime

## Anti-Corruption Layer

Functionality that insulates a bounded context and handles interaction with foreign systems or contexts

# References

## Books

Domain-Driven Design http://amzn.to/1kstiRg
Implementing Domain-Driven Design http://amzn.to/1dgYRY3

## Web

"Getting Started with DDD when Surrounded by Legacy Systems II"
Eric Evans, domainlanguage.com/newsletter/2012-03

Udi Dahan's Domain Events: bit.ly/DomainEventsSalvation

On Pluralsight:
Entity Framework in the Enterprise – bit.ly/PS-EFEnterprise
SOLID Principles of OO Design - bit.ly/solid-smith
Introducing SignalR - bit.ly/IntroSignalR
Design Patterns Library - bit.ly/DesignPatternsLibrary (Façade, Adapter)
Inversion of Control - bit.ly/PluralsightIOC

# Thanks!


**Julie Lerman**

**TheDataFarm.com**

**Twitter: @julielerman**

**Steve Smith**

**Ardalis.com**

**Twitter: @ardalis**


***To Teach Is To Learn Twice***